PARTICLE FILTER SLAM WITH TEXTURE MAPPING

Chun-Nien Chan
Department of Electrical and Computer Engineering
University of California, San Diego
chc030@eng.ucsd.edu

1. INTRODUCTION

Simultaneous Localization And Mapping (SLAM) is a challenging topic in robotics and has been researched for a few decades. When building a map from the observations of a robot, a good estimate of the robot's location is necessary. However, a robot needs a consistent and reliable map for localization. SLAM is the computational problem to simultaneously estimates a map of environment and pose of a moving robot relative to that map, without any a-priori information external to the robot except for the observations of the robot. SLAM approaches are capable of building the map online while correcting the errors of its pose estimate as it sensing the surroundings.

Over decades, several approaches for SLAM based on particle filter[1][2], extended Kalman filter[3], and neural networks[4] has been researched. These approaches are designed to build maps in different representations, including landmark based representation, surfels, polygonal mesh, and occupancy grid. Nowadays, they are widely used for applications such as motion planning of robot in unknown environments [5] and employed in self-driving cars, unmanned aerial vehicles, and autonomous underwater vehicles.

In this paper, we propose a solution for SLAM based on particle filter and occupancy grid. We also extend this solution for texture mapping, which project color points from the RGBD sensor onto the occupancy grid in order to color the floor. Our solution is evaluated with real-world odometry data, indoor 2-D laser scans, and RGBD measurements from THOR, a humanoid robot with lidar and Kinect v2 installed. It is able to estimate reliable map and robot trajectory on various dataset in reasonable time.

2. PROBLEM FORMULATION

2.1. Simultaneous Localization And Mapping

The SLAM problem can be described as a probabilistic Markov Chain. Given robot's pose $\mathbf{x}_t$ and control input $\mathbf{u}_t$ at discrete time steps t, the pose in the following time step $t+1$ is a probabilistic function:

$$\begin{aligned} \mathbf{x}_{t+1} &= f(\mathbf{x}_t, \mathbf{u}_t, \mathbf{w}_t) \sim p_f(\cdot|\mathbf{x}_t, \mathbf{u}_t) \\ \mathbf{w}_t &= \text{motion noise} \end{aligned} \tag{1}$$

which is also known as the motion model.

To map the environment, the robot may sense the environment to get observation at each time step. Denote the observation at time t as $\mathbf{z}_t$ and the environment as $\mathbf{m}$, the sensor observations are governed by a probabilistic law:

$$\begin{aligned} \mathbf{z}_t &= h(\mathbf{x}_t, \mathbf{m}, \mathbf{v}_t) \sim p_h(\cdot|\mathbf{x}_t, \mathbf{m}) \\ \mathbf{v}_t &= \text{observation noise} \end{aligned} \tag{2}$$

which is also known as the observation model.

With motion model Eq.1 and observation model Eq.17, SLAM is the problem to determine the environment $\mathbf{m}$ and robot poses $\mathbf{x}_t$ from observations $\mathbf{z}_0, \ldots, \mathbf{z}_t$ and control inputs $\mathbf{u}_0, \ldots, \mathbf{u}_{t-1}$ at each time step t. The objective to compute can be written in probabilistic form:

$$p(\mathbf{m}, \mathbf{x}_t|\mathbf{z}_{0:t}, \mathbf{u}_{0:t-1}) \tag{3}$$

The relation between $\mathbf{m}$ and $\mathbf{x}_t$ is difficult to determine. However, we can take advantages of the decomposition of the joint probability density function according to the Markov assumptions:

$$\begin{aligned} p(\mathbf{x}_{0:t}, \mathbf{z}_{0:t}, \mathbf{u}_{0:t-1}, \mathbf{m}) =& p_{0|0}(\mathbf{x}_0, \mathbf{m}) \\ & \prod_{i=1}^{t} p_h(\mathbf{z}_i|\mathbf{x}_i, \mathbf{m}) \\ & \prod_{i=1}^{t} p_f(\mathbf{x}_i|\mathbf{x}_{i-1}, \mathbf{u}_{i-1}) \end{aligned} \tag{4}$$

In practical implementations, maximum likelihood estimation (MLE) is used to find the optimal $\mathbf{x}_{0:t}$ and $\mathbf{m}$ in order to determine the poses of robot and the environment. The formulation can be written as:

$$\begin{aligned} &\max_{\mathbf{x}_{0:t}, \mathbf{m}} \sum_{i=0}^{t} \log(p_h(\mathbf{z}_i|\mathbf{x}_i, \mathbf{m})) \\ &+ \sum_{i-=1}^{t} \log(p_f(\mathbf{x}_t|\mathbf{x}_{t-1}, \mathbf{u}_{t-1})) \end{aligned} \tag{5}$$

2.2. Bayes Filtering

Bayes filtering is a probabilistic inference technique for estimating the state $\mathbf{x}_t$ of dynamical systems, like robot, that combines evidence from control inputs and observations using the Markov assumptions and Bayes rule. The Bayes filter relies on two steps to keep track of $p_{t|t}(\mathbf{x}_t)$ and $p_{t+1|t}(\mathbf{x}_{t+1})$

2.2.1. Prediction Step

Given a prior density $p_{t|t}$ over $\mathbf{x}_t$ and the control input $\mathbf{u}_t$, we use the motion model p_f to compute the predicted density $p_{t+1|t}$ over $\mathbf{x}_{t+1}$ as the following equation:

$$p_{t+1|t}(\mathbf{x}) = \int p_f(\mathbf{x}|\mathbf{s}, \mathbf{u}_t) p_{t|t}(\mathbf{s}) d\mathbf{s} \tag{6}$$

2.2.2. Update Step

Given the predicted density $p_{t+1|t}$ over $\mathbf{x}_{t+1}$ and the measurement $\mathbf{z}_{t+1}$, we use the observation model p_h to incorporate the measurement information and obtain the posterior $p_{t+1|t+1}$ over $\mathbf{x}_{t+1}$ as the following equation:

$$p_{t+1|t+1}(\mathbf{x}) = \frac{p_h(\mathbf{z}_{t+1}|\mathbf{x}) p_{t+1|t}(\mathbf{x})}{\int p_h(\mathbf{z}_{t+1}|\mathbf{s}) p_{t+1|t}(\mathbf{s}) d\mathbf{s}} \tag{7}$$

2.3. Occupancy Grid Map

Occupancy grid mapping aims to address the problem of generating a map of the environment $\mathbf{m}$ from noisy and uncertain sensor observations $\mathbf{z}$ with the assumption that the poses of the robot $\mathbf{x}$ is known. The idea of occupancy grid map is dividing the environment into a regular grid with n cells, which are called pixels in 2D, in order to represent it as a vector $\mathbf{m} \in \mathbb{R}^n$. The i_{th} cell of vector $\mathbf{m}$ is either $\mathbf{m}_i = -1$ (free) or $\mathbf{m}_i = 1$ (occupied). The goal of occupancy grid mapping is to estimate the posterior probability over time:

$$p(\mathbf{m}|\mathbf{z}_{0:t}, \mathbf{x}_{0:t}) \tag{8}$$

In standard approaches, an assumption that the occupancy grid cells (pixels) are independent conditioned on the robot trajectory is used to break down the problem. Therefore, the distribution of each cell $\mathbf{m}_i$ can be modeled individually:

$$p(\mathbf{m}|\mathbf{z}_{0:t}, \mathbf{x}_{0:t}) = \prod_i p(\mathbf{m}_i|\mathbf{z}_{0:t}, \mathbf{x}_{0:t}) \tag{9}$$

In our implementation, the occupancy grid map is a 2D matrix $\mathbf{m} \in \mathbb{R}^{M \times N}$, where $M = ceil((x_{max} - x_{min})/resolution + 1)$ and $N = ceil((y_{max} - y_{min})/resolution + 1)$. The coordinates in the world frame can be transformed to the indices in the pixel frame with following equation:

$$\begin{aligned} i_p &= \text{ceil}((x_w - x_{min})/resolution) \\ j_p &= \text{ceil}((y_w - y_{min})/resolution) \end{aligned} \tag{10}$$

2.4. Humanoid THOR Configuration

As described in the introduction, our goal is to design an SLAM solution for THOR, a humanoid robot with RGBD camera (Kinect v2), 2D lidar, and IMU installed. Fig.1 shows the configuration of THOR. At each time step t, the exact rotation angle of robot's neck motor and head motor are provided and there is no need to estimate these values.

The odometry at time t contains three entries $\mathbf{u}_t = [\Delta x_t, \Delta y_t, \Delta\theta_t]^T \in \mathbb{R}^3$, stands for the difference of position and heading between $\mathbf{x}_{t+1}$ and $\mathbf{x}_t$.

The lidar scan data is a vector $L \in \mathbb{R}^{1081}$, which is the range information in meters of a semicircular field from $-135°$ to $135°$. The maximum detection range is 30 meters, so all the values in L greater than 30 can be treated as no-hit and discarded.

The THOR Humanoid robot is installed with Kinect, which provides RGBD images at each time step. The RGBD data contains two images: color (RGB) image and depth (IR) image. The color image is matrix $M \in \mathbb{R}^{960 \times 540 \times 3}$. The depth image is a matrix $D \in \mathbb{R}^{512 \times 424}$ where each pixel is the depth value in milimeters.

The configuration of THOR is shown as follows:

- **Center of Mass**: Fixed at 0.93 meters
- **Head**: 0.33 meters above Center of Mass
- **Lidar**: 0.15 meters above Head
- **Kinect**: 0.07 meters above Head

All the sensors mentioned above worked independently, thus each data from the sensor is provided with a timestamp. Post-processing is necessary to align data from different sensors with their timestamps to form discrete time steps.

2.5. Texture Mapping

Given a occupancy grid map $\mathbf{m}$, the goal of texture mapping is to form a vector $\mathbf{m}_c \in \mathbb{R}^{n \times 3}$ where each cell in the vector is a RGB (three channel) floor color corresponding to the cell in the occupancy grid map.

3. TECHNICAL APPROACH

3.1. Motion & Observation Model

We define the pose of the robot at time t is as a vector $\mathbf{x}_t = [x_t, y_t, \theta_t]^T$, where x_t and y_t are the coordinates (in meters) of robot in 2D space with respect to a fixed origin, and θ_t is the heading (in radians) of the robot. Since the odometry contains the difference between position and heading angle at time t and $t-1$, the odometry-based motion model can be defined as an additive model:

$$\begin{aligned} \mathbf{x}_{t+1} &= f(\mathbf{x}_t, \mathbf{u}_t, \mathbf{w}_t) \\ &= \mathbf{x}_t + \mathbf{u}_t + \mathbf{w}_t,\ \mathbf{w}_t \sim \mathcal{N}(0, \mathbf{W}) \end{aligned} \tag{11}$$

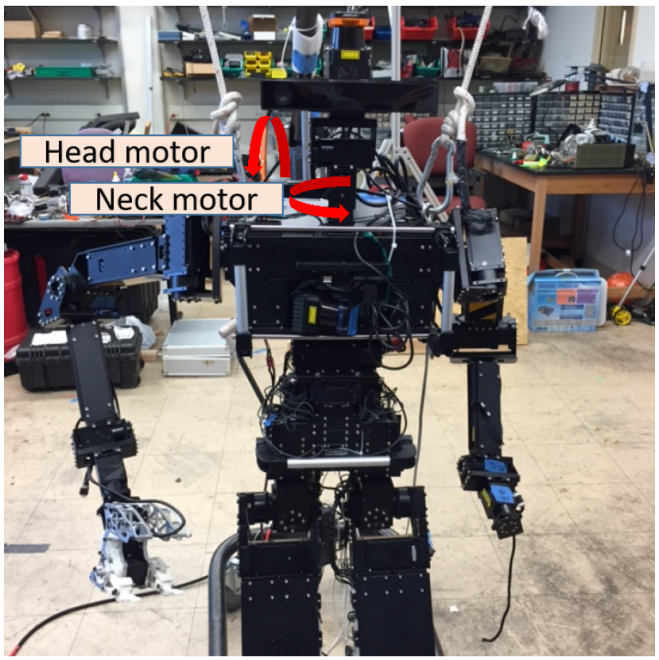


Fig. 1. THOR Configuration

where $\mathbf{W} \in \mathbb{R}^{3\times3}$ is the variance of Gaussian noise.

There are a variety of observation models for lidar used in researches, including laser range-azimuth-elevation model, and laser beam model. In this paper, we choose laser correlation model as the observation model for our problem to work with the occupancy grid map. The pdf $p_h(\mathbf{z}_t|\mathbf{x}_t, \mathbf{m})$ by comparing the correlation between map $\mathbf{m}$ and the lidar scan points in world frame $\mathbf{y}_t$, where $\mathbf{y}_t$ can be obtained by transforming observation $\mathbf{z}_t$ in the lidar frame to the world frame:

$$\mathbf{y}'_{i,t} = {}_w\mathbf{T}_b \cdot {}_b\mathbf{T}_h \cdot {}_h\mathbf{T}_l \cdot \mathbf{z}'_{i,t} \tag{12}$$

where $\mathbf{z}'_{i,t} = [\mathbf{z}^T_{i,t}, 1]^T$ and $\mathbf{y}'_{i,t} = [\mathbf{y}^T_{i,t}, 1]^T$.

The observation in lidar frame $\mathbf{z}_{i,t}$ is obtained by transforming lidar scan data L_I with the following equation:

$$\begin{aligned} \mathbf{z}_{i,t} &= [L_i \cos(\phi_i), L_i \sin(\phi_i), 0]^T \\ \phi_i &= -135^\circ + (i \cdot \frac{270}{1081})^\circ \end{aligned} \tag{13}$$

The transform matrix ${}_w\mathbf{T}_b$ and ${}_b\mathbf{T}_h$ are acquired from the pose $\mathbf{x}_t$, the neck angle, and the head angle of the robot at time t with the configuration of THOR. They are defined as following equations:

$${}_h\mathbf{T}_l = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0.15 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{14}$$

$$\begin{aligned} {}_b\mathbf{R}_h &= \text{Yaw}(NeckAngle_t) \\ &\quad \cdot \text{Pitch}(HeadAngle_t) \\ {}_b\mathbf{T}_h &= \begin{bmatrix} {}_b\mathbf{R}_h & \begin{matrix} 0 \\ 0 \\ 0.15 \end{matrix} \\ \begin{matrix} 0 & 0 & 0 \end{matrix} & 1 \end{bmatrix} \end{aligned} \tag{15}$$

$$\begin{aligned} {}_w\mathbf{R}_b &= \text{Yaw}(theta_t) \\ {}_w\mathbf{T}_b &= \begin{bmatrix} {}_w\mathbf{R}_b & \begin{matrix} x_t \\ y_t \\ 0.93 \end{matrix} \\ \begin{matrix} 0 & 0 & 0 \end{matrix} & 1 \end{bmatrix} \end{aligned} \tag{16}$$

The observation model is calculated by converting the correlation between $\mathbf{m}$ and $\mathbf{y}_t$ to probabilities with softmax function:

$$p_h(\mathbf{z}_t|\mathbf{x}_t, \mathbf{m}) = \frac{\exp(\text{corr}(\mathbf{y}_t, \mathbf{m}))}{\sum_{v_t} \exp(\text{corr}(\mathbf{v}_t, \mathbf{m}))} \tag{17}$$

where the correlation function $\text{corr}(\mathbf{y}_t, \mathbf{m})$ is defined as follows:

$$\text{corr}(\mathbf{y}, \mathbf{m}) = \sum_i \mathbf{1}\{\mathbf{m}_i = \mathbf{y}_i\} \tag{18}$$

3.2. Occupancy Grid Mapping

As what we mention in Sec.2.3, the environment is represented as a vector $\mathbf{m} \in \mathbb{R}^n$, with cells independent to each other. In this section, we introduce an algorithm to maintain the pdf $p(\mathbf{m}|\mathbf{z}_{0:t}, \mathbf{x}_{0:t})$ over time steps t.

In our problem, the cells $\mathbf{m}_i$ are modeled as Bernoulli random variables where:

$$\mathbf{m}_i = \begin{cases} \text{Occupied}(1) & \text{with prob. } \gamma_{i,t} = p(\mathbf{m}_i = 1|\mathbf{z}_{0:t}, \mathbf{x}_{0:t}) \\ \text{Free}(-1) & \text{with prob. } 1 - \gamma_{i,t} \end{cases} \tag{19}$$

and the goal of the mapping algorithm is to keep a vector of occupancy probabilities γ_t over time. Since the occupancy probabilities follow the Bayes rule $\gamma_{i,t} = \frac{1}{\eta_t} p_h(\mathbf{z}_t|\mathbf{m}_i = 1, \mathbf{x}_t)\gamma_{i,t-1}$, we can update the map by accumulating log-odds ratio $\lambda_{i,t}$ of the binary random variable $\mathbf{m}_i$ over time:

$$\begin{aligned} \lambda_{i,t} &= \lambda(\mathbf{m}_i|\mathbf{z}_{0:t}, \mathbf{x}_{0:t}) = \log \frac{p(\mathbf{m}_i = 1|\mathbf{z}_{0:t}, \mathbf{x}_{0:t})}{p(\mathbf{m}_i = -1|\mathbf{z}_{0:t}, \mathbf{x}_{0:t})} \\ &= \log \frac{p(\mathbf{z}_t|\mathbf{m}_i = 1, \mathbf{x}_{0:t})}{p(\mathbf{z}_t|\mathbf{m}_i = -1, \mathbf{x}_{0:t})} \frac{\gamma_{i,t-1}}{1 - \gamma_{i,t-1}} \\ &= \lambda(\mathbf{m}_i|\mathbf{z}_{0:t-1}, \mathbf{x}_{0:t-1}) + \log g_h(\mathbf{z}_t|\mathbf{m}_i, \mathbf{x}_t) \\ &= \lambda(\mathbf{m}_i) + \sum_{s=0}^{t} \log g_h(\mathbf{z}_s|\mathbf{m}_i, \mathbf{x}_s) \end{aligned} \tag{20}$$

where $g_h(\mathbf{z}_s|\mathbf{m}_i, \mathbf{x}_s)$ determines the reliability of the observation.

Since the last equation contains a summation of $g_h(\mathbf{z}_t|\mathbf{m}_i, \mathbf{x}_t)$ over time, $\lambda_{i,t}$ can be obtained by adding the log-odds ratio to $\lambda_{i,t-1}$ at each time step t:

$$\begin{aligned}\lambda_{i,t} &= \lambda_{i,t-1} + \log g_h(\mathbf{z}_t|\mathbf{m}_i, \mathbf{x}_t) \\ &= \lambda_{i,t-1} + \Delta\lambda_{i,t-1}\end{aligned} \tag{21}$$

The observation log-odds ratio $\log g_h(\mathbf{z}_s|\mathbf{m}_i, \mathbf{x}_s)$ determines how much we want to trust on the observation. It can be simplified as:

$$\begin{aligned}\Delta\lambda_{i,t-1} &= \log g_h(\mathbf{z}_t|\mathbf{m}_i, \mathbf{x}_t) \\ &= \log\frac{p(\mathbf{m}_i = 1|\mathbf{z}_t, \mathbf{x}_t)}{p(\mathbf{m}_i = -1|\mathbf{z}_t, \mathbf{x}_t)} - \lambda(\mathbf{m}_i)\end{aligned} \tag{22}$$

In our implementation, the first term is a constant over time for simplicity, and the prior occupancy log-odds ratio $\lambda(\mathbf{m}_i)$ is set to 0 due to the lack of prior information of the environment. We define two constants $\Delta\lambda_{occupied}$ and $\Delta\lambda_{free}$ to update cells which are considered occupied and free. At each time step t, we transform the lidar scan points to the pixel coordinates, every cell that contains a lidar scan point will be marked as "occupied" and the corresponding $\lambda_{i|t+1}$ will then be added $\Delta\lambda_{occupied}$. Similarly, cells and corresponding log-odds between robot pose (state) and lidar scan points will be marked as "free" and added $\Delta\lambda_{free}$. `cv2.drawContours` is used to fill values to all the cells in the semicircular lidar scan field.

Besides, lower-bound λ_{MIN} and upper-bound λ_{MAX} are defined to clip the value of $\lambda_{i,t}$ in order to prevent over confident estimation and influence of outlier in the observations.

After the log-odds of cells are estimated, the cell occupancy probability $\gamma_{i,t}$ can be obtained from the log-odds value:

$$\gamma_{i,t} = 1 - \frac{1}{1 + \exp(\lambda_{i,t})} \tag{23}$$

3.3. Particle Filter

Particle filter methods are a set of Monte Carlo algorithms used to solve filtering problems. It uses a set of particles to represent the posterior distribution of some stochastic process given noisy or partial observations. The particles are represented as the form of below:

$$\{\mu^{(k)}, \alpha^{(k)}\} \text{ for } k = 1, ..., N \tag{24}$$

And the particle filter uses a mixture of delta function:

$$\delta(x; \mu) := \begin{cases} 1 & x = \mu^{(k)} \\ 0 & \text{else} \end{cases} \text{ for } k = 1, ..., N \tag{25}$$

with weights $\alpha^{(k)}$ to represent the prior $p_{t|t}$ as follow:

$$p(\mathbf{x}_t|\mathbf{z}_{0:t}, \mathbf{u}_{0:t-1}) = p_{t|t}(\mathbf{x}_t) = \sum_{k=1}^{N_{t|t}} \alpha_{t|t}^{(k)} \delta(\mathbf{x}_t; \mu_{t|t}^{(k)}) \tag{26}$$

The delta mixture pdfs will be substituted in the Bayes filter prediction and update steps to derive the filter. The update and prediction of particle filters are in an approximate manner. The samples from the distribution are represented by a set of particles; each particle has a likelihood weight to represent the probability of that particle being sampled from the probability density function.

3.3.1. Prediction Step

In the Bayes filter, the prediction step is calculated with the prior $p_{t|t}$ over $\mathbf{x}_t$, the control input $\mathbf{u}_t$, and the motion model p_f to compute the predicted density $p_{t+1|t}$ over $\mathbf{x}_{t+1}$ as the Eq.(6).

The prediction steps in particle filter should maintain the mixture-of-delta functions form of the pdfs. So we substitute the prior from Eq.(26) into the prediction step of Bayes filter, i.e., Eq.(6) as follow:

$$\begin{aligned}p_{t+1|t}(x) &= \int p_f(\mathbf{x}|\mathbf{s}, \mathbf{u}_t) \sum_{k=1}^{N_{t|t}} \alpha_{t|t}^{(k)} \delta(\mathbf{s}; \mu_{t|t}^{(k)}) d\mathbf{s} \\ &= \sum_{k=1}^{N_{t|t}} \alpha_{t|t}^{(k)} p_f(\mathbf{x}|\mu_{t|t}^{(k)}, u_t)\end{aligned} \tag{27}$$

Since $p_{t+1|t}(\mathbf{x})$ is a mixture pdf with components $p_f(\mathbf{x}|\mu_{t|t}^{(k)}, \mathbf{u}_t)$, we may approximate it with particles by drawing samples from it as follows:

$$\begin{aligned}\mu_{t+1|t}^{(k)} &\sim p_f(\cdot|\mu_{t|t}^{(k)}, \mathbf{u}_t) \\ \alpha_{t+1|t}^{(k)} &= \alpha_{t|t}^{(k)}\end{aligned} \tag{28}$$

As the motion model shown in 1, we update every practicle with the following equation in our implementation:

$$\mu_{t+1|t}^{(k)} = \mu_{t|t}^{(k)} + \mathbf{u}_t + \mathbf{w}_t, \ \mathbf{w}_t \sim \mathcal{N}(0, \mathbf{W}) \tag{29}$$

where $\mathbf{w}_t$ is a 3-D Gaussian motion noise.

Theoretically, the approximation of the prediction step $p_{t+1|t}(\mathbf{x})$ can be obtained as follow:

$$\sum_{k=1}^{N_{t|t}} \alpha_{t|t}^{(k)} p_f(\mathbf{x}|\mu_{t|t}^{(k)}, \mathbf{u}_t) \approx \sum_{k=1}^{N_{t+1|t}} \alpha_{t+1|t}^{(k)} \delta(\mathbf{x}; \mu_{t+1|t}^{(k)}) \tag{30}$$

In our implementation, we use the particle with maximum weight $\alpha_{t+1|t}$ to represent the prediction state of the robot. It can be written as:

$$\mathbf{x}'_{t+1|t} = \mu_{t+1|t}^{(k)}, \ k = \text{argmax}(\alpha_{t+1|t}^{(k)}) \tag{31}$$

3.3.2. Update Step

In the Bayes filter, the update step of the posterior $p_{t+1|t+1}$ over $\mathbf{x}_{t+1}$ is calculated with the predicted density $p_{t+1|t}$ over

$\mathbf{x}_{t+1}$, the measurement $\mathbf{z}_{t+1}$, and the observation model p_h, to incorporate the measurement information as the Eq. (7).

We evaluate Bayes rule with the predicted delta-mixture pdf. So we substitute the predicted density from Eq.(31) into the posterior of Bayes filter, i.e., Eq.(7) as follow:

$$\begin{aligned} p_{t+1|t+1}(\mathbf{x}) &= \frac{p_h(\mathbf{z}_{t+1}|\mathbf{x}) \sum_{k=1}^{N_{t+1|t}} \alpha_{t+1|t}^{(k)} \delta(\mathbf{x}; \mu_{t+1|t}^{(k)})}{\int p_h(\mathbf{z}_{t+1}|\mathbf{s}) \sum_{j=1}^{N_{t+1|t}} \alpha_{t+1|t}^{(j)} \delta(\mathbf{s}; \mu_{t+1|t}^{(j)}) d\mathbf{s}} \\ &= \sum_{k=1}^{N_{t+1|t}} \left[\frac{\alpha_{t+1|t}^{(k)} p_h(\mathbf{z}_{t+1}|\mu_{t+1|t}^{(k)})}{\sum_{j=1}^{N_{t+1|t}} \alpha_{t+1|t}^{(j)} p_h(\mathbf{z}_{t+1}|\mu_{t+1|t}^{(j)})} \right] \\ &\quad \delta(\mathbf{x}; \mu_{t+1|t}^{(k)}) \end{aligned} \tag{32}$$

The resulting pdf turns out to be a delta mixture, so no approximation is necessary. The update step does not update the particle positions, but only their weights.

In our implementation, the probability $p_h(z_{t+1}|\mu_{t+1|t}^{(k)})$ is obtained via Eq.(17), with lidar scan $\mathbf{z}_t$ first transformed to the world frame using the state of each particle $\mu^(k)_{t+1|t}$ and Eq.(12).

In addition to the traditional update step for the particle filter mentioned above, we also update particles with local maximum correlation. We compute the map correlation with not only the state of particle $\mu^(k)_{t+1|t}$ itself, but also other 24 points in the 5x5 grid (add $i \times mapResolution$ to x and $j \times mapResolution$ for y, and keep the heading angle $theta$) near the particle. Thus, we will have 25 correlation values for each particle. Finally, we replace the state of each particle with the point in the 5x5 grid with largest correlation values.

3.3.3. Particle Resampling

To avoid weight collapse caused by weight disparity, the resampling steps are applied, and the particle with negligible weights are replaced by new particles in the proximity of the particles with higher weights.

Given a weighted particle set, resampling creates a new particle set with equal weights by adding many particles to the locations that had high weight and few particles to the locations that had low weights. It focuses the representation power of the particles to likely regions, while leaving unlikely regions with only few particles. It is applied at time t if the effective number of particles is less than a threshold, which is represented as the following inequation:

$$N_{eff} = \frac{1}{\sum_{k=1}^{N} (\alpha_{t|t}^{(k)})^2} \leq N_{threshold} \tag{33}$$

3.4. SLAM

Our SLAM solution combines particle filer and updating occupancy grid map. The pesudo code for the algorithm is shown in Algorithm.1.

Initialize occupancy log-odds
Initialize particles
forall t *in* $0 \dots T$ **do**
  Transform lidar scan points $\mathbf{z}_t$ to body frame
  Particle filter: Update Step:
    Transform lidar scan points to world frame with respect to the state (pose) of each particle to get $\mathbf{y}_t^{(k)}$
    Calculate map correlation $\text{corr}^{(k)}(\mathbf{y}^(k)_t, \mathbf{m}_t)$
    Update particles $\mu_{t|t}$ to their local maximum map correlation
    Update particle weights $\alpha_{t|t}^{(k)}$
    if $N_{eff} \leq N_{threshold}$ **then**
      Particle Filter*: Resampling*
    end
  Occupancy Map*: Update log-odds*
    Choose particle with maximum weight $\alpha_{t|t}$
    Transform lidar scan points to its world frame $\mathbf{y}_t$
    Update log-odds
    Particle filter*: Prediction Step*
end

Algorithm 1: Particle Filter SLAM

3.5. Texture Mapping

The first step of texture mapping is to do the calibration and undistortion[6] on color and depth image since the RGBD data from Kinect is not well calibrated. The distortion coefficients and intrinsic matrix for both color and depth images are provided in the dataset. In our implementation, we use `cv2.undistort` to undistort image with its corresponding distortion coefficient.

Then, we align color and depth images to create a point cloud from the given images in the Kinect frame with the method proposed in [7]. In addition to the intrinsic matrix, this algorithm also needs the extrinsic provided in the dataset. The output of this algorithm is a matrix $P \in \mathbb{R}^{N \times 6}$, where N is the number of pixels in the aligned image and each row in the matrix contains the information of x, y, z coordinates in the Kinect frame and its corresponding color in RGB.

After retrieving the point cloud in Kinect frame, we transform the coordinates of these points to the world frame and map it to the occupancy grid map. The following is the transformation equation:

$$\mathbf{y}_i = {}_w\mathbf{T}_b \cdot {}_b\mathbf{T}_h \cdot {}_h\mathbf{T}_k \cdot {}_r\mathbf{T}_o \cdot [x_i, y_i, z_i, 1]^T \tag{34}$$

where ${}_h\mathbf{T}_k$ is the transformation matrix from Kinect frame to head frame and ${}_r\mathbf{T}_o$ is the transform matrix from optical

frame to regular frame:

$$ {}_h\mathbf{T}_k = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0.07 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{35} $$

$$ {}_r\mathbf{T}_o = \begin{bmatrix} 0 & 0 & 1 & 0 \\ -1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{36} $$

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

Since the algorithm only map the color to the floor cells. We filter out points with $z_w \geq 0.1$ or $z_w \leq -0.1$ and those mapped to occupied cell.

Due to the observation noise, defect in calibration, and changes in the environment as time goes by, the mapping from the color and depth value to the transformed coordinates in world frame may not be consistent across time. To address this problem, we use moving average to update the color of the texture map. When updating color of pixel pi at time t with value v, the following equation is used:

$$ p_{t+1,i,j} = 0.9 \cdot p_{t,i,j} + 0.1 \cdot v_j, \ j \in \{R, G, B\} \tag{37} $$

4. RESULTS AND DISCUSSION

We test our SLAM algorithm with 5 data set, all of them are collected by the same THOR robot and three of them contains the RGBD data from Kinect. The hyper-parameters used in our algorithm are same for all the data set except for the grid size and resolution. We want it to match the real-world scenarios that this algorithm runs online and not able to tune parameters for every unknown environment. The hyper-parameters are listed in Table.1.

The results of particle filter SLAM are shown in Fig.2-6. These figures show the occupancy probability γ. The white pixels are free, and black pixels are occupied. The rainbow lines in the figures are the trajectories of the robot. The trajectory is plotted in "gist_rainbow" color map, where red part is the beginning of the trajectory and purple part is the end. The texture mapping are shown in Fig.7-9. Moreover, please refer to the video in the code submission and see the SLAM and texture mapping animation.

4.1. Image Calibration and Undistortion for Texture Mapping

We have tested the texture mapping with and without undistortion. The results are shown in 10-11 The result shows that undistort the depth and color images before doing the texture mapping always offers better results. The points in the RGBD alignment without undistortion are mapped to the pixel frame awfully.

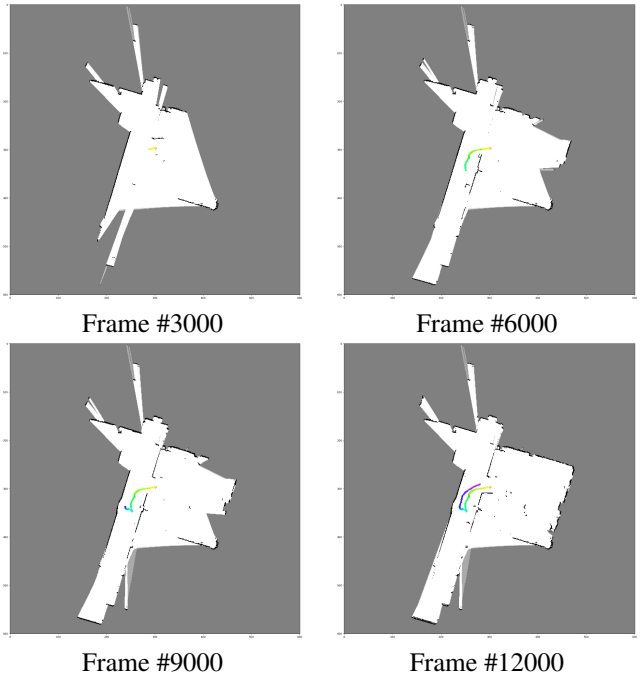


Fig. 2. Trajectory and Occupancy Map: TestCase 0

Even though the proposed texture mapping works on the testing data set, there are still some defects in the texture mapping. For example, there are pepper noise and dark strip on the robot trajectory. The reason may be from the noise of observation, quality of estimated distortion coefficients, and the lack of image calibration.

In theory, calibration on depth and color image with their intrinsics will also provide better results. However, our testing results with calibration are worse than the one without calibration. The bad results may originate from the bad calibration matrix. Recently, there are several researches on calibration and alignment approaches for Kinect[8][9][10]. These approaches may address this problem and further improve the quality of texture mapping.

5. REFERENCES

[1] Michael Montemerlo, Sebastian Thrun, Daphne Koller, Ben Wegbreit, et al., "Fastslam: A factored solution to the simultaneous localization and mapping problem," *Aaai/iaai*, vol. 593598, 2002.

[2] Michael Montemerlo, Sebastian Thrun, Daphne Koller, Ben Wegbreit, et al., "Fastslam 2.0: An improved particle filtering algorithm for simultaneous localization and mapping that provably converges," in *IJCAI*, 2003, pp. 1151–1156.

Parameter	Description	Value
λ_{belief}	Belief of the lidar observation	0.8
$\Delta\lambda_{occupied}$	Difference added when updating occupied cell in occupancy log-odds	log(belief / (1-belief))
$\Delta\lambda_{free}$	Difference added when updating free cell in occupancy log-odds	-0.5 * log(belief / (1-belief))
λ_{Max}	Maximum value of the occupancy grid log-odds	100
λ_{Min}	Maximum value of the occupancy grid log-odds	-100
$N_{particles}$	Number of particles in particle filter	128
$N_{threshold}$	N_{eff} threshold for particle filter resampling	4
$\mathbf{W}_{predict}$	Gaussian noise variance for motion model in particle filter prediction step	diag($[10^{-3}, 10^{-3}, 10^{-3}]$)
$\lambda_{occupied\_threshold}$	Threshold for log-odds value to be considered as a occupied cell	log(9)

Table 1. Hyper-parameters for particle filter SLAM

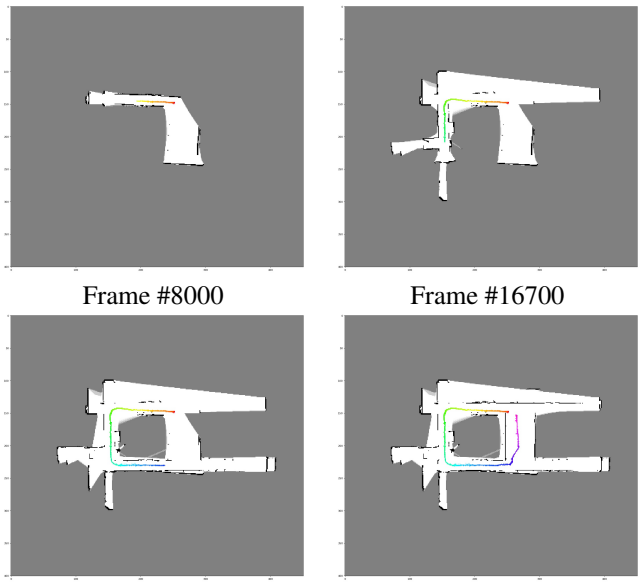


Fig. 3. Trajectory and Occupancy Map: TestCase 1

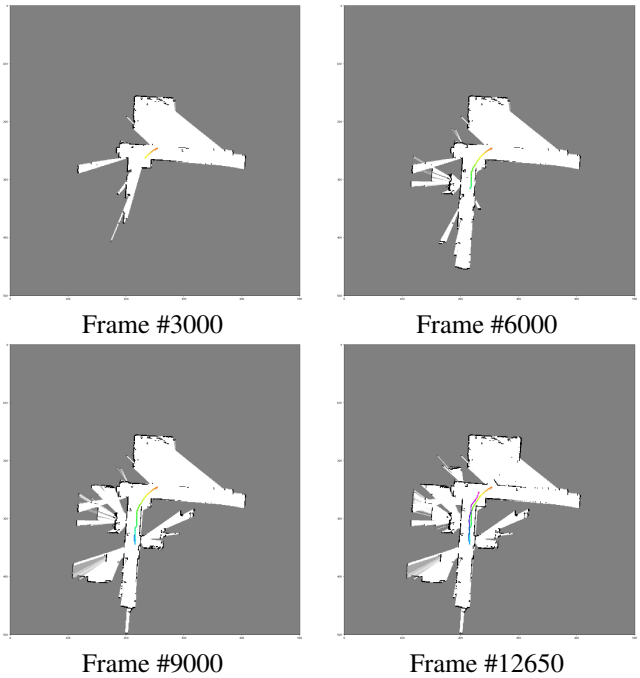


Fig. 4. Trajectory and Occupancy Map: TestCase 2

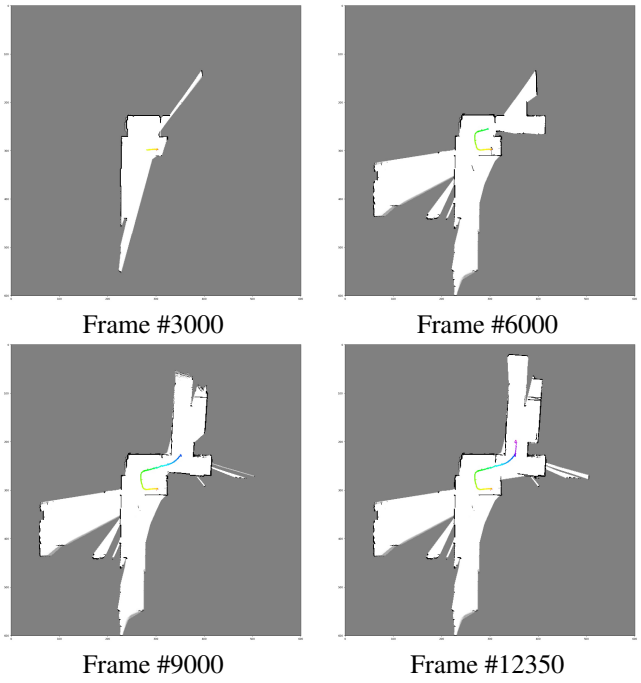


Fig. 5. Trajectory and Occupancy Map: TestCase 3

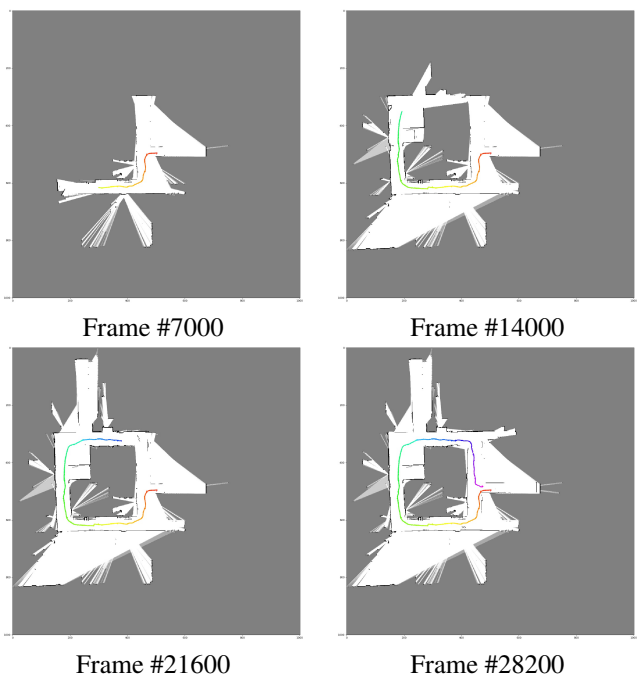


Fig. 6. Trajectory and Occupancy Map: TestCase 4

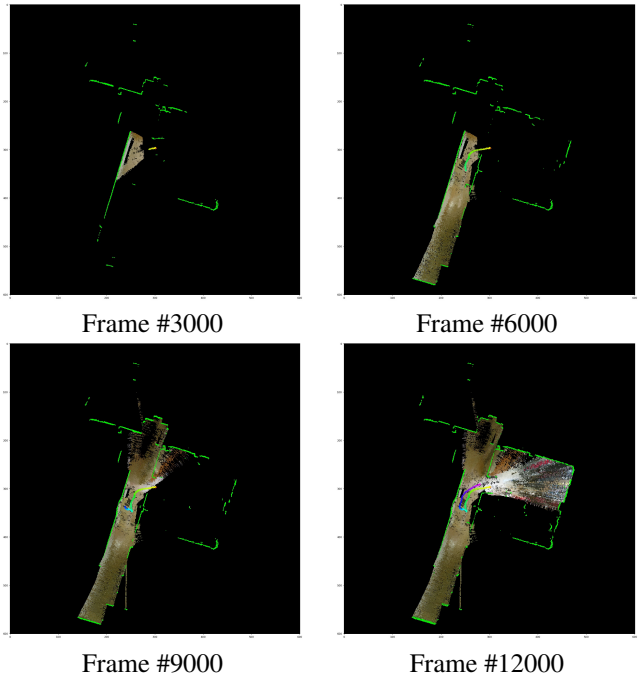


Fig. 7. Textured Map: TestCase 0

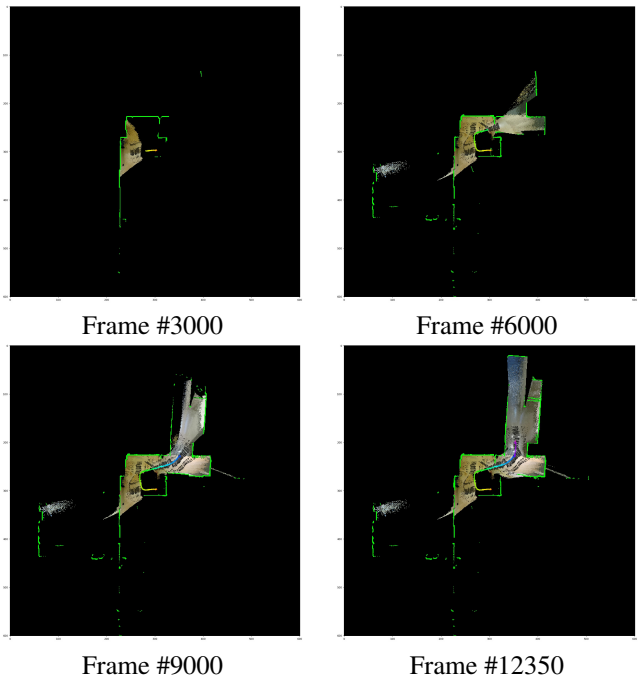
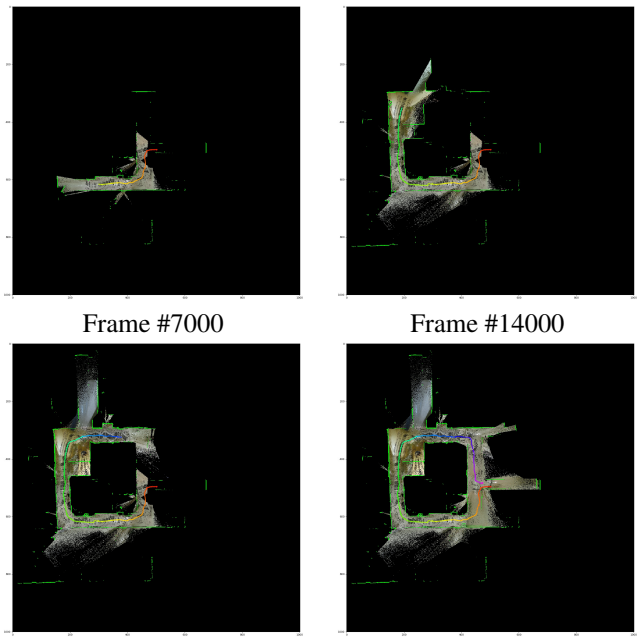


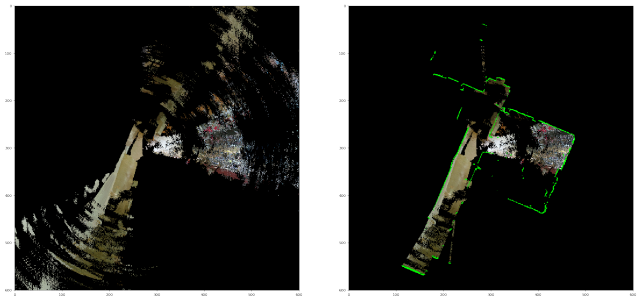
Fig. 8. Textured Map: TestCase 3



Frame #7000 Frame #14000

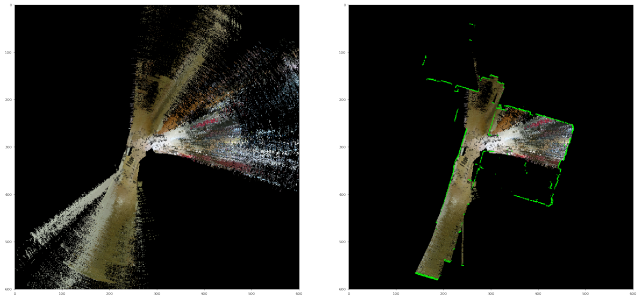
Frame #21600 Frame #28200

Fig. 9. Textured Map: TestCase 4



Raw texture mapping Filter with occupancy map

Fig. 10. Texture mapping without undistortion



Raw texture mapping Filter with occupancy map

Fig. 11. Texture mapping with undistortion

[3] Tim Bailey, Juan Nieto, Jose Guivant, Michael Stevens, and Eduardo Nebot, "Consistency of the ekf-slam algorithm," in *2006 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2006, pp. 3562–3568.

[4] Keisuke Tateno, Federico Tombari, Iro Laina, and Nassir Navab, "Cnn-slam: Real-time dense monocular slam with learned depth prediction," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017, pp. 6243–6252.

[5] Beril Sirmaccek, Nicolò Botteghi, and Mustafa Khaled, "Reinforcement learning and slam based approach for mobile robot navigation in unknown environments," in *ISPRS Workshop Indoor 3D 2019*, 2019.

[6] Jean-Yves Bouguet, "Description of the calibration parameters," 2015.

[7] Mehran Maghoumi, "Align depth and color frames – depth and rgb registration," 2016.

[8] Diana Pagliari and Livio Pinto, "Calibration of kinect for xbox one and comparison between the two generations of microsoft sensors," *Sensors*, vol. 15, no. 11, pp. 27569–27589, 2015.

[9] Daniel Herrera, Juho Kannala, and Janne Heikkilä, "Joint depth and color camera calibration with distortion correction," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 34, no. 10, pp. 2058–2064, 2012.

[10] Changhee Kim, Seokmin Yun, Seung-Won Jung, and Chee Sun Won, "Color and depth image correspondence for kinect v2," in *Advanced Multimedia and Ubiquitous Engineering*, pp. 111–116. Springer, 2015.