Particle Filter SLAM

Saurabh Himmatlal Mirani (A53319557)
Sensing and Estimation in Robotics
University of California, San Deigo
smirani@ucsd.edu

I. Introduction

Recent advances in autonomous mobile robots has allowed robots to be involved in many applications including planetary exploration, search and rescue operations etc. For accomplishing a generic task, it is important for the robot to autonomously navigate, use the information acquired through various sensors and a model of the surrounding environment. This problem of estimating both the robot pose and the environment representation at the same time is usually defined as Simultaneous Localization And Mapping (SLAM). SLAM problem has become one of the most important research area of robotics, and the key to realize autonomous navigation. Localization needs the information of the environment map, while mapping relies on the robot's position and pose. Due to the accuracy and cost of sensors as well as the unknown environment, both localization and mapping problems are challenging, which call for the technique for localization and mapping at the same time [1].

SLAM addresses the problem of building a map of the environment using sensor data obtained from a mobile robot. The mobile robot is subject to error, hence the mapping problem creates a robot localization problem; hence the name SLAM. The ability to simultaneously localize a robot and accurately map its environment is considered by many to be a key prerequisite of truly autonomous robots [2].

In this paper, SLAM is implemented using Particle Filter [3] [2]. Particle Filter maintains a set of particles to represent the posterior distribution of some stochastic process given some noisy and/or partial observations. The main advantage of using Particle Filter is that there is no assumption on noise being Gaussian. There are two important steps in Particle Filter based SLAM, (1) Predict & (2) Update. In the predict step, odometry motion model is used to predict the motion of each particle. In the update step, the map is updated based on the lidar scans as seen by the best particle, i.e. updating map based on the observations. The pipeline of the algorithm is as follows:

- Initial particle set $\mu_{0|0}^{(k)} = (0,0,0)^T$ with weights $\alpha_{0|0}^{(k)} = \frac{1}{N}$
- Use the first laser scan to initialize the map:
  1) Convert the scan to Cartesian coordinates
  2) Transform the scan from the lidar frame to the body frame and then to the world frame
  3) Threshold on Z coordinate to remove the ground plane from the scan
  4) Convert the scan to cells using Bresenham Algorithm and update the log-odds
- Use an odometry motion model to predict motion for each particle
- Use the laser scan from each particle to compute map correlation and update the particle weights using softmax function
- Choose the best particle, project the laser scan, and update the map log-odds

The rest of the paper is organized as follows, Section II describes Problem Formulation, Section III describes the Technical Approach, experimental results are reported in section IV, Section V concludes the paper.

II. Problem Formulation

The data received from various sensors and their use are as follows:

- **Encoders and IMU (odometry):** It helps in determining the current pose $(x, y, \theta)^T$ of the robot
- **2D Lidar:** It helps build the map of the environment
- **Kinect (RGBD camera):** It helps produce a texture map

The problem statement for this paper is to build a map of the environment using the 2D lidar scans and simultaneously get the actual trajectory of the robot which using IMU and Encoder data which has noise. The other problem is to generate a texture map of the environment using RGBD camera data.

A. SLAM Problem:

Given a series of controls u_t and sensor observations z_t over discrete time steps t, the SLAM problem is to compute an estimate of the robot's location x_t and a map of the environment m_t. All quantities are usually probabilistic, so the objective is to compute:

$$P(m_{t+1}, x_{t+1} | z_{1:t+1}, u_{1:t})$$

Applying Bayes' rule gives a framework for sequentially updating the location posteriors, given a map and a transition function $P(x_t|x_{t-1})$,

$$P(x_t | z_{1:t}, u_{1:t}, m_t) =$$

$$\sum_{m_{t-1}} P(z_t|x_t, m_t, u_{1:t}) \sum_{x_{t-1}} P(x_t|x_{t-1}) P(x_{t-1}|m_t, z_{1:t-1}, u_{1:t})/\eta$$

Similarly the map can be updated sequentially by

$$P(m_t|x_t, z_{1:t}, u_{1:t}) =$$

$$\sum_{x_t}\sum_{m_t} P(m_t|x_t, m_{t-1}, z_t, u_{1:t})P(m_{t-1}, x_t|z_{1:t-1}, m_{t-1}, u_{1:t})$$

III. Technical Approach

The SLAM problem as described in Section II can be solved using Particle Filter. N particles are initialized with weights 1/N. A mixture of delta functions is used, which is given by,

$$\delta = \begin{cases} 1 & x = \mu^{(k)} \\ 1, & \text{else} \end{cases} \tag{1}$$

for $k = 1, ..., N$

This delta function gives the positions of the particles. $\alpha^{(k)}$ initially for all particles is 1/N. The prior distribution of the particles is:

$$x_t|z_{0:t}, u_{o:t-1} \sim p_{t|t}(x_t) = \sum_{k=1}^{N_{t|t}} \alpha_{t|t}^{(k)} \delta(x_t; \mu_{t|t}^{(k)})$$

where,
x_t: state at time t
$z_{0:t}$: observations from 0 to t
$u_{0:t-1}$: control inputs from 0 to t-1

A. Prediction Step

The prediction step uses the odometry data as provided. The odometry model in terms of time steps is as follows:

For every particle $k = 1, ..., N_{t|t}$, compute

$$\mu_{t+1|t}^{(k)} = f(\mu_{t|t}^{(k)}, u_t + \epsilon_t)$$

where, f is the odometry model

$$s_{t+1} = f(s_t, u_t) = s_t + \begin{pmatrix} \Delta x \\ \Delta y \\ \Delta\theta \end{pmatrix}$$

where s_t is the pose of the robot given by:

$$s_t = (x_t, y_t, \theta_t)^T$$

where x_t,y_t,θ_t are the x and y coordinates and the orientation of the robot respectively, in the world frame.

The prediction step is hence given by,

$$p_{t+1|t}(x) = \int p_f(x|s, u_t) \sum_{k=1}^{N_{t|t}} \alpha_{t|t}^{(k)} \delta(x_t; \mu_{t|t}^{(k)}) ds \tag{2}$$

$$= \sum_{k=1}^{N_{t|t}} \alpha_{t|t}^{(k)} p_f(x|\mu_{t|t}^{(k)}, u_t)$$

B. Update Step

The update equations are given by:

$$p_{t=1|t+1}(x) = \frac{p_h(z_{t+1}|x) \sum_{k=1}^{N_{t|t}} \alpha_{t+1|t}^{(k)} \delta(x; \mu_{t+1|t}^{(k)})}{\int p_h(z_{t+1}|x) \sum_{j=1}^{N_{t|t}} \alpha_{t+1|t}^{(j)} \delta(s; \mu_{t+1|t}^{(j)}) ds} \tag{3}$$

$$\sum_{k=1}^{N_{t|t}} [\frac{\alpha_{t+1|t}^{(k)} p_h(z_{t+1}|\mu_{t+1|t}^{(k)})}{\sum_{j=1}^{N_{t|t}} \alpha_{t+1|t}^{(j)} p_h(z_{t+1}|\mu_{t+1|t}^{(j)})}] \delta(x; \mu_{t+1|t}^{(k)})$$

C. Resampling

Effective number of particles is given by,

$$N_{eff} = \frac{1}{\sum_{k=1}^{N_{t|t}} (\alpha_{t|t}^{(k)})^2}$$

Resampling is applied at time t is the effective number of particles is less than a threshold. If $N_{eff} < N_{thresh}$, then create a new set,$\{\overline{\mu}_{t+1|t}^{(k)}, \overline{\alpha}_{t+1|t}^{(k)}\}$ for $k = 1, ..., N_{t+1|t}$ (usually $N_{t+1|t} = N_{t|t}$) using Stratified Resampling. Pseudo Code for Stratified Resampling

Stratified (low variance) resampling

Input: particle set $\left\{\boldsymbol{\mu}^{(k)}, \alpha^{(k)}\right\}_{k=1}^{N}$
Output: resampled particle set
$j \leftarrow 1,\ c \leftarrow \alpha^{(1)}$
for $k = 1, \ldots, N$ **do**
    $u \sim \mathcal{U}\left(0, \frac{1}{N}\right)$
    $\beta = u + \frac{k-1}{N}$
    while $\beta > c$ **do**
        $j = j + 1,\ c = c + \alpha^{(j)}$
    add $\left(\boldsymbol{\mu}^{(j)}, \frac{1}{N}\right)$ to the new set

D. Laser Correlation model

A model for a laser scan z obtained from sensor pose x in an occupancy map m obtained by modeling the correlation between z and m

Transform the scan z to the world frame using x and find all points y in the grid that correspond to the scan Let the observation model be proportional to the similarity corr(y, m) between the transformed scan y and the grid m

The correlation is large if y and m agree:

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

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

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

E. Data Fusion

LIDAR: A horizontal (2D) LIDAR with 270-degree field of view and maximum range of 30m provides distances to obstacles in the environment. Each LIDAR scan contains 1081 measured ranges. We need to convert these ranges from the (x,y) coordinates in the lidar sensor frame to the world frame.

Kinect (RGBD camera): This sensor provides both RGB and disparity images which will be later used for texture mapping.

Odometry: We get the change in position $\Delta x, \Delta y$ & $\Delta\theta$ at LIDAR scan.

Joint Angles: We get the Head Angle and the Neck Angle along with the timestamp.

All the sensors have their own frequency for data collection. For making accurate estimations it is very important to synchronize the data. This can be done by matching the timestamps. Usually Joint angles are measured at much higher frequency as compared to LIDAR and odometry data. Hence for synchronisation, each LIDAR scan's timestamp is matched with that of Joint Angles' timestamp. Since exact same timestamp is not possible always nearest timestamp is chosen as the best timestamp. The same way was employed for Kinect (RGBD) data.

F. Occupancy Grid Map

Occupancy Grid Map is a grid of cells. It divides the environment into grids or cells of a particular resolution. Each cell is either occupied with $m_i = 1$ or free with $m_i = 0$

Log odds map: Since we are predicting or estimating an unknown environment we shall use probabilities instead of deterministic assignments to grid cells. Hence a pdf $p(m|z_{0:t}, x_{0:t})$ is maintained . Hence, given the measurements $z_{0:t}$ the distribution of m_i is given by,

$$m_i|z_{0:t} = \begin{cases} Occupied(1) & \text{with prob. } \gamma_{i,t} = p(m_i = 1|z_{0:t}, x_{0:t}) \\ Free(0) & \text{with prob. } 1 - \gamma_{i,t} \end{cases}$$

Now, to update $\gamma_{i,t}$,

$$\gamma_{i,t} = p(m_i = 1|z_{0:t}, x_{0:t})$$

$$= \frac{1}{\eta_t} p_h(z_t|m_i = 1, x_t) p(m_i = 1|z_{0:t-1}, x_{0:t-1})$$

$$= \frac{1}{\eta_t} p_h(z_t|m_i = 1, x_t) \gamma_{i,t-1}$$

$$(1 - \gamma_{i,t}) = p(m_i = 0|z_{0:t}, x_{0:t})$$

$$= \frac{1}{\eta_t} p_h(z_t|m_i = 0, x_t)(1 - \gamma_{i,t-1})$$

The odds of a binary random variable m_i updated over time via Bayes' rule and measurements $z_{0:t}$ is:

$$o(m_i|z_{0:t}, x_{0:t}) = \frac{p(m_i = 1|z_{0:t}, x_{0:t})}{p(m_i = 0|z_{0:t}, x_{0:t})} = \frac{\gamma_{i,t}}{(1 - \gamma_{i,t})}$$

$$= \frac{p_h(z_t|m_i = 1, x_t)}{p_h(z_t|m_i = 0, x_t)} \frac{\gamma_{i,t-1}}{(1 - \gamma_{i,t-1})}$$

$$= g_h(z_t|m_i, x_t) * o(m_i|z_{0:t-1}, x_{0:t-1})$$

Since the map cells are assumed to be independent, we can model it using how much we trust the occupancy measurement of cell i:

$$g_h(1|m_i, x_t) = \frac{p_h(z_t = 1|m_i = 1, x_t)}{p_h(z_t = 1|m_i = 0, x_t)} = \frac{80\%}{20\%} = 4$$

Similarly,

$$g_h(0|m_i, x_t) = \frac{1}{4}$$

Hence, our problem now is: For each observed cell i, decrease the log-odds if it was observed free or increase the log-odds if the cell was observed occupied, i.e.,

$$\lambda_{i,t+1} = \lambda_{i,t} + log(g_h(z_{t+1}|m_i, x_{t+1}))$$

Hence, our log odds map will be generated.

LIDAR scans: OpenCV's [4] cv2.line() function is used to determine the points of an n-dimensional raster that should be selected in order to form a close approximation to a straight line between two points. The best particle' position is used to get the best LIDAR scans in world frame, which after conversion to grid cells is passed to cv2.line(). We get the output with free space which is used to update the log odds map.

G. Texture Map

After we choose the best particle, we need to generate and update the texture map using this particles position and the RGB and disparity images given at that time duration. The disparity image is used to convert the pixel values to optical coordinates. But we can't use these coordinates directly since the axes in an image and a normal 3d space are defined in a different way. Hence, we need to convert these optical coordinated to a regular frame. Once in the 3D regular frame. Now we need to identify the coordinates which represent the floor. Hence, we use a threshold value of 0.1 for z coordinate, i.e. if $z < 0.1$ then that must be floor. After identifying these coordinates, we use their original pixel values to find the indices in the RGB image. The values at this position of the RGB image are then used to color the coordinates in the world frame.

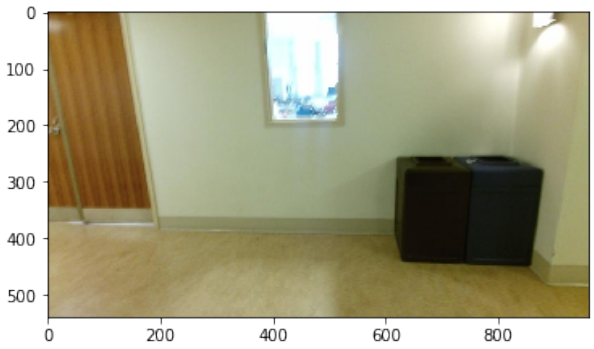


Fig. 1. Original RGB image

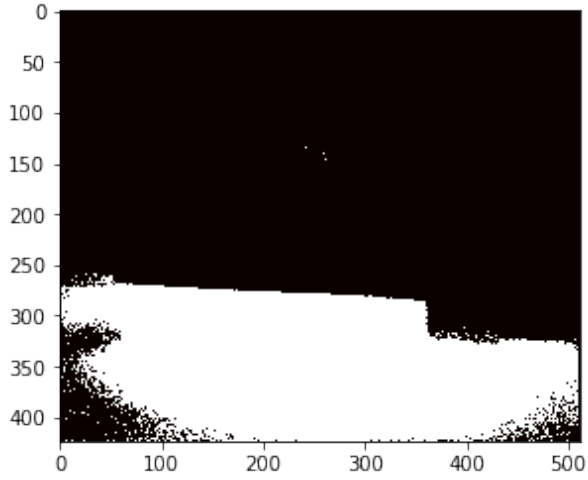


Fig. 2. After thresholding image, the floor is recognized as the white space

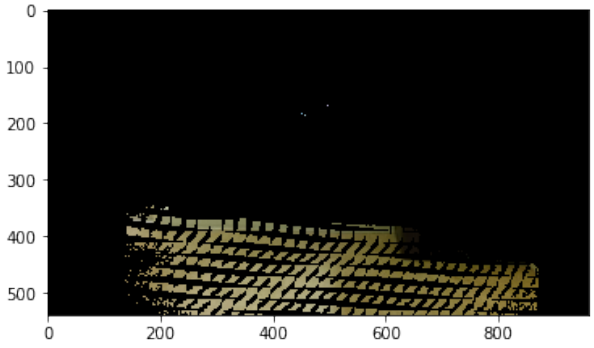


Fig. 3. RGB pixels corresponding to Floor (discontinuous due to low resolution of depth image as compared to RGB image)

IV. Results and Discussion

A. Dead Reckoning

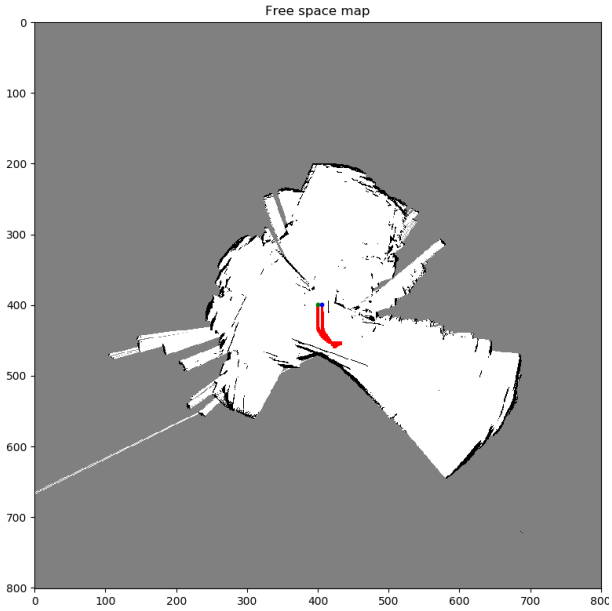


Fig. 4. Dead Reckoning for Dataset 0

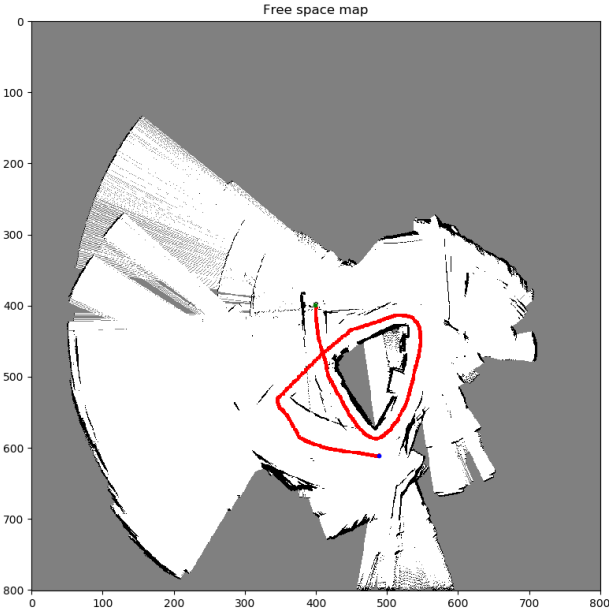


Fig. 5. Dead Reckoning for Dataset 1

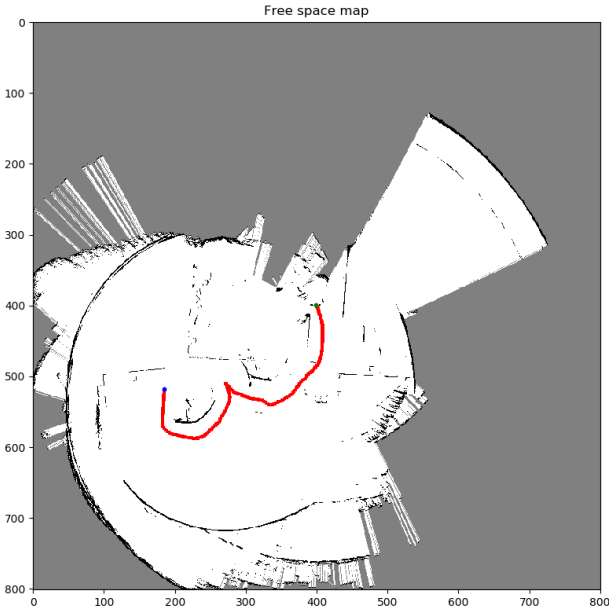


Fig. 6. Dead Reckoning for Dataset 2

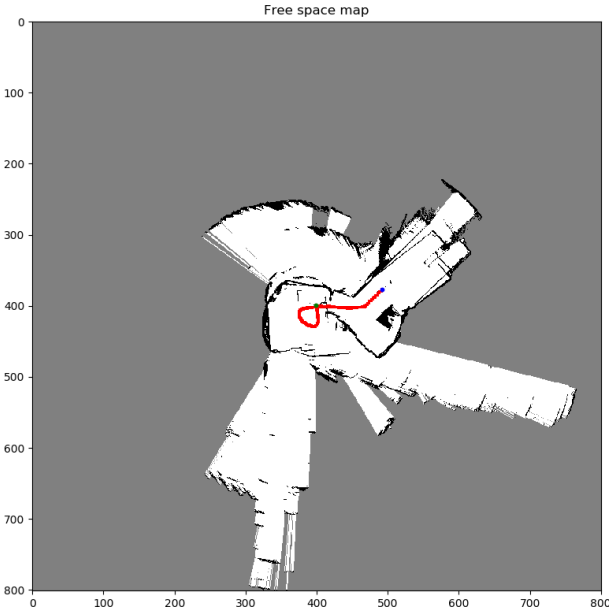


Fig. 7. Dead Reckoning for Dataset 3

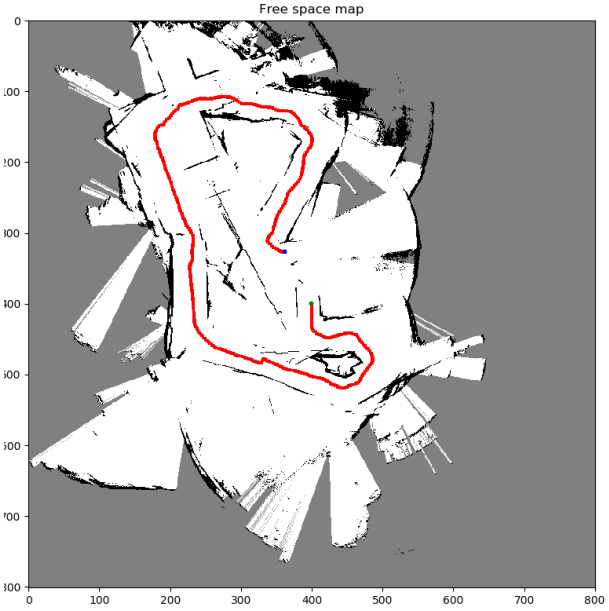
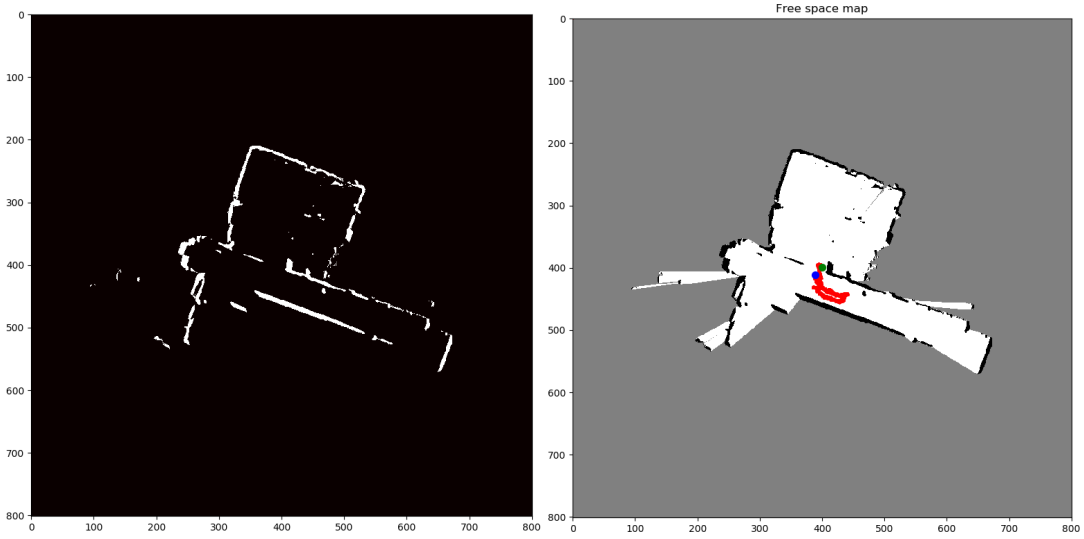


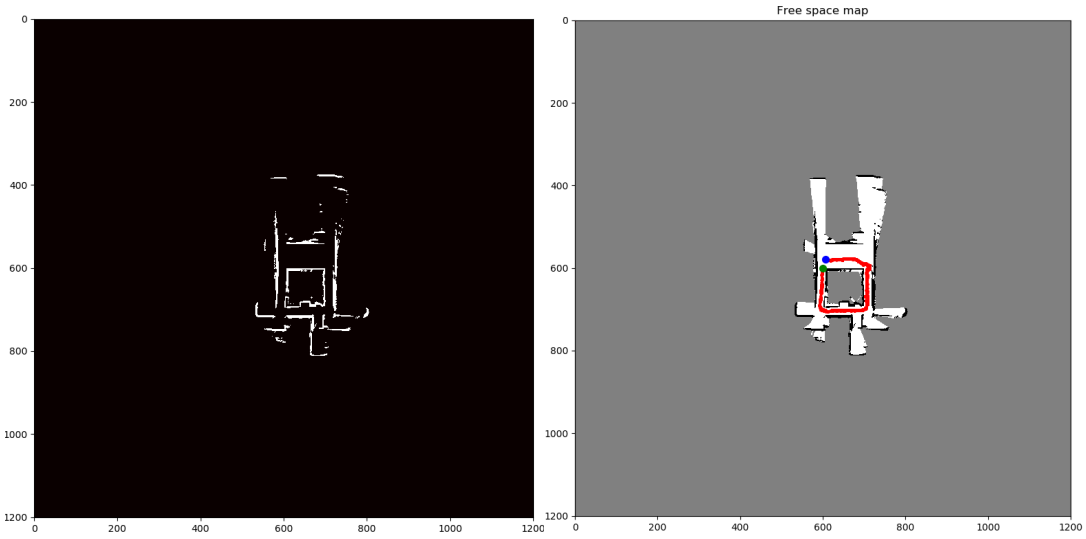
Fig. 8. Dead Reckoning for Dataset 4

B. Particle Filter



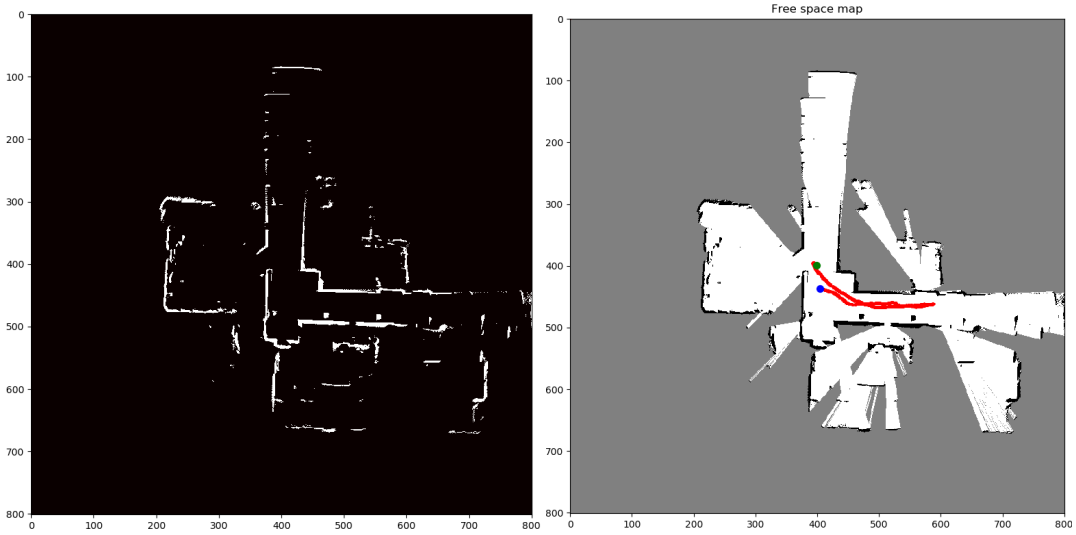
(a) Occupancy Map (b) Start point :– Green; End Point :– Blue

Fig. 9. Dataset 0: Particle Filter (200 particles)



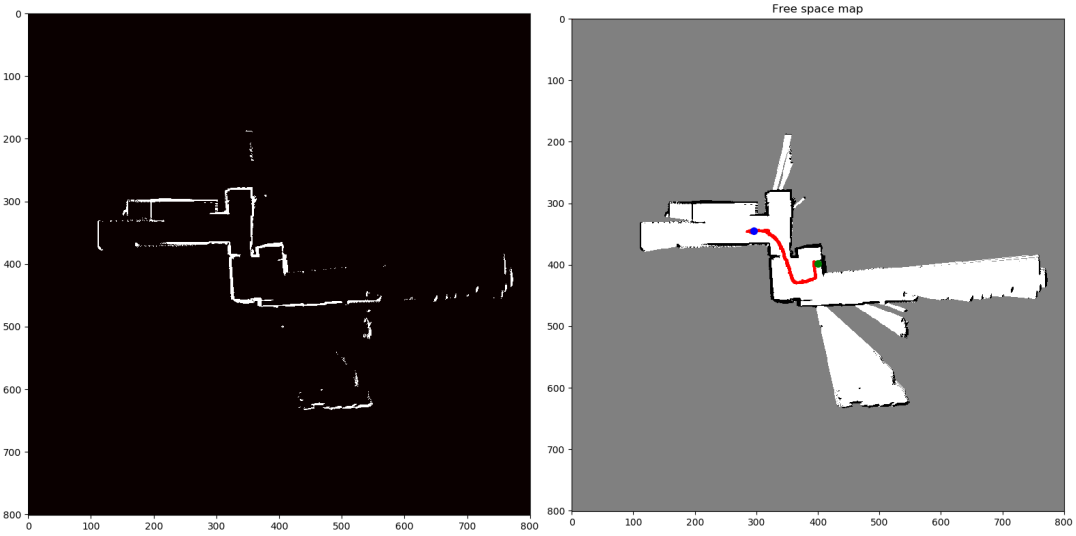
(a) Occupancy Map (b) Start point :– Green; End Point :– Blue

Fig. 10. Dataset 1: Particle Filter (200 particles)



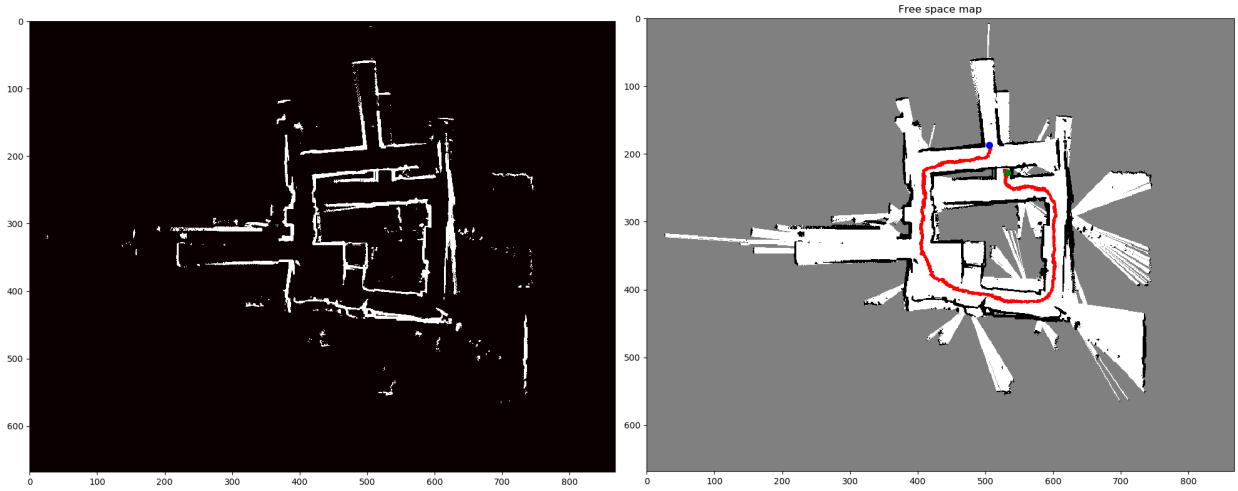
(a) Occupancy Map (b) Start point :– Green; End Point :– Blue

Fig. 11. Dataset 2: Particle Filter (200 particles)



(a) Occupancy Map (b) Start point :– Green; End Point :– Blue

Fig. 12. Dataset 3: Particle Filter (200 particles)

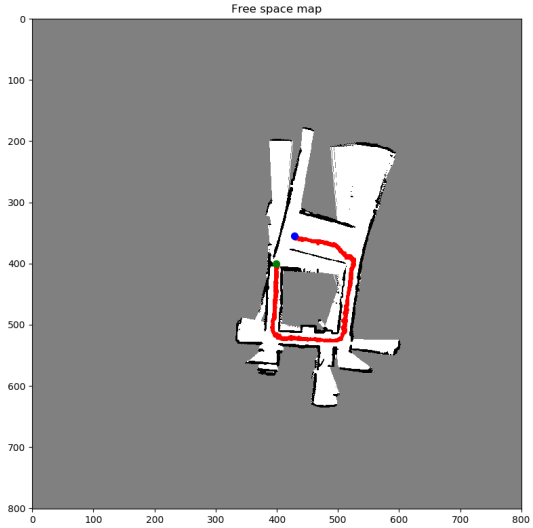


(a) Occupancy Map (b) Start point :– Green; End Point :– Blue

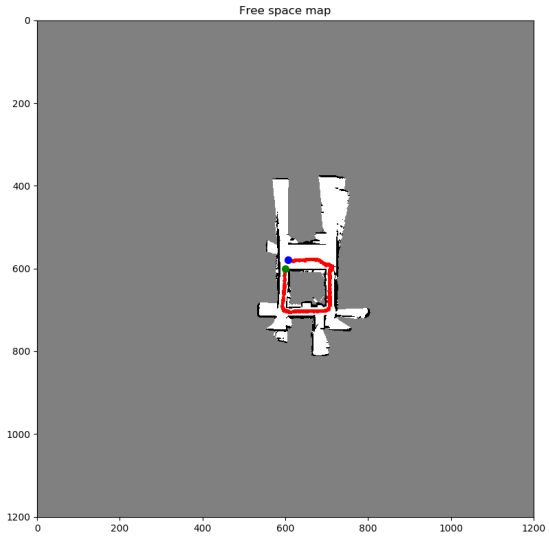
Fig. 13. Dataset 4: Particle Filter (200 particles)

C. Comparison with respect to number of particles

It can be seen clearly seen from Figure 15,16 and 17 that increasing the number of particles results in better SLAM. This is because $(x, y, \theta) \in \mathbb{R}^3$. Hence, generating only a few particles cannot account for variation in the real space. Had, it been discrete, less particles would have worked. Hence, to get accurate results, maximum number of particles considering time and space complexity must be used.

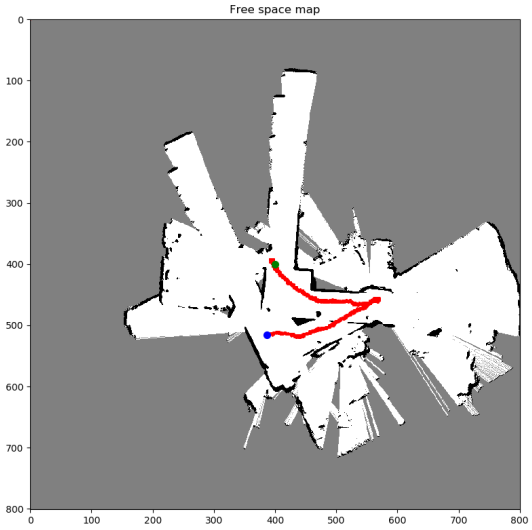


(a) 50 particles

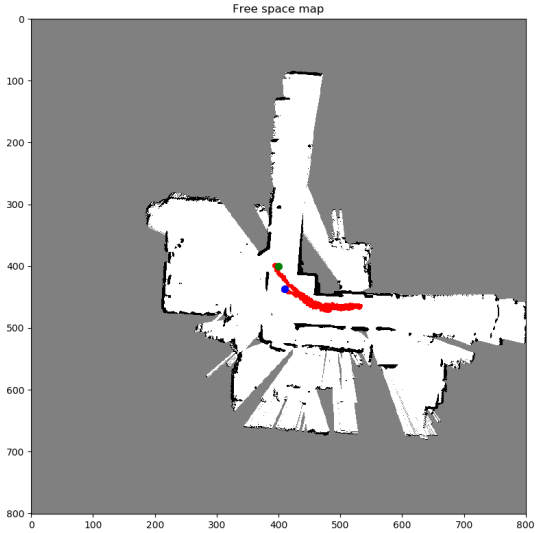


(b) 200 particles

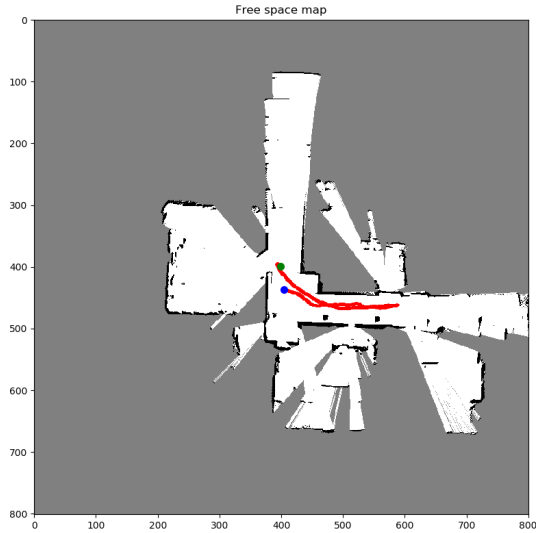
Fig. 14. Dataset 1: Particle Filter



(a) 10 particles

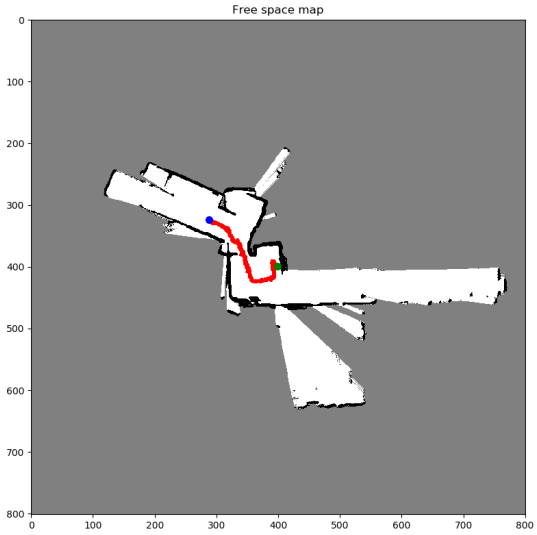


(b) 100 particles

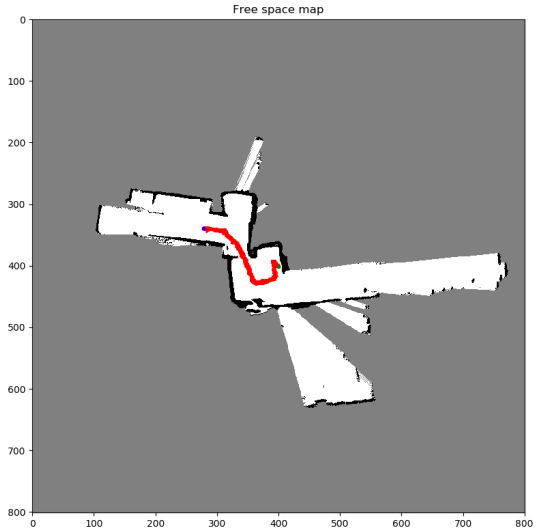


(c) 200 particles

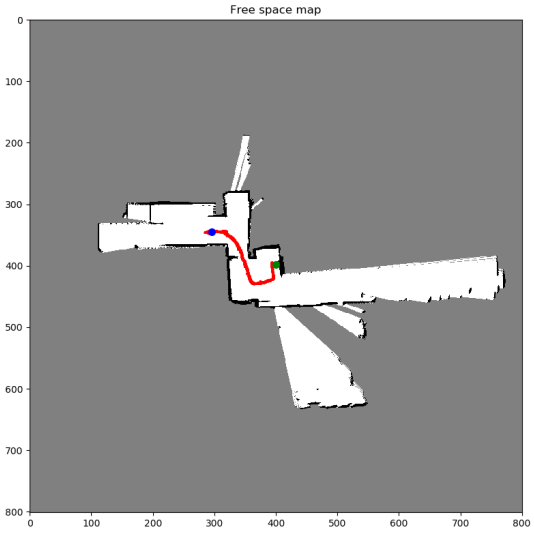
Fig. 15. Dataset 2: Particle Filter



(a) 20 particles

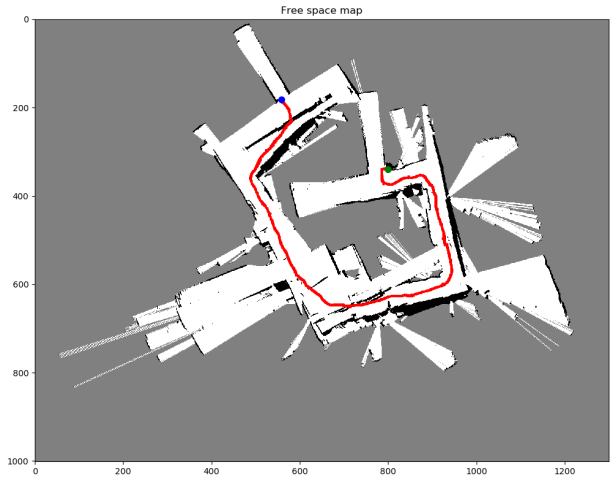


(b) 50 particles

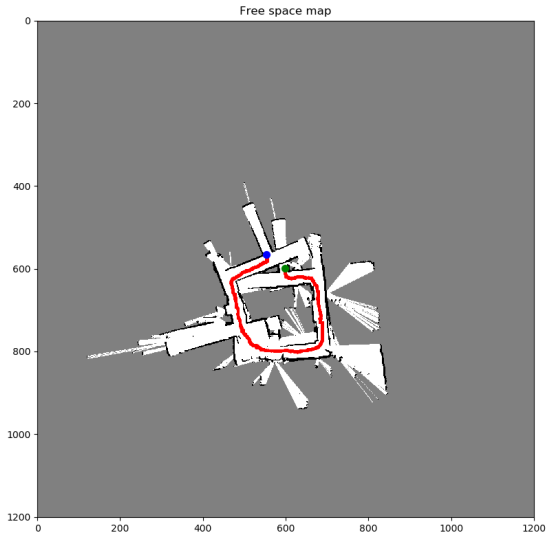


(c) 200 particles

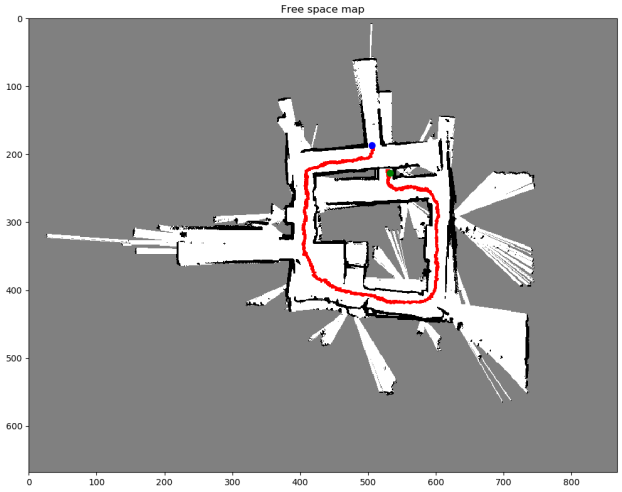
Fig. 16. Dataset 3: Particle Filter



(a) 50 particles



(b) 100 particles

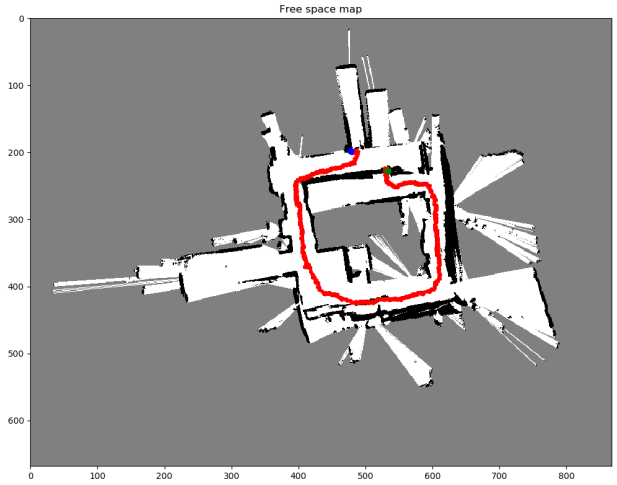


(c) 200 particles

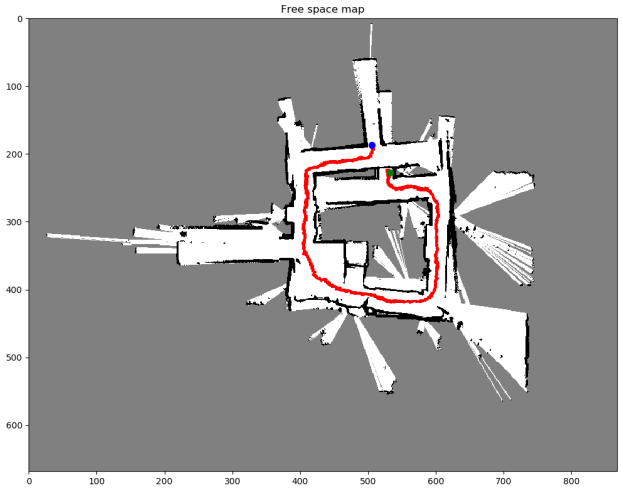
Fig. 17. Dataset 4: Particle Filter

D. Comparison with respect to variation in noise

(a) High noise in (x, y), Medium noise in θ



(b) Medium noise in (x, y), Low noise in θ



(c) Medium Noise in (x, y), Medium noise in θ

Fig. 18. Dataset 4: Particle Filter

There is a considerable variation with respect to noise as seen in Figure 18 It can be seen that noise needs to be injected sensibly. Too low or too high noise can result in incorrect maps. However, the filter is almost invariant to small changes in noise. Hence, we do not need to get the exact value of noise. Approximately will work just as fine.

E. Texture Mapping

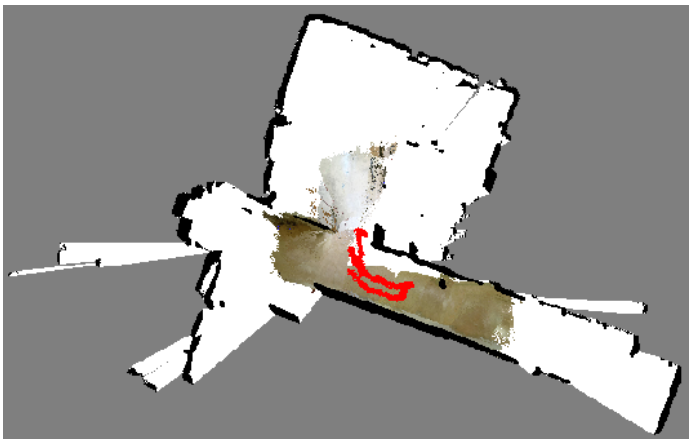


Fig. 19. Texture Mapping for Dataset 0

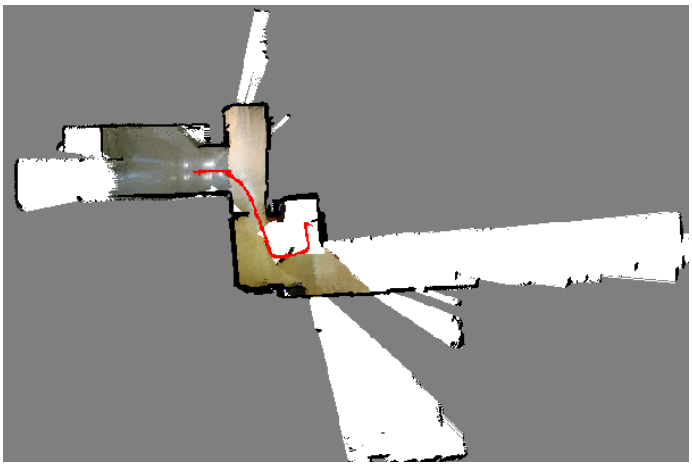


Fig. 20. Texture Mapping for Dataset 0

In figure 20, it can be seen that there are 2 distinct colors, one is yellowish (indoor flooring) and other is blackish (outdoors). There is a straight line which separates the two colors. This is exactly as in reality. This shows that our texture mapping is accurate. There are many white spaces, it is due to range difference between LIDAR and Kinect. LIDAR has a range of 30m while Kinect has about 4-5m. Hence, this was expected and can be clearly seen in the results.

V. Conclusion and Future Work

Particle Filter SLAM improves the results drastically. It helps in estimating what the actual pose of robot would have been by and hence create accurate maps of the environment. This thus solves the chicken egg problem of mapping and localization.

Currently for calculating map correlation yaw is not considered. Adding yaw variation is expected to improve the results to a great extent. It can help in solving problems faced for example in Dataset 4, where a loop closure can be seen clearly but is not obtained.

Acknowledgment

I thank Mr. Shiladitya Biswas, Mr. Sushruth Nagesh, Ms. Asfiya Baig, Ms. Savitha Srinivasan for collaborating with me on this project. We held many whiteboard discussion sessions.

References

[1] Mei Wu, Hongbin Ma, Mengyin Fu, and Chenguang Yang. Particle filter based simultaneous localization and mapping using landmarks with rplidar. In Honghai Liu, Naoyuki Kubota, Xiangyang Zhu, Rüdiger Dillmann, and Dalin Zhou, editors, *Intelligent Robotics and Applications*, pages 592–603, Cham, 2015. Springer International Publishing.

[2] Michael Montemerlo. *FastSLAM: A Factored Solution to the Simultaneous Localization and Mapping Problem with Unknown Data Association*. PhD thesis, Carnegie Mellon University, Pittsburgh, PA, July 2003.

[3] Nikolay Atanasov. Ece276a: Sensing estimation in robotics.

[4] G. Bradski. The OpenCV Library. *Dr. Dobb's Journal of Software Tools*, 2000.