Particle Filter SLAM and Texture Mapping

Surya Prakash Thoguluva Kumaran Babu
Department of Mechanical and Aerospace Engineering
University of California San Diego
stk222@ucsd.edu

***Abstract*—The objective of this paper is to implement particle filter simultaneous localization and occupancy-grid mapping (SLAM) using readings from LiDAR, Encoder and Gyroscope sensor measurements. In addition we obtained the texture map of the environment using the robot trajectory and the images collected from the camera mounted on the mobile robot.**

***Index Terms*—particle filter SLAM, LiDAR**

I. Introduction

The ability to accurately know a robot's positions in an unknown environment using the sensors is of absolute importance to a mobile robot. In real world, robots face this exact challenge of knowing it's own location in an unknown environment. Several methods and algorithms have been developed to address this issue over the past few decades. SLAM is a general term that is given to the solutions that addresses this problem. This is an active area of research as there is no one standard algorithm which will work for every mobile robot. For example, a moon rover might implement a sophisticated SLAM algorithm using highly accurate sensors, whereas implementing those algorithms for a home cleaning mobile robot may not be feasible. Hence there is a need to explore several variants of SLAM like particle filter SLAM, Extended Kalman Filter (EKF).

In this paper, we have discussed an implementation of particle filter SLAM for a mobile robot that is equipped with encoder, LiDAR scanner, IMU sensors and a camera. Differential drive robot motion and LiDAR scans are used to localise the robot and build a 2D occupancy grid map of the environment. In the second part of the project, a texture map of the environment is constructed based on the images collected by the camera attached to the head of the mobile robot.

II. Problem Formulation For Particle Filter SLAM

A. Given Input

We have 2 sets of data obtained from a mobile robot. Each sets contain the following data:

- Encoders: A $4 \times N$ matrix where $N \in \mathbb{N}$ represents the number of time steps for which the data is provided. The rows of this matrix corresponds to encoder counts of front-right, front-left, rear-right, and rear-left wheels. In addition Unix timestamps corresponding to every column is also given.
- Inertial Measurement Unit: Linear and angular measurements from a noisy IMU sensor sampled at 40 Hz is provided along with the Unix timestamps at which these were recorded.
- Distances measured by a 270 degrees field of view LiDAR sensor is given in the form a $1081 \times N$ matrix where $N \in \mathbb{N}$ is the number of time steps and 1081 indicates that there are 1081 distances measured per scan at 1081 angles that at uniformly spaced between -135 degrees to +135 degrees. The sensor used is called Hokuyo UTM-30LX. In addition Unix timestamps are also provided.
- RGB Images and disparity images from a RGBD camera is provided.

B. Notations

The following notations are used in the problem formulation.

- Motion model function f and its probability density function p_f that describes the states x_{t+1} resulting from applying input u_t at state x_t is given as follows:

$$x_{t+1} = f(x_t, u_t, w_t) \sim p_f(.|x_t, u_t) \quad w_t = motionnoise$$

- Observation Model function h and its probability density function p_h that describes the observation z_t depending on x_t is given as

$$z_t = h(x_t, v_t) \sim p_h(.|x_t) \quad v_t = observationnoise$$

- The update probability density function (update pdf) is defined as the conditional probability on state x_t given the observation $z_{0:t}$ and the control input $u_{o:t}$

$$p_{t|t}(x_t) := p(x_t|z_{0:t}, u_{0:t-1})$$

- The predicted probability density function (predicted pdf) is defined as conditional probability on the state x_t given the motion model and control input at time t.

$$p_{t+1|t}(x_{t+1}) := p(x_{t+1}|z_{0:t}, u_{0:t})$$

C. Markov Assumptions

The particle filter implemented in this projects are based on the following Markov assumptions listed as below:

- Control inputs $u_{0:t}$ and observation $z_{0:t}$ are known
- The robot states $x_{0:t}$ and map states $m_{0:t}$ are unknown
- The state x_{t+1} only depends on the previous input u_t and the state x_t. This is to say that x_{t+1} given u_t, x_t is independent of the history $x_{0:t-1}, z_{0:t-1}, u_{0:t-1}$
- The observation z_t only depends on the state x_t

D. Particle Filter

The particle is a special case of Bayes filter in which the update pdf and predicted pdf are discrete distributions with N Possible values called particles. A particle is defined as a hypothesis that the value of the state x is $\mu[k]$ with probability $\alpha[k]$. The probability mass function of the state x for N such hypothesis can be viewed as probability density function:

$$p(x) = \sum_{k=1}^{N} \alpha[k]\delta(x - \mu[k])$$

where δ is the Dirac Delta function.
This can then be used to find pdf $p_{t|t}$ and $p_{t+1|t}$ that can be plugged into Bayes filter to obtain particle filter predict step and update step. The goal of this project is to find an real time implementation of the below steps to perform SLAM.

E. Particle Filter Prediction Step

The predict step of particle filter uses the motion model and the control input to obtain a new state $\mu_{t+1|t}[k] \sim p_f(.|\mu_{t|t}[k], u_t)$ The predict pdf of the new state is then given as

$$p_{t+1|t}(x_{t+1}) = \sum_{k=1}^{N} \alpha_{t|t}[k]\delta(x_{t+1} - \mu_{t+1|t}[k])$$

The prediction step changes only the particle positions but not their weights.

F. Particle Filter Update Step

The $p_{t+1|t}(x_{t+1})$ from the predict step is used in the Bayes filter update step to get the update step of the particle filter $p_{t+1|t+1}(x_{t+1})$ is given by

$$\sum_{k=1}^{N} \left[\frac{\alpha_{t|t}[k] p_h(z_{t+1} \mid \mu_{t+1|t}[k])}{\sum_{j=1}^{N} \alpha_{t|t}[j] p_h(z_{t+1} \mid \mu_{t+1|t}[j])} \right] \delta(x - \mu_{t+1|t}[k])$$

The update step changes only the particle weights but not their positions.

G. Particle Re-sampling

Recursively implementing the particle filter predict and update step will make the most updated particle weights to become zero. This is called particle depletion. In order avoid this re-sampling is done, where the less likelihood particles are move the positions which has more likelihood and the particle filter update and predict steps are implemented again.

H. Occupancy Grid Mapping

Occupancy grid map is a way to represent an environment into a regular grid with n cells. Mathematically occupancy grid is a vector $m \in \mathbb{R}^n$ whose i entry is free if $m_i = -1$ and occupied if $m_i = 1$. The mapping problem is to find m based on the robot trajectory $x_{0:t}$ and observations $z_{0:t}$. Since the measurements $x_{0:t}$, $z_{0:t}$ are uncertain, m can also be represented as probability mass function $p(m|z_{0:t}, x_{0:t})$ that evolves over time. Log-odds occupancy grid mapping is estimating this probability mass function of m_i (assuming Bernoulli random variable) conditioned on $z_{0:t}$ and $x_{0:t}$. This is equivalent to accumulating the log-odds ration $\Delta\lambda_{i,t}$ defined as:

$$\Delta\lambda_{i,t} = \lambda_{i,t-1} + (\Delta\lambda_{i,t} - \lambda_{i,0})$$

The log-odds ratio specifies the measurement trust and is defined as:

$$\Delta\lambda_{i,t} = \log \frac{p(m_i = 1 \mid z_t, x_t)}{p(m_i = -1 \mid z_t, x_t)}$$

where z_t indicates whether m_i is occupied or not.

III. Problem Formulation for Texture Mapping

The objective is to transform the RGB image obtained from the camera and map it to the occupancy grid.

IV. Technical Approach For Particle Filter SLAM Implementation

Typically a SLAM problem can be solved by initializing the environment with the current map, assume robot to be at origin followed by alternatively applying the robot prediction step using motion model and map update step using the observation model. To implement particle filter SLAM, the following procedure was followed:

- Data cleaning and time synchronization of sensor values
- Implement prediction step using differential drive motion model
- Implementing particle filter update step using scan grid correlation model
- Particle re-sampling step to remove less likely hypothesises and move them to higher probable locations.
- Updating map based on the observation from the most likely particle

A. Data Cleaning and Time Synchronization

1) Time Sync: Time synchronization is done first as the least count of all the sensors are different and the Unix timestamps are different for each of them. As encoder has the lesser number of readers and since it started late, the timestamp of encoder is used a reference. A time sync map was created based on the closest in the past time with respect to encoder time. Each row in this map corresponds to the rows of those corresponding sensors that have a time sync.

2) LiDAR Data Cleaning: All the range values of LiDAR that are below 0.1 meters and more than 30 meters are set to zero. This is done to reduce measurement errors due to limited field of view and also it is hard to distinguish features that are near as the angle variation is very small in close ranges. Farther readings are discorded as well because farther distances introduce more noise into the data.

3) Remove IMU noise: Noise from IMU sensor needs to be removed. To this, a moving average filter was implemented. Since IMU has more number of measurements compared to encoder, an average of IMU measurements that are present between the timestamps that were synchronised is computed and used for further processing. In this way, the noise due to IMU measurements can be reduced.

4) Velocity Estimate from Encoder and IMU Data: Encoders count the rotation of the wheel. Data is sampled at 40 Hz, which means the counter is reset every 0.025 sec. Every 0.025 seconds, the number of ticks the encoder detects is returned as the data point. Based on the number of encoder counts, timestamps, 0.254 meter wheel diameter, velocities of all the 4 wheels can be calculated and the average of these velocities is interpreted as the robot's linear velocity. Angular velocity estimate is directly fetched from the yaw rate of the IMU data.

5) LiDAR Frame Cartesian Coordinates of the Obstacles: The liDAR range values from LiDAR frame are transformed to body frame coordinates as mentioned below:

$$\begin{bmatrix} x_b \\ y_b \\ z_b \end{bmatrix} = \begin{bmatrix} r\cos\alpha \\ r\sin\alpha \\ 0 \end{bmatrix} + \begin{bmatrix} 0.133 \\ 0 \\ 0.494 \end{bmatrix}$$

where r is the range measurement from LiDAR sensor and $\alpha \in (-135^0, 135^0)$ is the angle at which the range measurement was captured. The above transform is assuming that the lidar frame is orientated exactly with respect to the robot frame and there is an offset of 0.133 meters and 0.494 meters in the x and z axis respectively.

B. Prediction Step with Differential Drive Motion Model

In this project we have used differential drive motion model to predict the future state of the robot trajectory. The following equation is used to obtain to describe the motion model:

$$f(x,u) = \begin{bmatrix} X_{t+1} \\ Y_{t+1} \\ \theta_{t+1} \end{bmatrix} = \begin{bmatrix} X_t \\ Y_t \\ \theta_t \end{bmatrix} + \tau \begin{bmatrix} v\cos\theta \\ v\sin\theta \\ \omega \end{bmatrix}$$

where $X_t, Y_t, \theta_t, v, \tau, \omega$ are the x coordinate, y coordinate, angle of rotation about the z-axis, velocity of the robot, time step and angular velocity of the robot respectively. Robot is assumed to start at 0,0 position and with 0 degrees orientation. v, ω are obtained from encoder, IMU data. Dead reckon of the trajectory of the robot is shown in the figure 1. Noise can be added to the control input. A visualization of varying noise of zero mean and 0.05 standard deviation in linear velocity and 0 mean and 0.0001 standard deviation in angular velocity is being added to the input is shown in figure 5 for 10 particles. The prediction step of each particle $\mu_{t|t}[k] \in \mathbb{R}^3$ represents the 2D position (x,y) and orientation θ. The prediction step is then defined as:

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

where ϵ_t is a 2D Gaussian motion noise with zero mean and 0.05 standard deviation for v_t and 0.0001 standard deviation for ω_t

C. Initialize Occupancy Grid Map

The LiDAR data from LiDAR frame is converted to world frame assuming that the robot is at origin. A map of size 1401 x 1401 pixels is created with all values initialised to 0.5. The resolution of map is set to 0.05 meters per pixel in both the x and y direction. With this information, the world frame coordinates are converted to pixel locations in the map and are assigned a value of 1 indicting that it is occupied. All the pixels from origin to the end point of the pixels are found and set to 0. Similarly a log odds map is initialised with 0.5 value meaning that all the cells are equally likely. The pixels in the log odds map corresponding to LiDAR endpoints are added log(4) (assuming 80% trust) and the free space between origin to the lidar end point is subtracted a value of -log(4) to indicate that these pixels are less likely to be occupied. The initialized map is shown in figure 3.

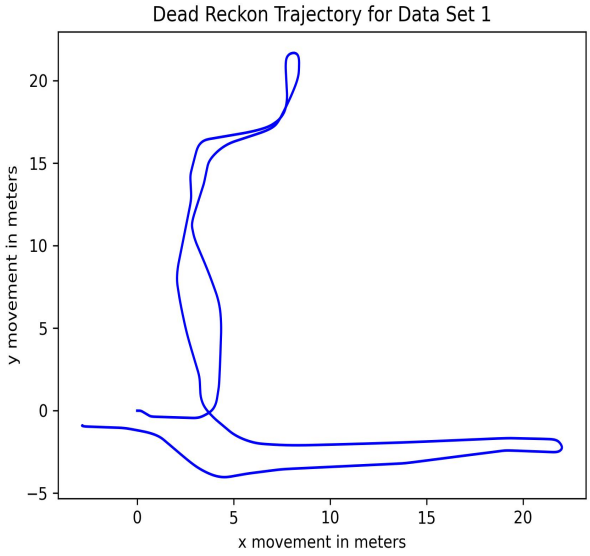

Fig. 1. Dead Reckon of the Robot Trajectory.

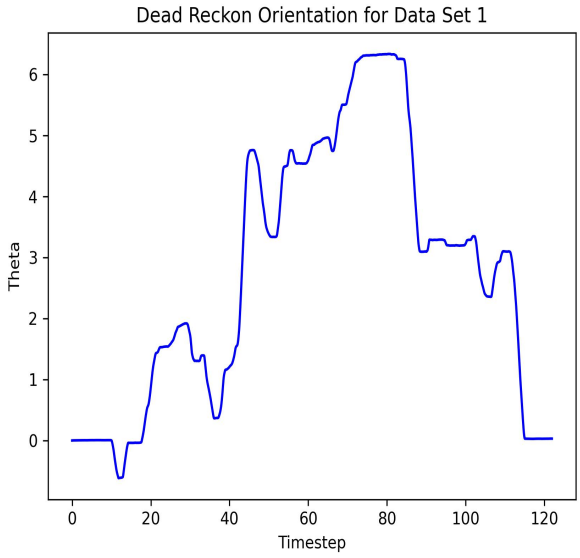

Fig. 2. Dead Reckon Orientation of Data set 1.

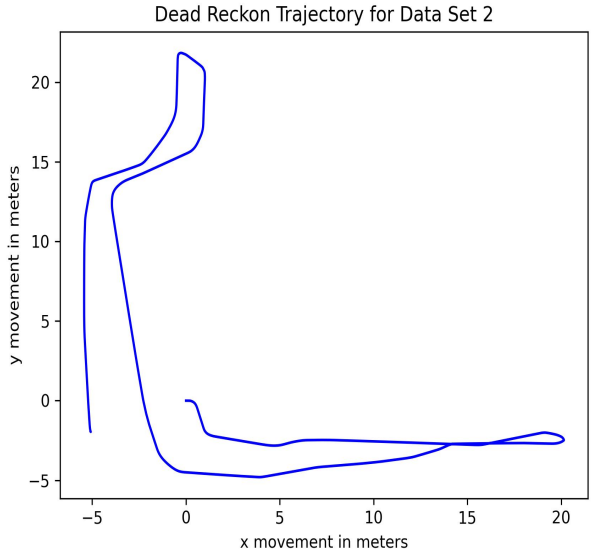

Fig. 3. Dead Reckon of the Robot Trajectory.

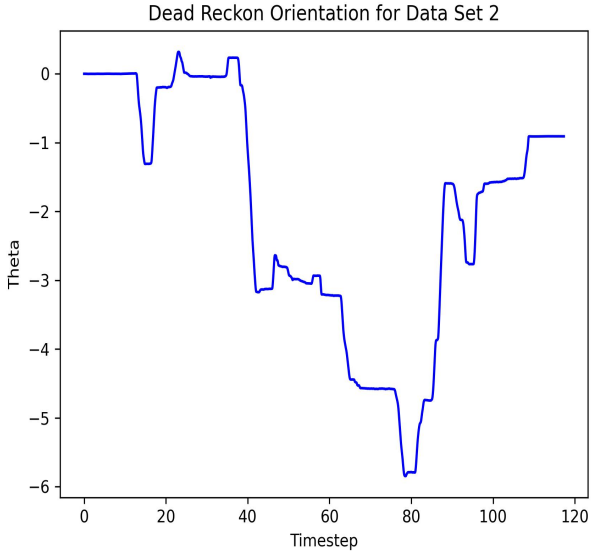

Fig. 4. Dead Reckon Orientation of Data set 2.

D. Update Step with LiDAR Measurements

As discussed in the problem formulation the particle hypothesis remains fixed during the update step whereas the weights of the particle are updated based on the observation model. In this project we are assuming that the correlation obtained from LiDAR scans with respect to the map created from $0 : t-1$ is proportional to the observation model pdf. Therefore the weigths are updated as :

$$\alpha_{t+1|t+1}[k] = corr(y_{t+1}[k], m)\alpha_{t+1|t}[k]$$

where the correlation is found as mentioned below:

- First transform the lidar scans z_t to the world frame coordinates using $\mu_{t+1|t}[k]$
- Convert the world frame coordinates to pixel coordinates
- Check these pixel locations in the map created from time 0 to t-1. A value of 1 is added to the correlation of the map is occupied at that particular location. Add -1 to the correlation if the location 0 to penalize the incorrect estimate.
- The final correlation after going through all the scans will be the correlation of this LiDAR scan withe map.
- This entire process is repeated for a neighbourhood of 9x9 pixels around the estimated robot coordinates in the world frame.
- A 2D 9x9 matrix of all the correlations for 1 scan. The elements in the matrix are normalized to range 0 to 2 using a linear function. Since the maximum value of this sum can be -1081 (if all the cells are free) and 0 (if all the cells are unexplored) and 1081 (if all the cells are occupied), a scaling factor is obtained to bring all these values in range of 0 to 1.
- The element with maximum correlation is the correlation of this position's lidar scan and the pixel from which this was calculated will be the most likely particle location.
- The same procedure is repeated for N particles.

The correlation values obtained this way indicates how much the LiDAR scans agree with the map. This measure is then used to the particle weights. In addition to this step, the particle hypothesis was moved to the position in the neighbourhood with maximum likelihood.

E. Particle Re-sampling

To counter the problem of particle depletion an measure of effective number of particles is tracking. That is defined as

$$N_{eff} = \frac{1}{\sum \alpha_i^2} <= N_{threshold}$$

where α_i are the weights of each particles. If N_{eff} falls below a threshold, re-sampling is done. A random index is generated from the given set of index which has the probability of α_i for the i index. Then the hypothesis $\mu[k]$ is added to the new set of hypotheses. This is repeated for N times, where N is the number of particles.

F. Occupancy Grid Map Update

The Lidar scans of the particle with maximum weight is transformed to the world frame coordinates and then to the pixel coordinates of the occupancy grid. A value of log(4) is added to the existing value in the occupancy grid. Bresenham2D ray tracing algorithm is used to identify all the free pixels in between and -log(4) is added to the existing locations. A lower and upper bound on the values of scan grid to avoid overconfident estimations. A value of 50 and -50 was used in this project implementation.

G. Implementing Particle Filter SLAM

The steps mentioned above was programmed in python. 100 particles with, motions model noise of 0.01 standard deviation in linear velocity and 0.001 standard deviation in angular velocity was used for dataset 1. For dataset 2, 0.005 standard deviation noise in linear velocity and 0.0001 standard deviation in angular velocity with 50 particles that are sampled every 20th timestep was implemented. A bigger map dimension was used for dataset 2. The results are shown in the results section.

V. Technical Approach For Texture Mapping

RGB and disparity images are given as input. The first step is to time synchronise the disparity, RGB images with the encoder timestamps. I have used dead reckon trajectory to implement texture mapping as I was not sure of my results from my SLAM output and it was also computationally faster to debug and test my results. From the disparity images, using the inverse transform of stereo camera model, the x, y,z coordinates were calculated. The stereo camera model is defined as below:

$$\begin{bmatrix} u_l \\ v_l \\ d \end{bmatrix} = \begin{bmatrix} fs_u & 0 & c_u & 0 \\ 0 & fs_v & c_v & 0 \\ 0 & 0 & 0 & fs_u b \end{bmatrix} \frac{1}{z} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

where $d = u_l - u_r = \frac{1}{z} f s_u b$, d is disparity measurement, u_l, v_l are pixel coordinates and f, s_u, s_v, c_u, c_v are calibration parameters given to us. All the coordinates for which z value was between 0 and 0.04 was chosen as they represent the base of the robot. These x,y coordinates were then transformed from optical to regular frame and then to the world frame by obtaining the robot's state from motion model. The world frame coordinates are then converted to pixel coordinates with respect to the occupancy grid. This essentially gives the map from the disparity image location to the occupancy grid. The RGB value from the RGB image that is corresponding to the u, v is assigned to the corresponding occupancy grid map which now has a third dimension to store the RGB values. This array is then plotted as texture map.

VI. Discussions

A. Optimization The Code

The code tried to minimise the runtime of the program by avoiding for loops as much as possible. A few observations made are discussed below.

- Vectorising the mapCorrelation function showed a significant boost in run time when the number of particles are of 100s of magnitude
- Bresenham2D is the most time consuming loop which could have been vectorised to get a much more optimal runtime.

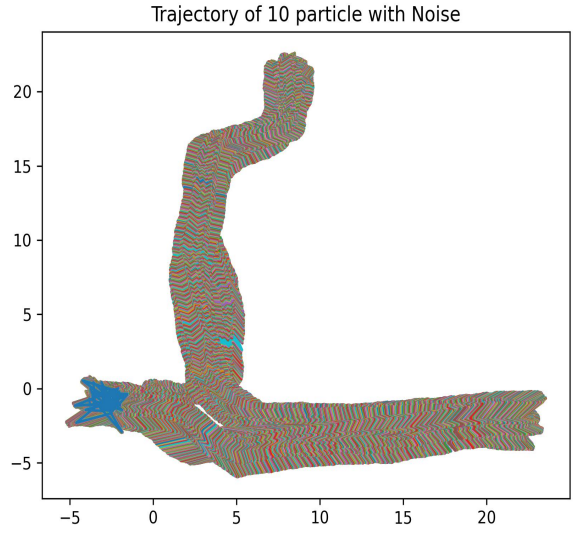

Fig. 5. Dead Reckon With Noise.

B. Effect of Noise

Noise that is added to the motion model will significantly determine the output. Listed below are a few observations. A plot of noise effect on dead reckon is shown in figure 5.

- There is a significant difference noticed when the noise was added to the control input vs the noise added to the robot's state. The noise in input had a trajectory close to the robot's dead reckon while the noise in the robot's state did not have any kind off over all pattern. One reason could be due to the noise added to the input somehow gets attenuated when it translates to noise in the robot's state.
- Noise that is added to the angular velocity needs to have a smaller standard deviation compared to noise that is added to the linear velocity.

C. Effect of Correlation Function

The type of correlation function being used needs to be strictly greater than 0. In this project a linear approximation was used to bring the correlation from -1081 to 1081 to an interval of 0 to 2. Although it was intuitive to use a softmax or a exponential function for the correlation, it did not yield significant change in the output from SLAM.

D. Effect of Changing Bounds for Log-Odds

The output map had a significant difference depending on the choice of log-odds. The observations are summarised below:

- The maximum value that log odds could take in the upper limit was much lower than the lower bound absolute value. So equal values of threshold needs to be avoided. In general a larger upper bound and a small lower bound seems to be reasonable as in a large map, the probability

of free space is in general higher than the probability of occupied space.
- Not setting a bound led to abrupt results. This is especially due to the fact that the robot visits some places twice but the duration of stay is not the same. So the output from the scans for which the robot stayed more could have caused a bias in the occupancy grid.

E. Sampling Frequency

Various sampling frequencies were experimented to check for optimal solution.

- A threshold of N/10 condition was never met when the number of particles was 5. However as the number of particles increased the resampling frequency also increased. This could be due to the fact that now there could be more probability that the robot is in wrong position as the number of particles is more.
- Re-sampling during every timestamp also did not cause any useful results for a smaller number of particles. However for N¿40, re-sampling every step provided a better output compared to using N/10 threshold
- A threshold of N/5 was a good hyper parameter to get valid outputs.

F. Particle Filter Plots Discussion

The output seems to vary significantly from the dead reckon plot. The observations are mentioned below:

- Varying the number of particles from 100 to 500 increased the computational time significantly without providing better results
- There is a drift in the map. Attempted adding more noise to the motion model, but however the results were not better than the existing output
- The algorithm is not able to handle a 180 degree rotation of the robot accurately. I tried to use exact integration of the differential drive motion model. However the result was exactly the same.
- As it can be seen that although the map dimensions were same for dead reckon and the particle filter SLAM for dataset 1, the SLAM output is not able to be fit in the map. This maybe a consequence of higher error in the linear velocity of the motion model.
- The output from the second SLAM deviated even further from the dead reckon in spite of higher re sampling with lesser number of particles. This indicates that re-sampling more often can negatively impact the final results.

G. Texture Map Discussion

It can be seen the texture map developed has patches. This could be due to the fact that the number of images are lesser than the number of data we have from encoders.

VII. Results of Particle Filter SLAM

The dead reckon plot with LiDAR mapping for dataset 1 and 2 is shown in figure 6 and 7. The initial occupancy grid of lidar scan from dataset 1 and 2 is shown figure 8 and 9 respectively. The particle filter output of 50 particles with noise of 0.05 standard deviation in linear velocity input and 0.0001 standard deviation in angular velocity for both dataset is obtained. The figures below show the occupancy map evolution at 1000, 2000, 4000 th timestep along with the best hypothesis estimated at that time.

Fig. 6. Dead Reckon Mapping for Data Set 1.

Fig. 7. Dead Reckon Mapping for Data Set 2.

VIII. Results of Texture Mapping

The texture map for dataset 1 and 2 are shown in the last 2 figures.

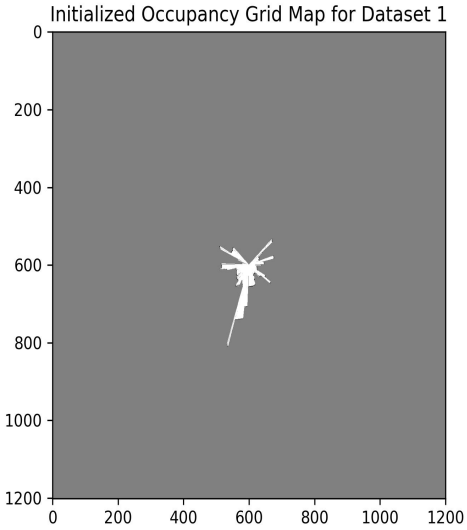

Fig. 8. Occupancy Grid Initialization for Dataset 1.

Fig. 9. Occupancy Grid Initialization for Dataset 2.

IX. Acknowledgement

I would like to acknowledge that I have discussed concepts related to this project on a high level with Ritika Kishore Kumar.

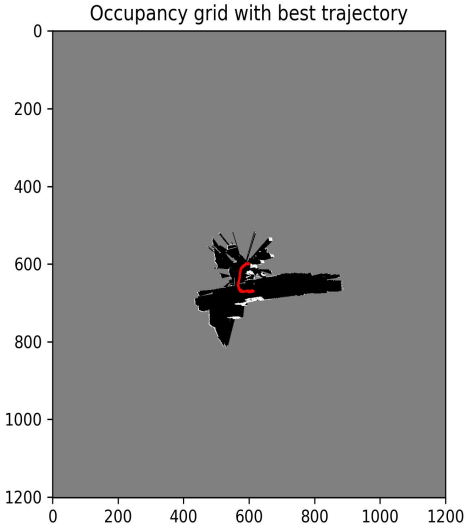

Fig. 10. SLAM For Dataset 1 at 1000th Timestep.

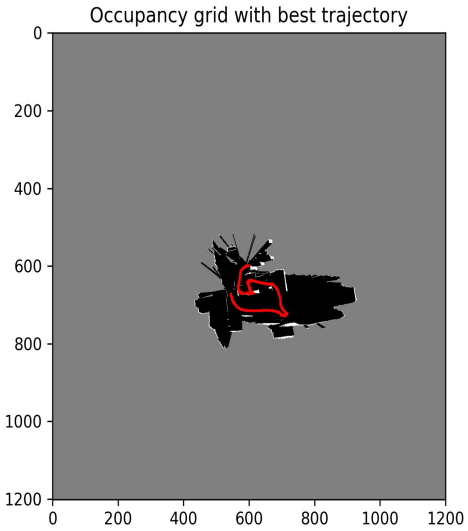

Fig. 11. SLAM For Dataset 1 at 2000th Timestep.

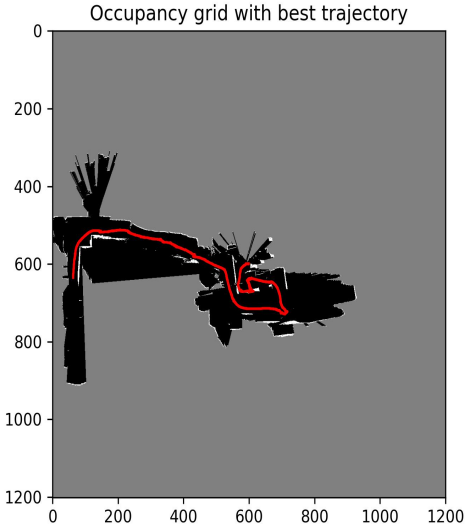

Fig. 12. SLAM For Dataset 1 at 3000th Timestep.

Fig. 13. SLAM For Dataset 1 at the Last Timestep.

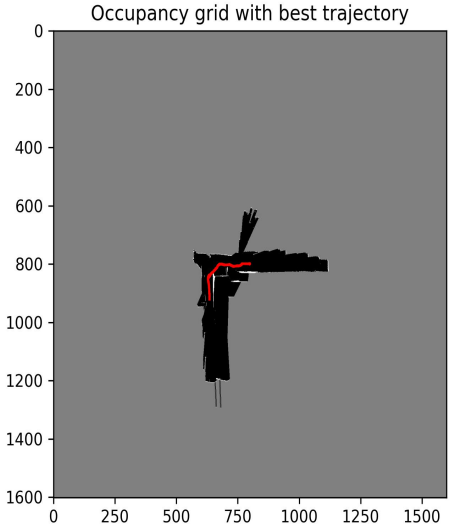

Fig. 14. SLAM For Dataset 2 at the 1000th Timestep.

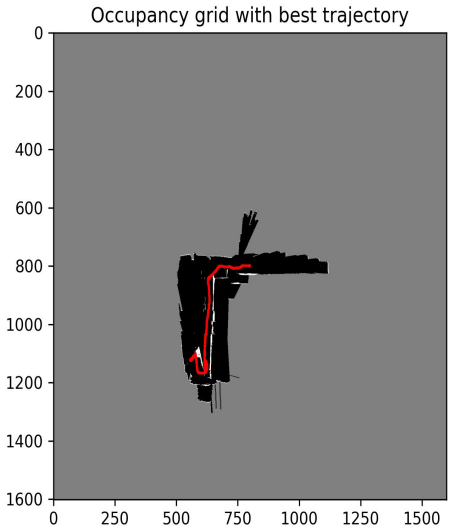

Fig. 15. SLAM For Dataset 2 at the 2000th Timestep.

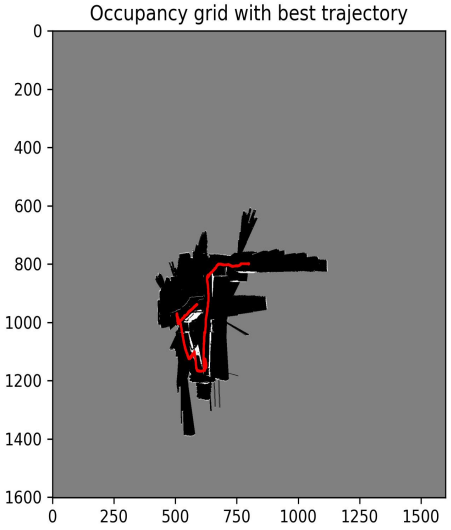

Fig. 16. SLAM For Dataset 2 at the 3000th Timestep.

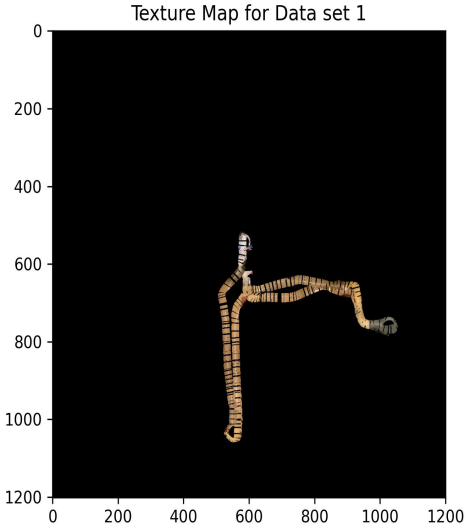

Fig. 18. Texture Map for Dataset 1.

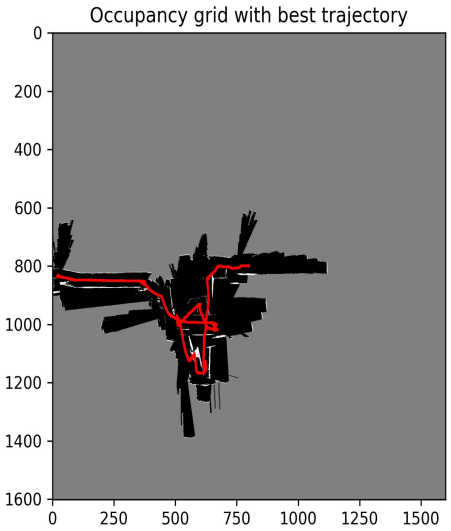

Fig. 17. SLAM For Dataset 2 at the Last Timestep.

Fig. 19. Texture Map for Dataset 2.