Particle Filter SLAM

Renu Krishna Gutta, PID: A59018210, ECE 276A, Project-2

1. INTRODUCTION

The goal of this project is to implement simultaneous localization and mapping (SLAM) using encoder and IMU odometry and LIDAR scans from a differential-drive robot. Later, use the RGBD measurements taken by the robot to assign colours/texture to the 2D-map of the floor.
SLAM using robots is a powerful way to study spaces where humans cannot physically enter. For example, we can send nanobots into human body to map the narrow blood vessel system and identify any issues. Miners can map the underground channels without risking their lives.
In this project, I implemented a particle filter strategy where robot pose was considered as a random variable with multiple values depicted by the particles and their weights as the probability measure. Noise was introduced to odometry measurements to make up for the sensors' intrinsic noise. With the help of LIDAR scans, I identified the free space and obstacles, and performed a similarity measure to choose the most probable particle (robot) pose at each instant. Finally, I estimated the trajectory and the 2D map of the robot's surroundings. With the colour and depth information from the RGBD data, I added texture to the map.

2. PROBLEM FORMULATION

Following data is available:

a. Encoder – Provides linear velocity information of four wheels.
b. Inertial Measurement Unit (IMU) – We need only the yaw rate as we are mapping the 2D surface on which the robot moves. (w.r.t body/IMU frame)
    i. Yaw rate: W_z
c. LIDAR scans: Provides distances of obstacles in the 270° field of vision in front of the robot.
d. RGBD – Disparity and RGB images captured by robot's camera that provide depth and colour information

2.1. Symbols definitions

Symbol	Description
v_t	Linear velocity from Encoder sample
ω_t	Yaw rate from IMU sample
$X_t = \begin{bmatrix} x_t \\ y_t \\ \theta_t \end{bmatrix}$	Robot pose – (x,y) position on 2D plane and yaw orientation (θ) in world frame
X_t^s	LiDAR sensor pose
Z_t	LiDAR scan
Y_t	LiDAR scan's world frame projection
m	Occupancy grid map
τ_t	Difference between consecutive time stamps
$\mu_{t\|t}[k]$	Pose of k-th particle
$\alpha_{t\|t}[k]$	Weight (probability mass) of k-th particle
$im_{rgb,t}$	RGB image
$im_{d,t}$	Disparity image

2.2. Motion Model: Differential-drive Kinematics Model

We use the approximate Euler discretization form, assuming v_t, ω_t are constant during the τ_t period.

$$X_{t+1} = \begin{bmatrix} x_{t+1} \\ y_{t+1} \\ \theta_{t+1} \end{bmatrix} = f_d(X_t, v_t, \omega_t) = X_t + \tau_t \begin{bmatrix} v_t \cos(\theta_t) \\ v_t \sin(\theta_t) \\ \omega_t \end{bmatrix}$$

2.3. Observation Model: LiDAR Correlation Model

Likelihood model $p_h(Z_t|X_t^s, m)$ for LiDAR scan Z_t obtained from sensor pose X_t^s in occupancy grid m.
Scan's world-frame projection $Y_t = r(Z_t, X_t)$, via the robot pose X_t
Likelihood is proportional to the correlation between Y_t and m

$$p_h(Z_t|X_t, m) \propto corr(r(Z_t, X_t), m)$$

2.4. SLAM

Simultaneous Localization and Mapping
Localization: Given a map m, a sequence of control inputs $u_{0:T-1}$, and a sequence of measurements $Z_{0:T}$, infer the robot state trajectory $X_{0:T}$. Discussed in sec.2.5.

Mapping: Given a robot state trajectory $X_{0:T}$ and a sequence of measurements $Z_{0:T}$, build a map m of the environment. Discussed in sec.2.6.

Use a **particle filter** to maintain the pdf $p(X_t|Z_{0:T}, u_{0:T-1}, m)$ of the robot state X_t over time. Each particle $\mu_{t|t}[k]$ is a hypothesis on the state X_t with confidence $\alpha_{t|t}[k]$.

2.5. Trajectory Estimation: Particle Filter

- This is the **Localization** step of SLAM

- **Prediction step:** Use the inputs v_t, ω_t and motion model (sec.2.2) to obtain the predicted pdf $p_{t+1|t}(X_{t+1})$.
  Poses updated using velocity and yaw rate inputs; weights remain same. Multiple hypotheses are generated by introducing noise.

- **Update step:** Use the observation Z_{t+1} and observation model p_h (sec.2.3) to obtain the updated pdf $p_{t+1|t+1}(X_{t+1})$.

2.6. Occupancy grid mapping

- This is the **Mapping** step of SLAM.
- Occupancy grid m is a grid/vector whose i-th entry indicates whether i-th cell is free ($m_i = -1$) or occupied ($m_i = 1$).
- Let n be the total number of cells. These cells are pixels in 2D.
- Occupancy grid m is estimated probabilistically, given the robot trajectory $x_{0:t}$ and a sequence of observations $z_{0:t}$
- Independent Assumption: The cell values are independent conditioned on the robot trajectory.

$$p(m|z_{0:t}, x_{0:t}) = \prod_{i=1}^{n} p(m_i|z_{0:t}, x_{0:t})$$

2.7. Texture map

Use the RGB and Disparity images ($im_{rgb,t}, im_{d,t}$) and estimated robot trajectory to produce 2D color map of the floor surface.

Assigning depth to each RGB pixel:

- $d = im_{d,t}(i,j)$: value of pixel (i,j) of disparity image
- $dd = (-0.00304d + 3.31)$
- $z = \frac{1.03}{dd}$ (depth)
- $rgbi = \frac{526.37i + (-4.5 \times 1750.46)dd + 19276.0}{585.051}$
- $rgbj = \frac{526.37j + 16662}{585.051}$

This means that the pixel in the location $(rgbi, rgbj)$ of the RGB image has the depth of z

World frame transformation of disparity image pixels:

- Intrinsic parameters of depth camera:

$$K = \begin{bmatrix} fs_u & fs_\theta & c_u \\ 0 & fs_v & c_v \\ 0 & 0 & 1 \end{bmatrix}$$
$$= \begin{bmatrix} 585.05108211 & 0 & 242.94140713 \\ 0 & 585.05108211 & 315.83800193 \\ 0 & 0 & 1 \end{bmatrix}$$

Find the world frame coordinates of each disparity pixel. Segregate the floor pixels by applying threshold to the z-coordinate. Assign the corresponding (x,y) location in the map with the pixel's RGB value.

3. TECHNICAL APPROACH

3.1. Data Processing

3.1.5. Encoder:

- **Goal:** To obtain instantaneous linear velocity, $\boldsymbol{v_t}$ of the robot.
- Encoders count the rotations of the four wheels. Here, we have such data collected at regular times during the robot motion.
- Diameter of robot wheel = 0.254 m
  Number of ticks per revolution = 360
  Distance travelled per tick = 0.254/360 = 0.0022 m
- At any instance, the encoder counts, $[FR, FL, RR, RL]$ are the counts of ticks corresponding to front-right, front-left, rear-right and rear-left wheels respectively.
- These counts get reset after every reading.
- If $\tau_t = time_{t+1} - time_t$, then $[FR, FL, RR, RL]$ at $time_{t+1}$ correspond to the distances travelled by each wheel during the τ_t period.
- Let d_L, d_R be the distances and v_L, v_R be the velocities of left and right wheels respectively.

$$d_L = \frac{FL + RL}{2} \times 0.0022\, m$$
$$d_R = \frac{FR + RR}{2} \times 0.0022\, m$$

$$v_{L,t} = \frac{d_L}{\tau_t};\ v_{R,t} = \frac{d_R}{\tau_t}\ m/s$$

Robot velocity, $\boldsymbol{v_t} = \frac{\boldsymbol{v_{L,t}} + \boldsymbol{v_{R,t}}}{\boldsymbol{2}}$

3.1.2. IMU:

- **Goal:** To obtain instantaneous yaw rate, $\boldsymbol{\omega_t}$ of the robot.
- Yaw rate was directly available in the IMU data without any requirement for processing.
- However, the data was noisy and needed low pass filtering.
- Butterworth low pass filter with cut-off 10Hz and order 5, was used to filter the yaw rate data.

3.1.3. Hokuyo LiDAR:

- **Goal:** To obtain occupied and free space around the robot and use it for mapping.
- This is a horizontal LiDAR with 270^o degree field of view and maximum range of 30 m.
- It provides distances to the obstacles in the environment.
- Each LiDAR scan contains 1081 measured range values. That, the whole 270^o is divided equally into 1080 divisions

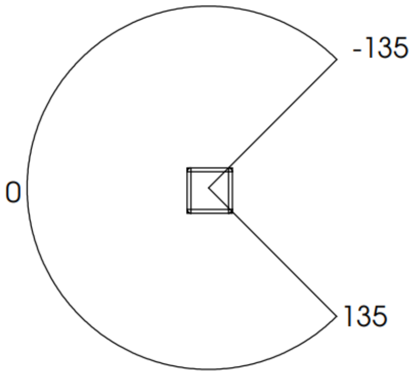


Figure 3.1: LiDAR field of vision with angles in degrees

- Filtering: Distances/ranges that are below 0.1m and above 30m are discarded as invalid/noisy data.
- Sensor frame 2D coordinates: As sensor looks the forward direction of the robot,
  x-axis – Forward direction
  y-axis – Left direction
  Angle of increment for each of the 1081 values = 270°/1080 = 0.25°
  Min angle = -135°; Max angle = 135°

  $\alpha_i = -135° + (i-1)0.25°, \qquad i = 1, 2, \ldots, 1081$
  r_i: Range (m) of obstacle at i-th angle
  $Sensor\ frame\ coordinates\ (in\ metres)$:
  $$sx_i = r_i \cos(\alpha_i)$$
  $$sy_i = r_i \sin(\alpha_i)$$
  $$i = 1, 2, \ldots, 1081$$
- The process was executed in parallel for all the LiDAR measurements collected throughout the robot motion.
- In the later stages, these sensor frame coordinates of the obstacles at each time instance would be converted into world frame coordinates using the corresponding estimated robot pose.

3.1.4. Kinect:

- **Goal:** Obtain RGB images and disparity images captured by the RGBD camera.
- The depth camera is located at (0.18; 0.005; 0.36) m with respect to the robot centre and has orientation with roll 0 rad, pitch 0.36 rad, and yaw 0.021 rad
- The image matrices would be used to add colour and texture to the map constructed using the estimated robot's trajectory.

3.1.5. Time synchronization:

- All the sensors mentioned above have different frequencies in recording their observations.
- For the SLAM problem, I chose the Encoder time stamps as the anchor and obtained remaining sensors data nearest to these time stamps.
- In case of Kinect (RBGD) data, disparity image timestamps were used as anchor as they provide the essential depth information.

3.2. Dead Reckoning

Noiseless trajectory: I computed the robot pose $X_t = \begin{bmatrix} x_t \\ y_t \\ \theta_t \end{bmatrix}$ at all timestamps using the motion model in sec.2.2. Initial pose is taken as [0, 0, 0]

Occupancy map using dead reckoning: Using the poses computed in noiseless trajectory, I performed probabilistic occupancy grid mapping, which will be explained in the later section.

3.3. Occupancy grid

The objective is to maintain a grid of map log-odds each cell whether being occupied or not. The grid depicts the X-Y 2D plane in the world frame and we consider the grid center as the origin (initial position of the robot)

Map parameters:
Map resolution: $res = 0.05\ m$
Boundaries along x-axis: $xmin = -30m, xmax = 30m$
Boundaries along y-axis: $ymin = -30m, ymax = 30m$
Map dimensions in terms of pixels:
$$(sizex, sizey) = \left(\left\lceil \frac{xmax - xmin}{res} + 1 \right\rceil, \left\lceil \frac{ymax - ymin}{res} + 1 \right\rceil\right)$$

- Let's denote the map grid with m and each cell in the map as m_i, where $i = 0, 1, 2 \ldots (sizex \times sizey - 1)$.
- For convenience, we are indexing the 2D map linearly. After processing, we reshape it into 2D matrix.
- Let's assume we have the robot trajectory X_t^{est} estimated using the particle filter, which will be discussed in the later section.
- Given a new LiDAR scan Z_t, transform it to the world frame using the robot pose X_t^{est}.
- For this, I pre-computed the body frame coordinates of the filtered LiDAR scans as described in sec.3.1.3. I assumed sensor and body frame to be the same.
- Once I have the body frame coordinates, the world frame coordinates of the LiDAR scans are:
  $$\begin{bmatrix} wx_{i,t} \\ wy_{i,t} \end{bmatrix} = R(\theta_t^{est}) \begin{bmatrix} sx_{i,t} \\ sy_{i,t} \end{bmatrix} + \begin{bmatrix} x_t^{est} \\ y_t^{est} \end{bmatrix}$$
  $$i = 1, 2, \ldots, 1081$$
  $$(sx, sy): Body\ frame\ coordinates\ of\ LiDAR\ points$$
  $$X_t^{est} = [x_t^{est}, y_t^{est}, \theta_t^{est}]^T: Robot\ pose$$
  $$R(\theta_t^{est}): 2D\ Rotation\ matrix\ (body\ to\ world\ frame)$$
- Then we scale these world coordinates (in m) to pixel locations of the map grid.
  Let the pixel coordinates be (wpx, wpy) for a LiDAR scan point (wx, wy)

  $$wpx_{i,t} = \left\lceil \frac{wx_{i,t} - xmin}{res} \right\rceil - 1$$
  $$wpy_{i,t} = \left\lceil \frac{wy_{i,t} - ymin}{res} \right\rceil - 1$$
- Similarly, the pixel coordinates (px^{est}, py^{est}) for the robot position (x^{est}, y^{est})

$$px_t^{est} = \left\lceil \frac{x_t^{est} - xmin}{res} \right\rceil - 1$$
$$py_t^{est} = \left\lceil \frac{y_t^{est} - ymin}{res} \right\rceil - 1$$

- As shown in Fig.3.2, I determined the cells that LiDAR beams pass through using the Bresenham's line rasterization algorithm.

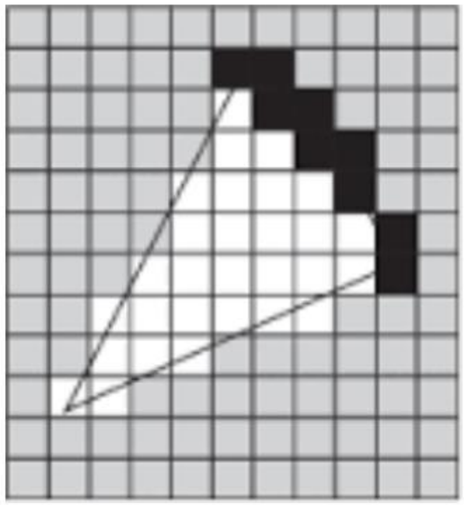


Figure 3.2: Free and occupied cells for a LiDAR scan

- Cells that the beams passed through are considered as free cells and those at which the beams stopped are considered as occupied as there are obstacles over there.
- Assume that the LiDAR sensor has a trust percentage of 80%.
- Hence the log-odds ratio would be $\log\left(\frac{80}{20}\right) = \log(4)$
- For each observed cell *I*, we decrease the log-odds if it was observed free or increase the log-odds if the cell was observed occupied
$$\rho_{i,t+1} = \rho_{i,t} \pm log4$$
- At the end of mapping, to avoid overconfident estimation, I applied threshold as follows: $-1000log4 < \rho_i < 1000log4$
- For visualization gray scale display of the log-odds map in sufficient.
- To recover the exact probabilities of occupancy of each cell, we apply logistic sigmoid function.
$$\gamma_{i,t} = p\left(m_i = 1 \middle| z_{0:t}, x_{0:t}\right) = \frac{\exp\left(\rho_{i,t}\right)}{1 + \exp\left(\rho_{i,t}\right)}$$

3.4. Particle Filter SLAM

Simultaneous Localization and Mapping using Particle Filter
Each particle $\mu_{t|t}[k]$ is 3x1 real number vector, represents a possible 2-D position (x, y) and orientation θ of the robot.

Prediction: For every particle $\mu_{t|t}[k], k = 1, 2, \ldots N$, using the motion model in sec.2.2, compute:
Poses updated; weights remain same
$$\mu_{t+1|t}[k] = f_d\left(\mu_{t|t}[k], v_t + \epsilon_t^{(v)}, \omega_t + \epsilon_t^{(\omega)}\right)$$
$$\alpha_{t+1|t}[k] = \alpha_{t|t}[k]$$

$\epsilon_t := \begin{bmatrix} \epsilon_t^{(v)} \\ \epsilon_t^{(\omega)} \end{bmatrix} \sim N(0, \begin{bmatrix} \sigma_v^2 & 0 \\ 0 & \sigma_\omega^2 \end{bmatrix})$ is 2-D Gaussian motion noise

Noise is added to inputs to generate different pose hypothesis for each particle.

Update: Particle poses remain same and the weights are scaled by the observation model and normalized to get summation one.
$$\mu_{t+1|t+1}[k] = \mu_{t+1|t}[k]$$
$$\alpha_{t+1|t+1}[k] \propto p_h(Z_{t+1}|X_t, m)\, \alpha_{t+1|t}[k]$$

LiDAR Correlation Model: $\boldsymbol{p_h(Z|X, m)}$
Recall from sec.2.3:
$$p_h(Z_t|X_t, m) \propto corr(r(Z_t, X_t), m)$$

For each particle *k*, transform the scan Z_{t+1} to the world frame using $\mu_{t+1|t}[k]$.

- Find all cells $y_{t+1}[k] = r\left(Z_{t+1}, \mu_{t+1|t}[k]\right)$ in the grid corresponding to the scan. Here $r(.)$ operation is nothing but finding free and occupied cells in the current LiDAR scan, using Bresenham's algorithm.
- Update the particle weights using the scan-map correlation:
$$\alpha_{t+1|t+1}[k] \propto corr(y_{t+1}[k], m)\, \alpha_{t+1|t}[k]$$
- Finding correlation:
$$corr(y, m) = corr(r(Z, X), m)$$

$y = r(Z, X)$, is the collection of free and occupied cells on the grid with respect to current LiDAR scan.
$$corr(y, m) = \sum_i I\{y_i = m_i\}$$
$$I\{y_i = m_i\} = \begin{cases} 1, if\ y_i = m_i \\ \quad 0, else \end{cases}$$

We are counting the number of cells that are classified into the same category (occupied or free) by both LiDAR scan and the map.
Hence, the prediction (particle pose) that has high correlation with the observation (LiDAR scan) gets higher weight in the update step.

- Pose improvement:

This is not part of the conventional particle filter implementation, but a strategy to pick the best correlation.
For each particle, we deterministically perturb it's 2D position generating 9x9 grid positions around its actual position. We compute correlation for all the 81 perturbed positions and pick the maximum. The corresponding position adjustment is applied to the particle.

Resampling:
As the algorithm progresses, some particles will obtain near zero weights which means that the robot has very less probability to be in those poses. Hence, through resampling we pick new set of *N* particles from the most probable poses.

Given particle set $\{\mu_{t|t}[k], \alpha_{t|t}[k]\}$, resampling is applied if the *effective number* of particles: $N_{eff} := \frac{1}{\sum_{k=1}^{N}(\alpha_{t|t}[k])^2}$ is less than a threshold. I chose the threshold as $N/2$

- Draw $j \in \{1, 2, \dots, N\}$ independently with replacement with probability $\alpha_{t|t}[j]$.
- Add $\mu_{t|t}[j]$ with weight $\frac{1}{N}$ to the new particle set.
- Repeat N times.

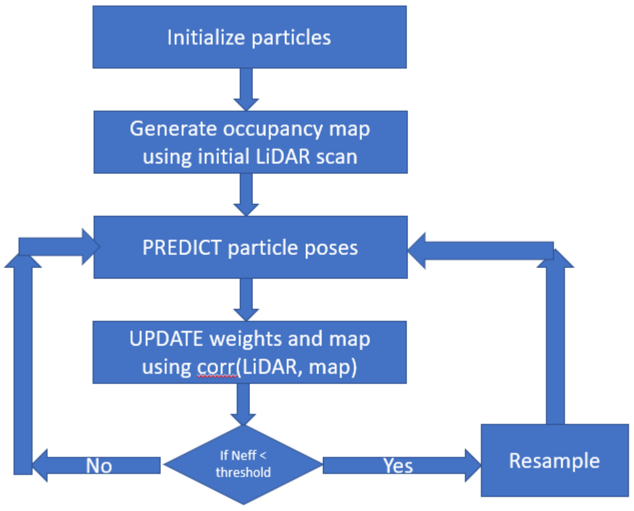

Initialize particles
Generate occupancy map using initial LiDAR scan
PREDICT particle poses
UPDATE weights and map using corr(LiDAR, map)
If Neff < threshold
No
Yes
Resample

Figure 3.3: Flow diagram of Particle filter SLAM

3.5. Texture Map

- As mentioned in sec.2.7;
  - Compute depth (z) of each disparity pixel.
  - As there is a shift between disparity and RGB pixel locations, we use the provided relations to compute the corresponding RGB pixel indices ($rgbi, rgbj$). Discard the invalid indices.
- Using intrinsic parameters K and depth z, calculate the optical frame coordinates.

$$\begin{bmatrix} x_o \\ y_o \\ z_o \end{bmatrix} = K^{-1} \begin{bmatrix} rgbi * z \\ rgbj * z \\ z \end{bmatrix}$$

- Convert optical frame to regular camera frame.

$$({}_{reg}R_{opt}) = \begin{bmatrix} 0 & 0 & 1 \\ -1 & 0 & 0 \\ 0 & -1 & 0 \end{bmatrix}$$

$$\begin{bmatrix} x_r \\ y_r \\ z_r \end{bmatrix} = ({}_{reg}R_{opt}) \begin{bmatrix} x_o \\ y_o \\ z_o \end{bmatrix}$$

- Convert regular camera frame to body frame.
  Position of camera w.r.t robot center: $(0.18, 0.005, 0.36)$
  Orientation of camera w.r.t robot center: $(roll\ 0, pitch\ 0.36, yaw\ 0.021)\ radian$
  $i.e., (\emptyset = 0, \theta = 0.36, \varphi = 0.021)\ radian$

$$({}_{body}R_{cam}) = R_z(\varphi)R_y(\theta)R_x(\emptyset)$$

$$({}_{body}R_{cam}) = \begin{bmatrix} cos\varphi & -sin\varphi & 0 \\ sin\varphi & cos\varphi & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} cos\theta & 0 & sin\theta \\ 0 & 1 & 0 \\ -sin\theta & 0 & cos\theta \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & cos\emptyset & -sin\emptyset \\ 0 & sin\emptyset & cos\emptyset \end{bmatrix}$$

$$\begin{bmatrix} x_b \\ y_b \\ z_b \end{bmatrix} = ({}_{body}R_{cam}) \begin{bmatrix} x_r \\ y_r \\ z_r \end{bmatrix} + \begin{bmatrix} 0.18 \\ 0.005 \\ 0.36 \end{bmatrix}$$

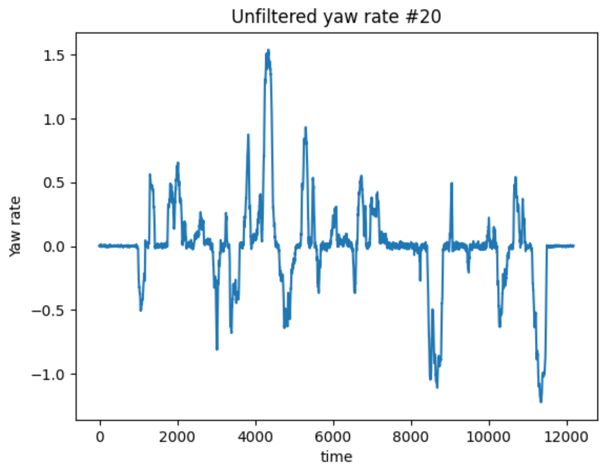
- Convert body frame to world frame, using robot's pose $[x_t^{est}, y_t^{est}, \theta_t^{est}]$ estimated by maximum weight particle.
  Since robot has only XY translation and yaw rotation, z-coordinate remains same. We can apply 2D transformation to the (x,y) coordinates.

$$\begin{bmatrix} x_w \\ y_w \\ z_w \end{bmatrix} = \begin{bmatrix} R(\theta_t^{est}) \begin{bmatrix} x_b \\ y_b \end{bmatrix} + \begin{bmatrix} x_t^{est} \\ y_t^{est} \end{bmatrix} \\ z_b \end{bmatrix}$$

- Initialize TEXTURE_MAP grid with certain resolution and boundaries. Scale the computed world coordinates of the RGB pixels according to this grid and assign add the RGB colors to the location in the TEXTURE_MAP.

4. RESULTS

4.1. *Yaw rate filtering*



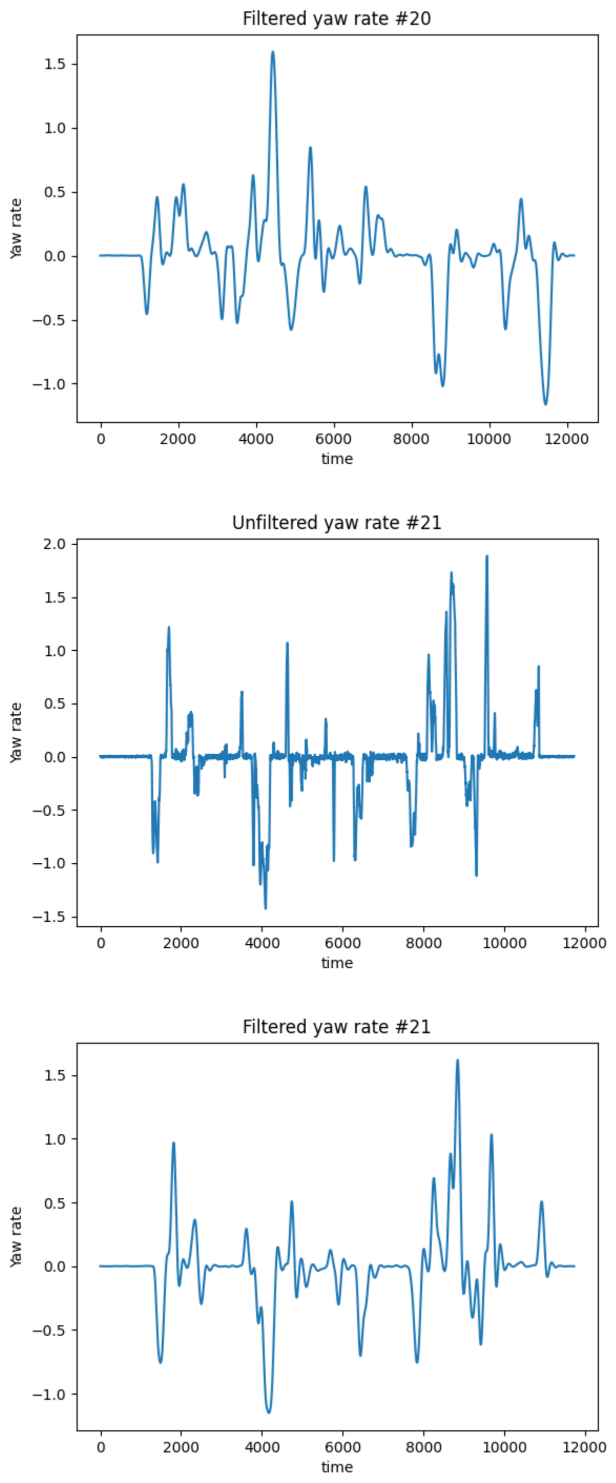


Figure – 4.1(i-iv): *Filtered and unfiltered Yaw rates for datasets #20, #21*

4.2. Dead reckoning

<u>Trajectory:</u>

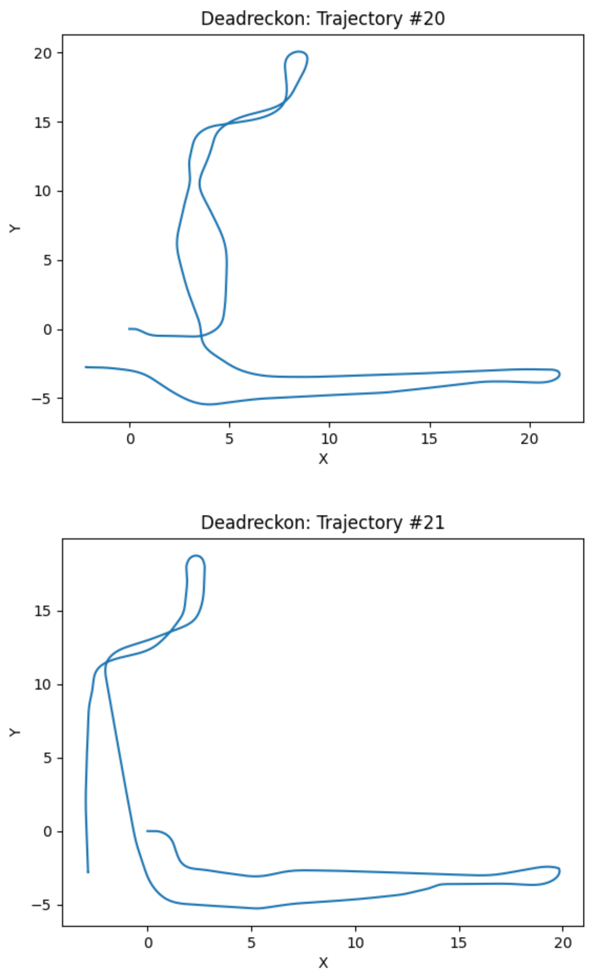
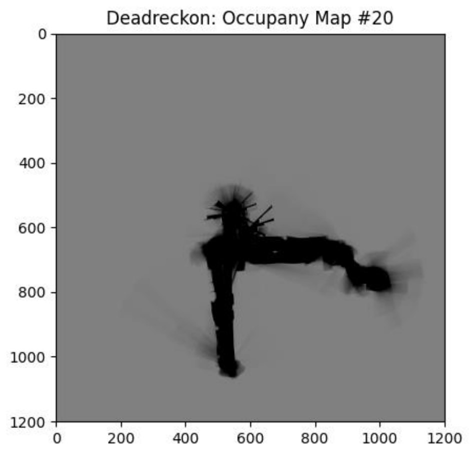


Figure – 4.2(i-ii): *Dead reckoning trajectory plots of datasets #20, #21*

<u>Occupancy Map:</u>



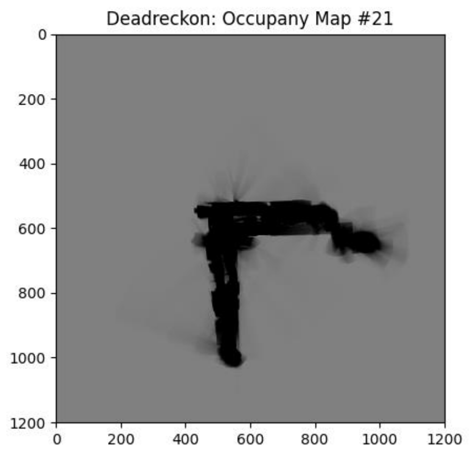
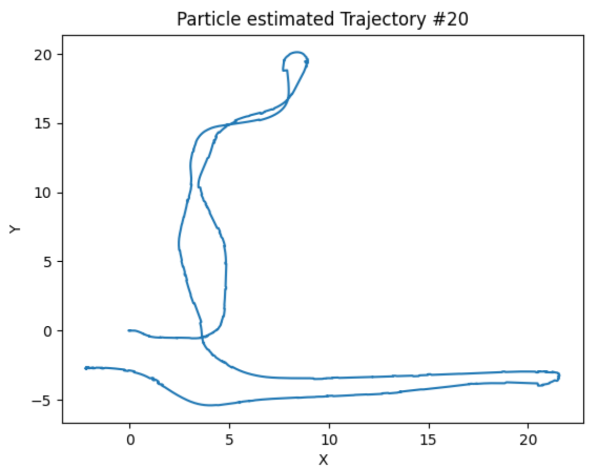


Figure – 4.3(i-ii): *Dead reckoning occupancy maps of datasets #20, #21*

4.3. Particle Filter SLAM

Number of particles = 500. Update step was run for every five Predict steps.

Trajectory:



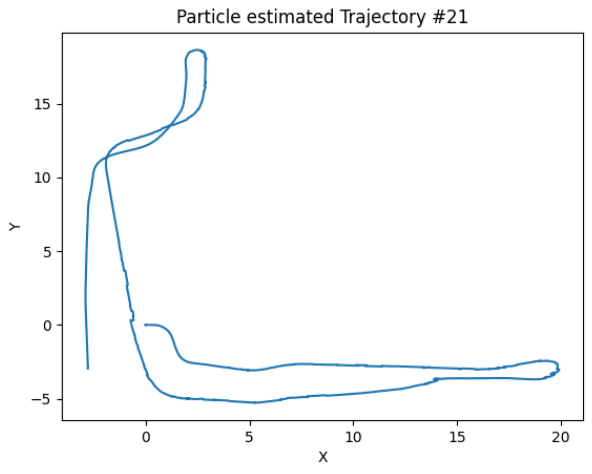
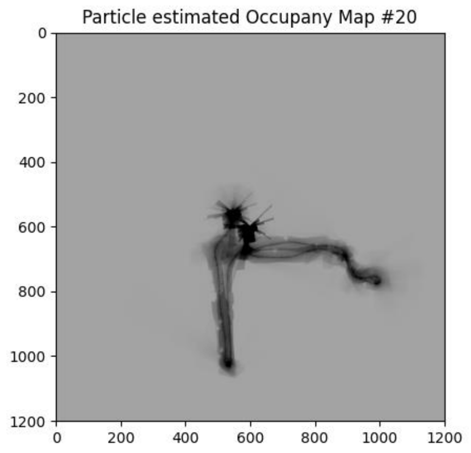


Figure – 4.4(i-ii): *Particle estimated trajectories of datasets #20, #21*

Occupancy Map:



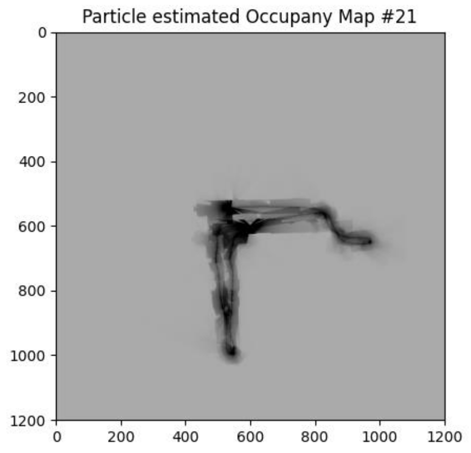
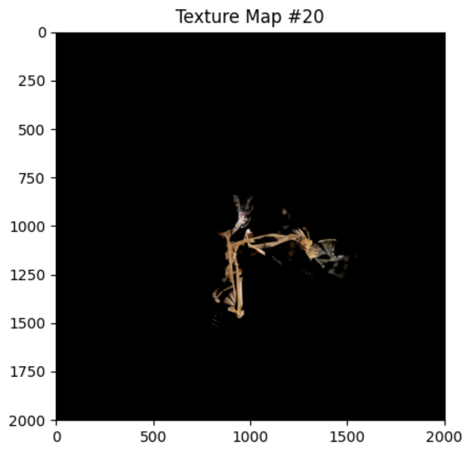


Figure – 4.5(i-ii): *Particle estimated occupancy maps of datasets #20, #21*

4.4. Texture Map

Texture maps are generated using the particle estimated trajectories.



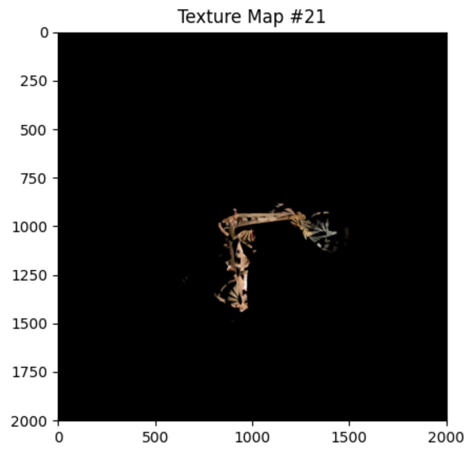
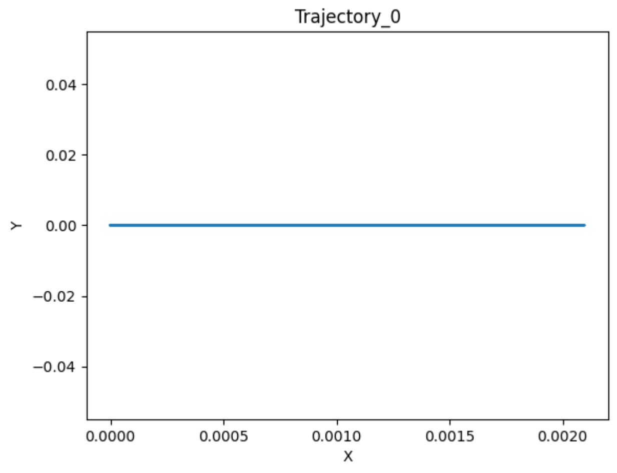


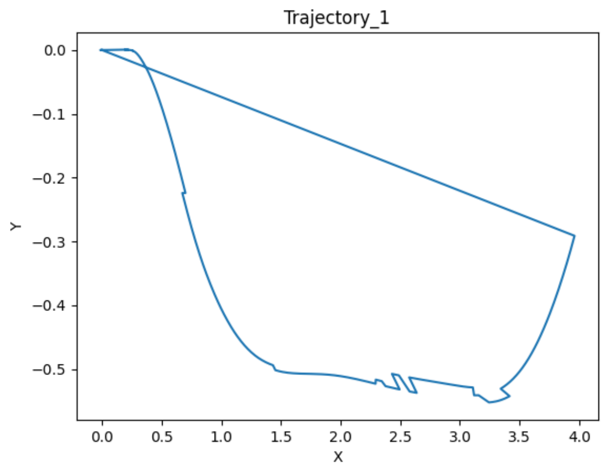
Figure – 4.6(i-ii): *Texture maps of datasets #20, #21*

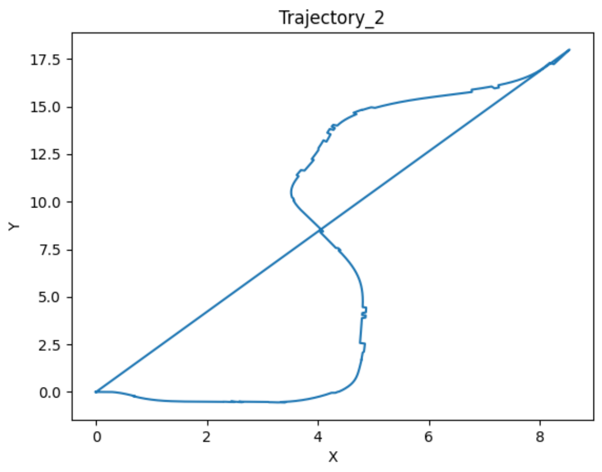
4.5. Over the time plots

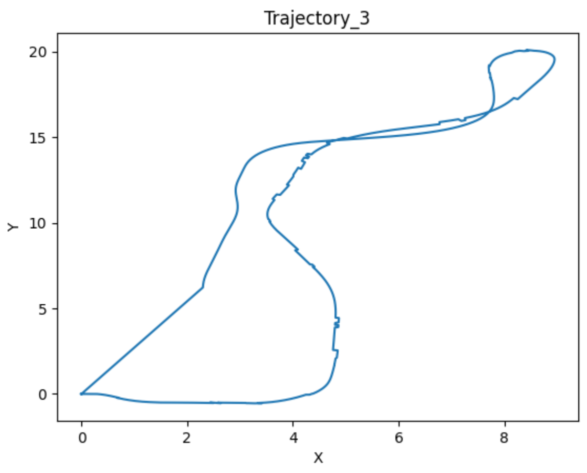
Plots are shown for dataset #20

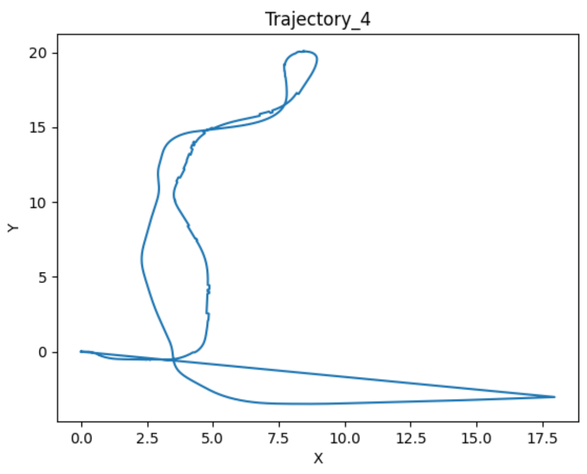
<u>Trajectory: (X-Y plots)</u>

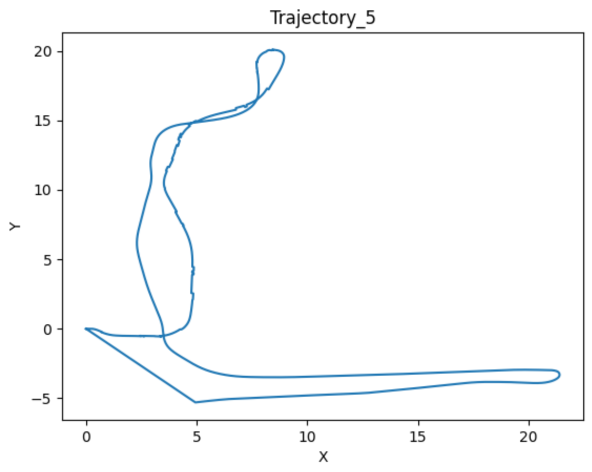


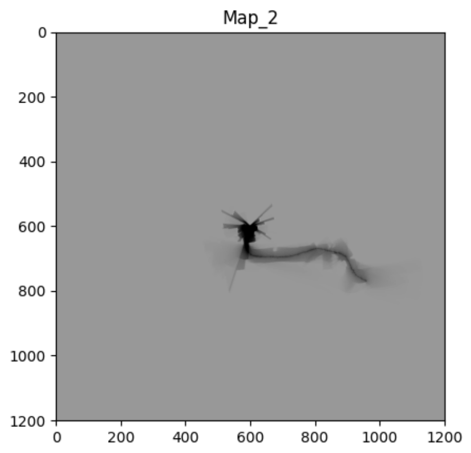




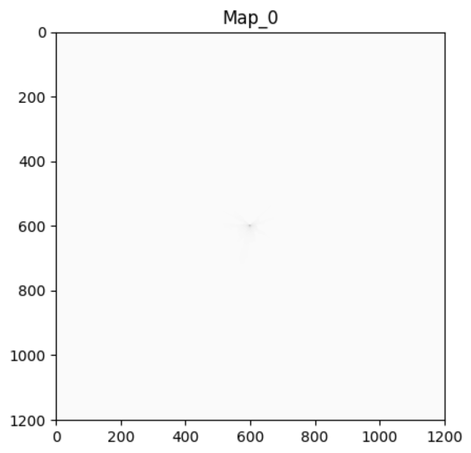


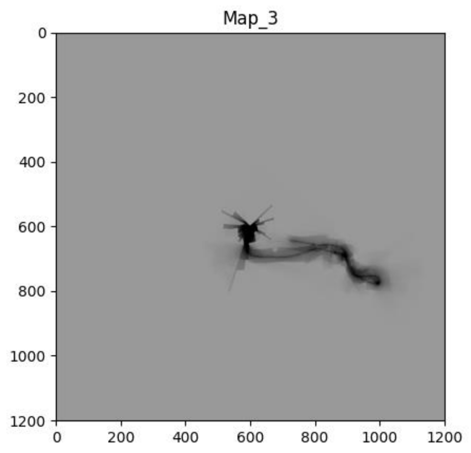


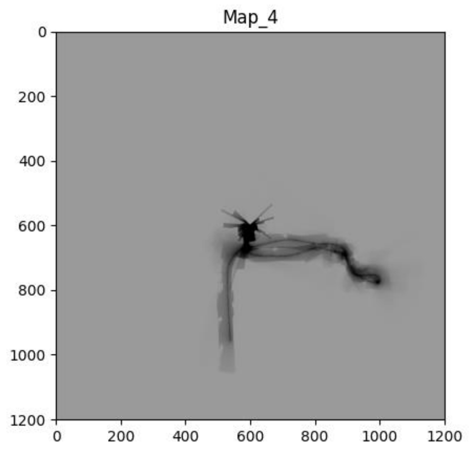


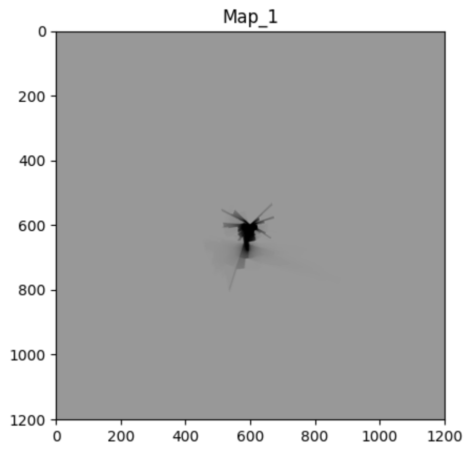


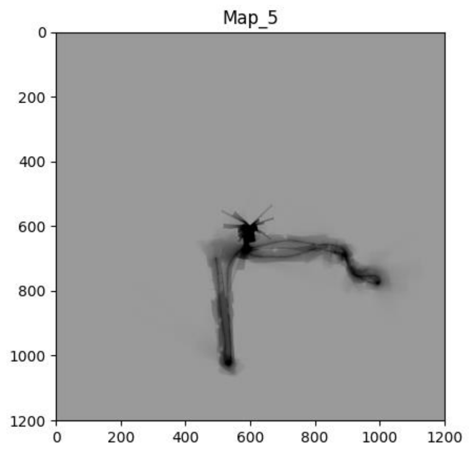
Occupancy Map:

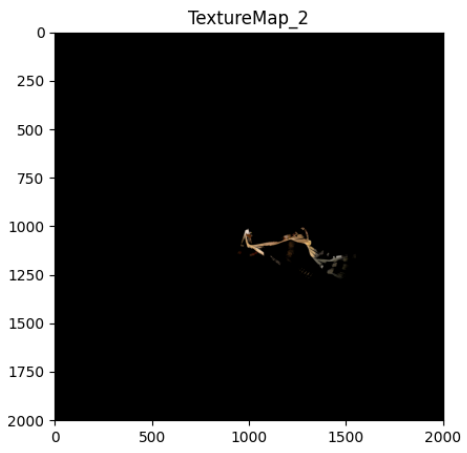




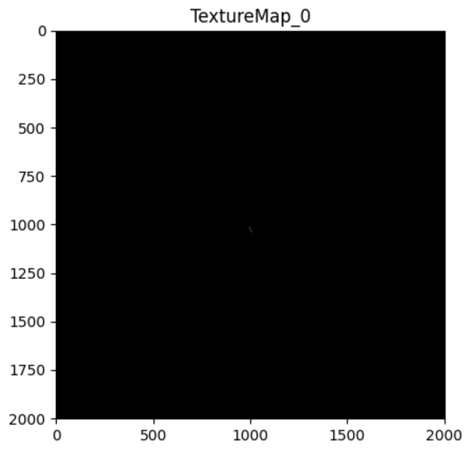


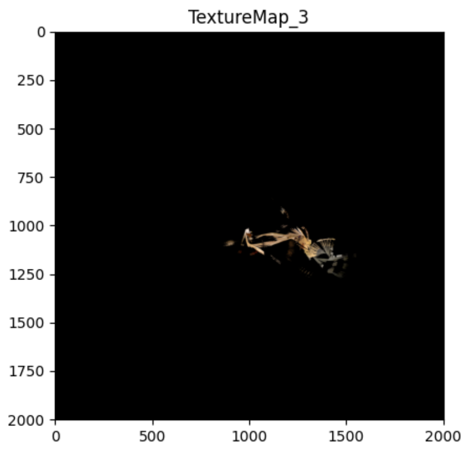


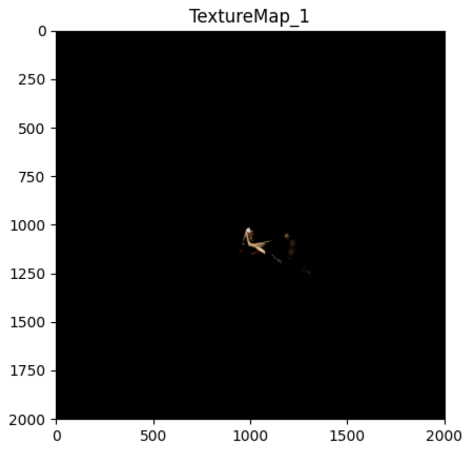

Map_5

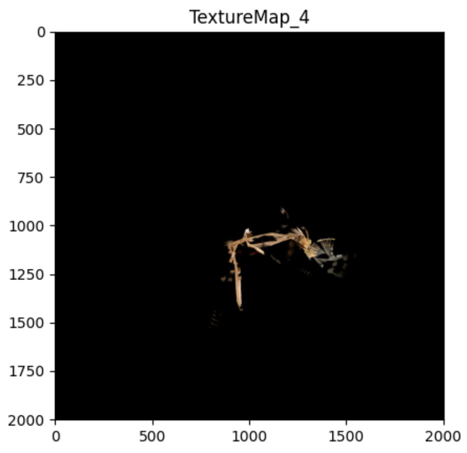

TextureMap_2

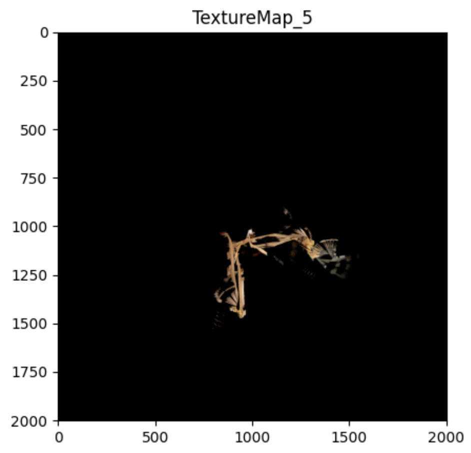
Texture Map:


TextureMap_0


TextureMap_3


TextureMap_1


TextureMap_4



4.6. Discussion

- The particle filter SLAM algorithm runs correlation computation for each particle sequentially.
- I chose 500 particles and to speed up the process, I set the update step to occur every five steps.
- The estimated robot trajectories have finer information of little deviations of the robot, which seems to be more accurate than the dead reckon trajectory. But since, we do not have the ground truth, we cannot surely say this.
- The occupancy maps generated by particle filter are less detailed compared to the dead reckon maps. This is because of running weight update steps for every five prediction steps, the pixel intensities might be less compared to dead reckon maps.
- The floor texture mapping was also not perfect. There must be some issue with accurately decoding the depth values and mapping them to the right RGB pixels. Or more experimentation could have been done with the threshold for classifying the floor pixels.
- For the overtime plots, better way could have been followed to show the progress of robot trajectory.
- Overtime plots were generated for just dataset #20 due to time constraints.

ACKNOWLEDGEMENT

I would like to thank Prof. Nikolay Atanasov, TAs – Sambaran Ghosal and Shrey Kansal for guiding me with this project.

REFERENCES

[1] https: //natanaso.github.io/ece276a
[2]https://hokuyo-usa.com/application/files/8115/8947/8333/UTM-30LX_Specifications_Catalog.pdf
[3]https://github.com/yashv28/Particle-Filter-SLAM/blob/master/bresenham2D.py
[4]https://www.delftstack.com/howto/python/low-pass-filter-python/