ECE 276A Project 2: Particle Filter SLAM

Unay Shah **PID:** A59015777

Keywords:

Data source: https://drive.google.com/drive/folders/1_SLyxhnkSIKVsb_cdXW8BEZlQEDXkuRx

Abstract:

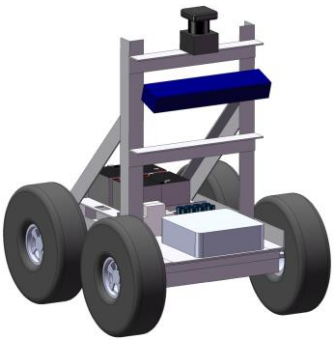
Simultaneous localization and mapping (SLAM) is a way to map an area while simultaneously tracking an agent that is traversing the area. We have a differential motion model robot with encoder wheels, IMU, LiDAR, and RGB and Kinect cameras, that is traversing an unknown environment and its sensors are capturing data from its surroundings. These data points can be used to create a map of the environment. Various techniques like particle filtering, scan-grid correlation, variations in motion model etc. help acquire and improve the trajectory. After a trajectory is obtained, the floor image corresponding to each location is overlaid to create a map of the environment.

Introduction

Simultaneous localization and mapping (SLAM) is used to map unknown environments using a robot carrying various sensors, continuously collecting data of its surroundings. These data points need to be processed in order to create a clean and accurate map. The robot might face some disturbances or the sensor may collect noisy data, which needs to be cleaned. This is where particle filter SLAM along with various other techniques used in this project are useful. The robot provided is has a differential drive motion model and carries an IMU (Inertial Measurement Unit), a LiDAR (Hokuyo UTM-30LX) and a Kinect (RGBD camera). Along with this, it has encoder wheels, which can be used to understand its linear motion through its velocity. The angular velocity in yaw is used from IMU for this project, LiDAR is used to measure distance from the surroundings, and the Kinect is pointed downwards to capture the floor. First dead reckoning is used to obtain a rough trajectory of the path while transforming the LiDAR data into real world. This is followed by a noise based prediction and particle filtering of the data. A number of particles are introduced at each step and perturbed slightly. The real world coordinates of LiDAR are correlated with the existing map to identify the most likely motion of the robot among the introduced particles and motion is registered in that direction, along with its LiDAR measurements. A log probability map is used to store the current path observed by the robot and the particle with highest correlation with the map from the previous timestamp. The particles are used to obtain possible particles at the next timestamp using perturbations in the robot's linear and angular velocity. The correlation of each particle with the map is used to get the confidence for each particle and this is updated at every timestamp. Particles are resampled regularly to keep only those with good correlation. The trajectory obtained after the robot traverses the entire path is used for texture mapping, which refers to projecting the images obtained from the RGBD camera onto the path of the robot. Depth information is used alongside the RGB values to obtain the projection of the images on the map and a threshold for the depth is set to plot only the part of the image below a certain depth.

Problem Formulation

We start with processing the sensor data. We are provided with data from Encoder wheels and IMU, using which we can get the trajectory of the robot and create a SLAM map. This data, along with the image data which will be used in the future are captured over similar time frames, but



at different times. These need to be synchronized, which is done by iterating over the encoder timestamps and finding the timestamp of other quantities which are closest to it. The index of the corresponding timestamp is stored in a map.

```
map[encoder_ts[i]] = argmin(|encoder_ts[i]-imu_ts|)
```

The encoder data is available as tics of its wheels, which is reset after each reading. The readings take place at 40Hz. The wheel has a diameter of 0.254m and there are 360 ticks per revolution. So the distance travelled by the wheel per tick can be given by:

$$distance\ per\ tick = \frac{\pi d}{360} = \frac{0.254\pi}{360} = 0.0022\text{meters}$$

The distance travelled by the wheels is given in the order of [FR, FL, RR, RL] and distance travelled by each side can be calculated by

$$\frac{FR + RR}{2} \times 0.0022\ and\ \frac{FL + RL}{2} \times 0.0022$$

Given that the robot has a differential motion model, the linear velocity of the robot can be calculated by

$$V_{linear} = \frac{V_L + V_R}{2}$$

Angular velocity corresponding to yaw can be retrieved from the IMU.

The current pose of the robot can be represented in 2D as $[x, y, \theta]$ and for the prediction step, we can use the current pose along with linear and angular velocity measurements to get pose at next timestamp

$$\begin{bmatrix} x_{t+1} \\ y_{t+1} \\ \theta_{t+1} \end{bmatrix} = \begin{bmatrix} x_t \\ y_t \\ \theta_t \end{bmatrix} + dt \begin{bmatrix} v_t \cos(\theta_t) \\ v_t\ sin(\theta_t) \\ \omega_t \end{bmatrix} \quad \text{--1}$$

```
(where dt is the difference in timestamp of
consecutive encoder readings)
```

Particle filter to sample the particles is done based on whether the "effective" number of particles are below a threshold. If N_{eff} falls below $0.1N$, then the particles are resampled. This is calculated by the particle weights.

$$N_{eff} = \frac{1}{\sum_i w_i^2} > 0.1N$$

This can be used to obtain the dead reckoning trajectory of the robot. The dead reckoning trajectory can be used to obtain a preliminary LiDAR map of the robots motion using the LiDAR data along with the robot pose. LiDAR provides a 270° view of the surroundings and the coordinates provided by the LiDAR need to be transformed into the world frame. First it is converted into the body frame of the robot. I have considered the robot center to be at 0.29833m above the back wheels in the center of the frame.

$$x = \alpha \cos(\theta) + 0.29833, y = \alpha \sin(\theta) + 0$$

```
Where α is the angle of the LiDAR from its front
axis
```

This is done using the rotation matrix of the robots pose with the position offset of the LiDAR sensor. The rotation matrix to transform endpoints received from the LiDAR to the world frame is calculated from the robot pose as follows

$${}_wR_b = \begin{bmatrix} \cos\theta & -\sin\theta & x \\ \sin\theta & \cos\theta & y \\ 0 & 0 & 1 \end{bmatrix}$$

World frame coordinates = Body Coordinates x ${}_wR_b$

The coordinates obtained give the location of obstacle in the world frame or are the maximum distance that the robot can see. A similar approach is used to obtain obstacle locations for SLAM, along with adding noise in angular and linear velocity. In addition, SLAM includes processing on particles and the processing of weights and correlation of particles. A correlation function calculates the match between a particle's location and the occupancy map. This correlation is used to assign a weight to the particles. This weight is multiplied with existing weights and normalized.

The trajectory obtained from particle filter SLAM is used for texture mapping of the floor. Given colour and depth data from RGBD camera, we can transform these colours into the world coordinates from the sensor frame. The depth camera is located at $[0.18, 0.005, 0.36]$with respect to the robot center and has roll 0rad, pitch 0.36rad, and yaw 0.021rad. Intrinsic camera parameters are

$$K = \begin{bmatrix} fs_u & fs_\theta & c_u \\ 0 & fs_v & c_v \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 585.051 & 0 & 242.941 \\ 0 & 585.051 & 315.838 \\ 0 & 0 & 1 \end{bmatrix}$$

Depth data is used as follows to obtain RGB coordinates

$$dd = -0.00304d + 3.31$$

$$depth = \frac{1.03}{dd}$$

$$rgbi = \frac{526.37i + (-4.5 \times 1750.46)dd + 19276.0}{585.051}$$

$$rgbj = \frac{526.37j + 16662}{585.051}$$

Where i and j are pixel coordinates in the image and d is depth of the pixel

These coordinates are converted from the sensor frame to the optical frame, followed by the body frame and finally into the world frame

$${}_oR_s = \mathrm{K}^{-1}(\begin{bmatrix} rgbi \\ rgbj \\ 1 \end{bmatrix} \times \text{depth}$$

$${}_{\mathrm{b}}\mathrm{R}_o = \mathrm{R} \times \left(\begin{bmatrix} 0 & -1 & 0 \\ 0 & 0 & -1 \\ 1 & 0 & 0 \end{bmatrix} \times {}_{\mathrm{b}}\mathrm{R}_{\mathrm{o}} \right) + \begin{bmatrix} 0.18 \\ 0.005 \\ 0.36 \end{bmatrix}$$

$${}_wR_b = \left({}_bR_o \times \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \right) + \begin{bmatrix} x \\ y \\ 0 \end{bmatrix}$$

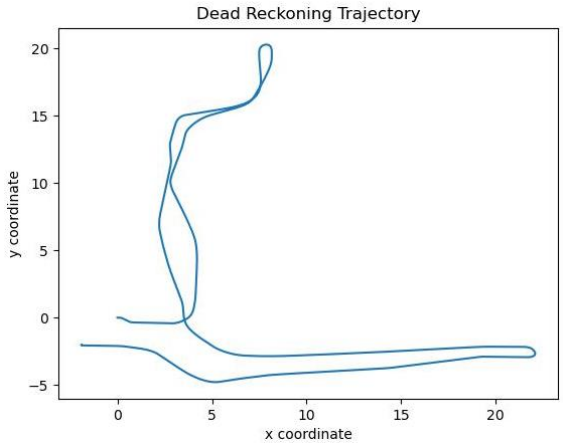
Where [x, y, θ] is the pose of the robot

This gives the coordinate mapping of the pixels into an image which can be plotted using the RGB data.

Technical Approach

Dead Reckoning

We start by calculating the linear velocity of the robot using encoder data and isolating the angular velocity from the IMU data. These are used to obtain the dead reckoning trajectory of the robot. At the start, we assume identity pose $[0 \quad 0 \quad 0]$ for $[x \quad y \quad \theta]$ and using formula 1 with the velocity of the robot, we obtain the next pose using current pose.



Occupancy Grid with Dead Reckoning

We use the poses of the robot obtained from dead reckoning to plot an approximate LiDAR scan. We convert the LiDAR coordinates into body frame and then for every pose, obtain the world frame coordinates. These can be plotted using the Bresenham function that is provided. This plots the LiDAR paths from the robot location to the end point that the LiDAR sees. This gives a rough estimate of what can be expected from particle filter SLAM.

I also tried to initialize 10 particles with some noise and plot their dead reckoning trajectory just to see variations that could be seen.

Particle Filter SLAM

For SLAM, 100 particles are initialized with identity pose and a map of the first LiDAR scan is created. This scan is a binary map with values 1 or 0 for unoccupied or occupied/unchecked respectively. Another empty occupancy grid map is created. For subsequent timestamps, the LiDAR scans are transformed into the world frame based on the poses of all the 100 particles. Correlation is carried out between the existing binary occupancy grid and the LiDAR scan end points along a particular matrix. This correlation is used to obtain the weights for each particle. A particle that has high correlation with the existing map shows greater probability of having followed the correct path than other particles. The matrix used alongside tries to fit in the particle into offset grids around the position in the occupancy map. Using this, we can identify if the particle would be more suitable to occupy a space that is slightly offset from the current position (x, y).

The correlation for each particle on the map is calculated using a 9x9 matrix, using the LiDAR end points in the world frame obtained by the particle's transformation matrix. Maximum values of correlation for each particle in the 5x5 grid is obtained. This value can be used to add an offset to the position of the particle. (I obtained worse results using this, and hence omitted it, showing results at the end). The maximum correlation for each particle is multiplied with its existing weight and then all the weights are normalized by dividing by the mean.

$$curr\_weight = curr\_weight \times \max\_correlation$$

$$weights = \frac{weights}{\sum weights}$$

Other possible approaches that I tried are

$$curr_{weight} = curr_{weight} \times \left(\frac{\mathrm{e}^{maxcorr}}{1 + \mathrm{e}^{\mathrm{maxcorr}}} \right)$$

But using this, the particles did not seem to decay.

The particle that has the maximum weight in the current iteration is used to plot the map for the current timestamp. The Bresenham function is used to find all points from the robot to the LiDAR end point of this particle. For points which are unoccupied, $4 \times \log(4)$ is subtracted from the occupancy map. For the end points, which imply walls or occupied cells, the same value is added. Using the occupancy grid, the binary is updated to have 1s where occupancy grid has negative values and 0s where the values are positive. This is done to increase correlation in empty spaces and decrease it in occupied spaces. The shifting of positions and identifying the most correlated particle correspond to the update step.

To ensure that all particles are used for the SLAM through all iterations, particles with lower weights should be dropped. All remaining particles are resampled based on their weights and then these are used for the next prediction. After resampling, they are all assigned equal weights.

For the next prediction, noise is added to all the particles. One of the ways this can be done is by adding a random normal value to the linear and angular velocity and then use the current particles to predict their next corresponding poses. Another way is to add random normal noise to the position and angle of the robot. Using these perturbed particles, we again obtain the LiDAR projections for all the particles at a new position and try to identify the most likely particle locations.

Texture Mapping

Once we obtain the most likely trajectory from SLAM, we can use this trajectory, which contains the pose of the robot at each timestamp, to transform the images of the environment. The images are transformed using the camera orientation and position data, along with the depth data from Kinect. World coordinates are obtained using rotation matrices corresponding to the pose of the robot at each timestamp. A threshold value is set for the depth under which the images should be projected, as the images also contain walls. Texture mapping gives an RGB image of the floor and the surroundings that the robot traverses.

Results

Starting with dead reckoning, gave me a rough idea about the shape to expect and visualize the motion of the robot. Carrying out dead reckoning with noisy particles showed what variation would take place when adding noise, even though this noise was on the pose of the robot obtained from the sensors and not the predicted one. The LiDAR scan obtained from dead reckoning served the same purpose of helping visualize the possible end result.

Performing particle filter SLAM started refining various edges of the graph. I tried to fine tune various parts of this algorithm. Changing the correlation map to a 5x5 matrix instead of a 9x9 reduced some level of variation and rotation in the path traversed by the particles. I tried 3 combinations of noise: $[0.1, 0.1, 0.01]$, $[0.01, 0.01, 0.001]$, $[0.001, 0.001, 0.001]$. The first set of values gave a lot of variation in the path being detected, while the next two gave similar paths. I noticed that after a certain value, noise no longer affects the prediction, and this is expected once the perturbations get too small to realise.

I tried adding noise to the particle pose instead of the velocities. The results showed that adding noise to the pose of the robot gave more controllable results, as compared to noise on the velocity, which caused relatively larger changes. Various parameters needed to be fine-tuned to use noise with velocity to predict the next step.

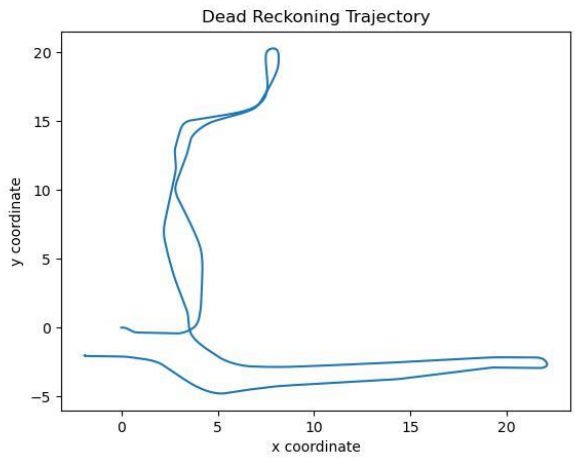
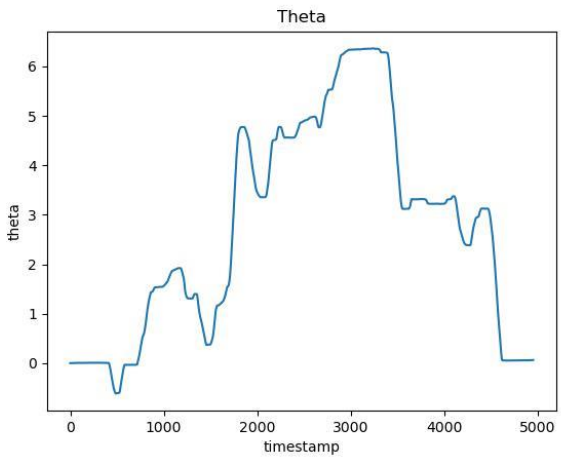
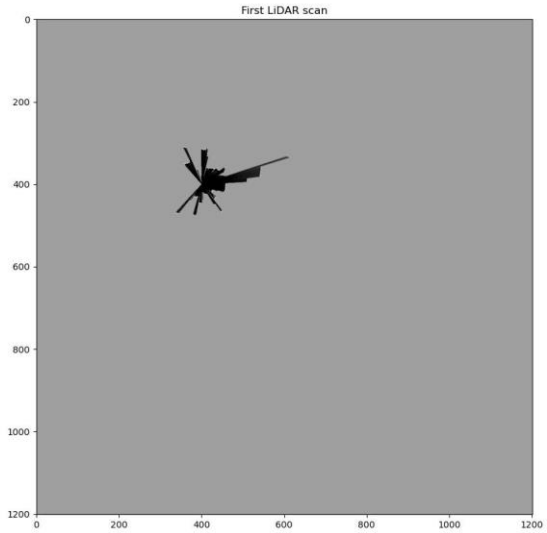
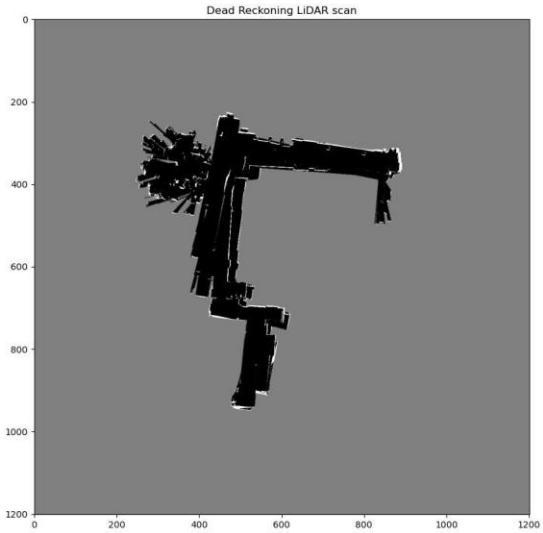
Changing the weight updating function to sigmoid or any variation of exponential function led to particles never going below N_{eff}.This led to no particles getting eliminated and all of them having relatively high weights. The path hence obtained had more variation towards the later part of the plot.

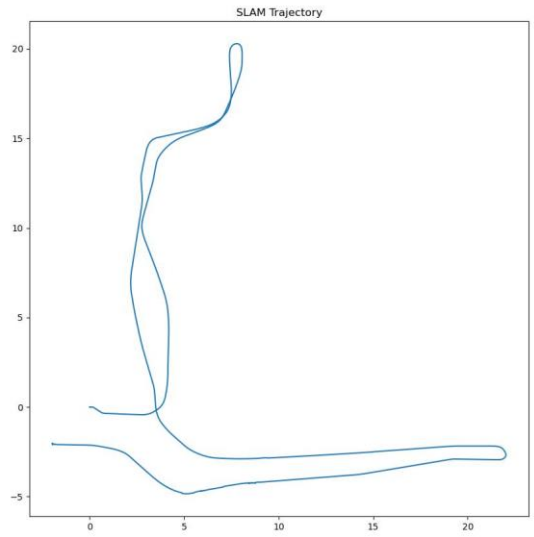
Setting the threshold of N_{eff} to 20% of N, instead of 10% of N, gave slightly better results in some cases, while no improvements in others.

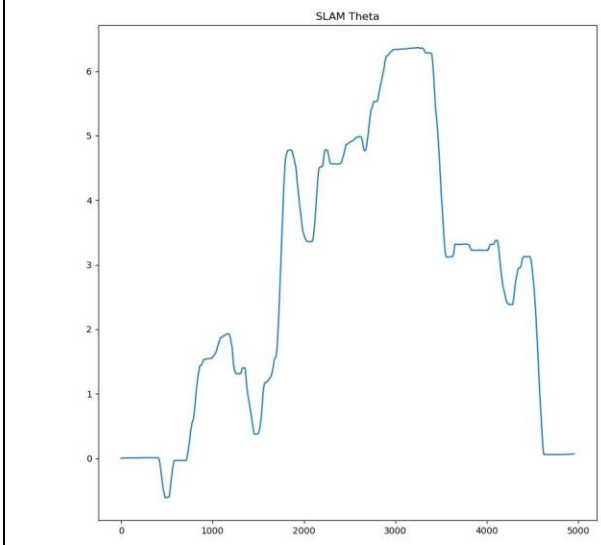
For the final plot of SLAM, there is a noisy rotation when the robot is returning to its initial location and taking the last turn, which is causing rotation in the orientation of the corridor in the plot. Some pillars are visible in the SLAM output in the last corridor.

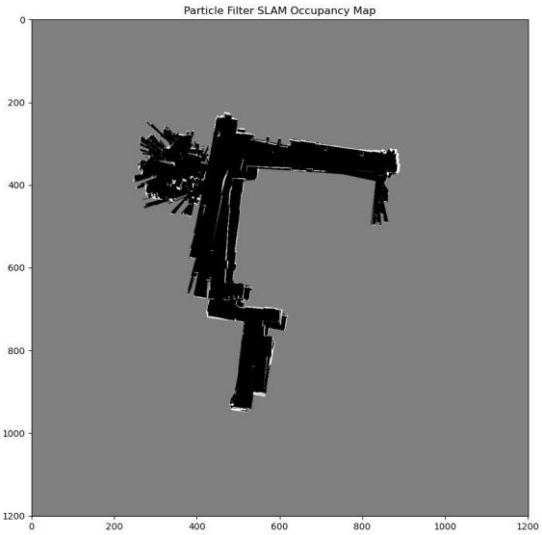
I was able to vectorise the correlation function, but speed did not increase due to Bresenham being the bottle neck. I found this alternative Bresenham function, but could not get it to work correctly with my code. Using cv2.line did not improve processing time.

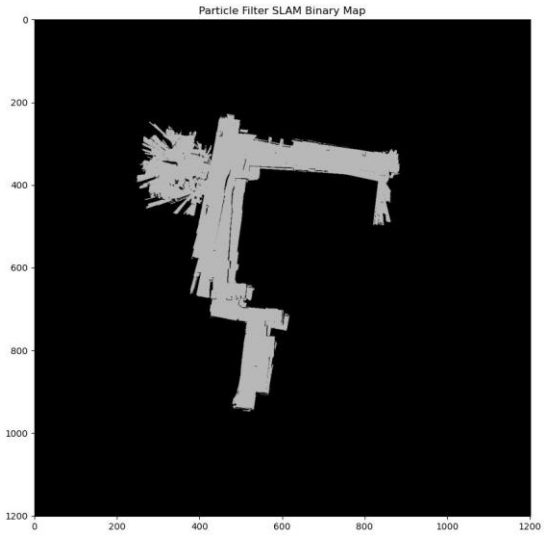
Dataset 20

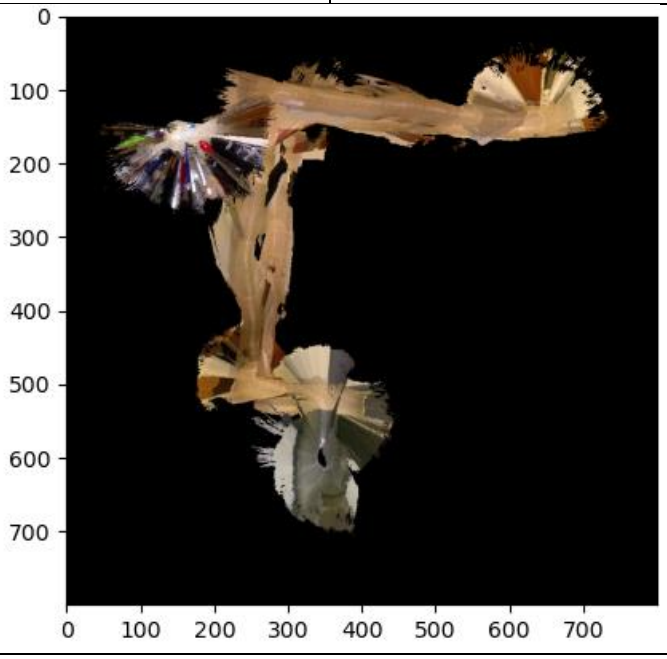
<table>
<tr><td colspan="2">Dead Reckoning</td></tr>
<tr><td> 
</td><td> 
</td></tr>
<tr><td> 
</td><td> 
</td></tr>
<tr><td colspan="2">Particle Filter SLAM</td></tr>
</table>


SLAM Trajectory


SLAM Theta

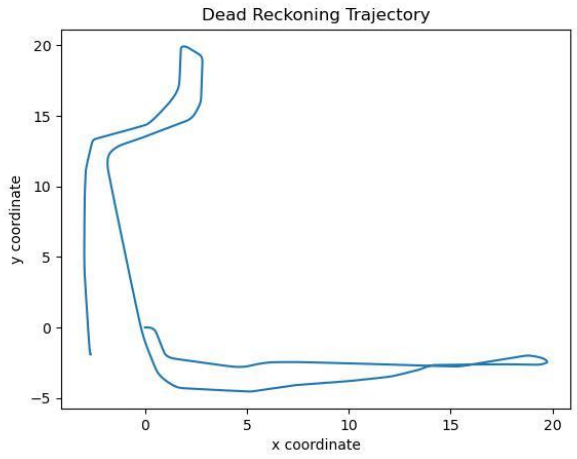

Particle Filter SLAM Occupancy Map

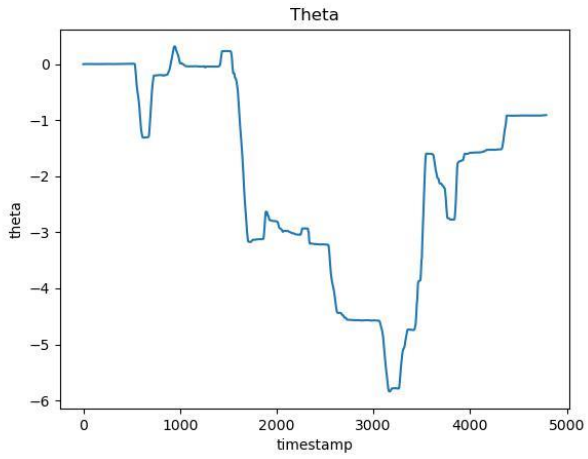

Particle Filter SLAM Binary Map

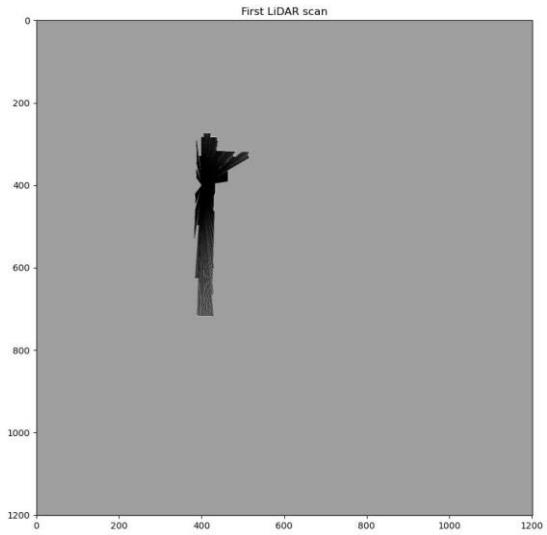


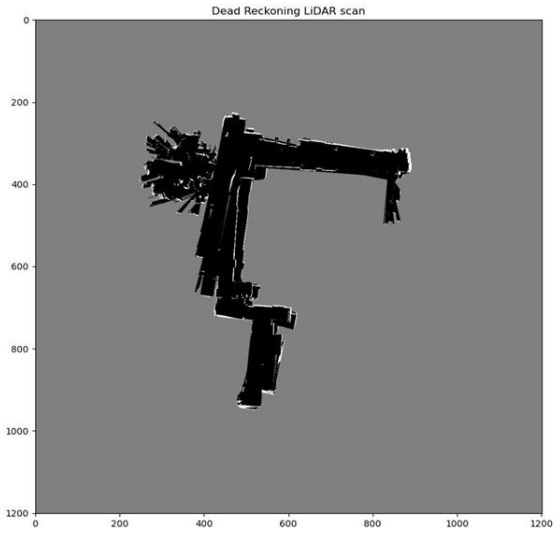
Dataset 21

Dead Reckoning

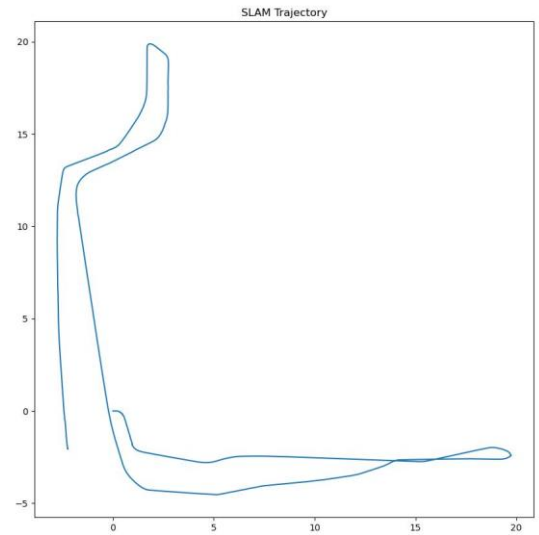


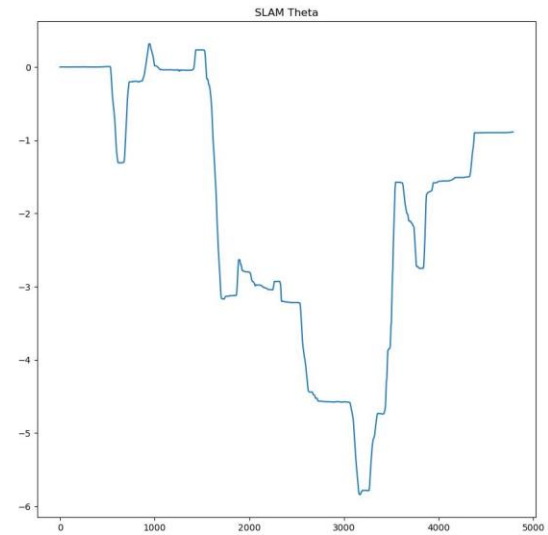


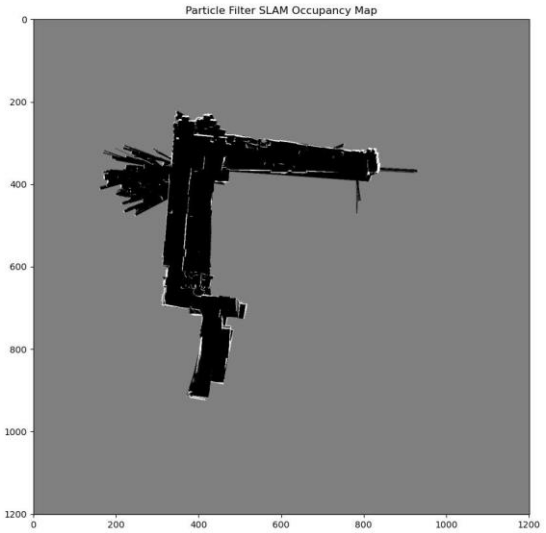


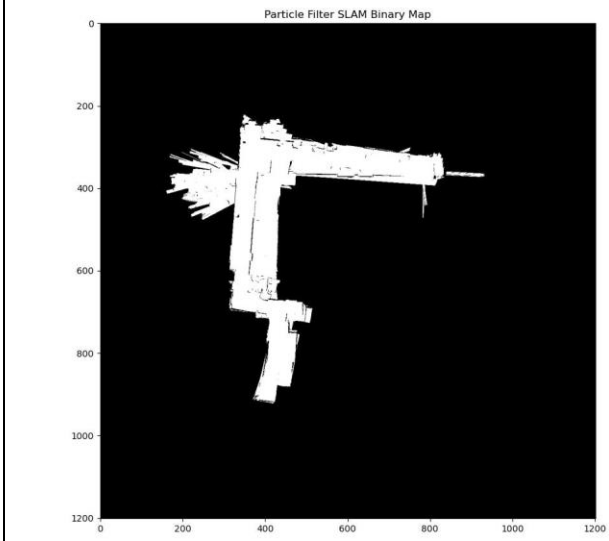


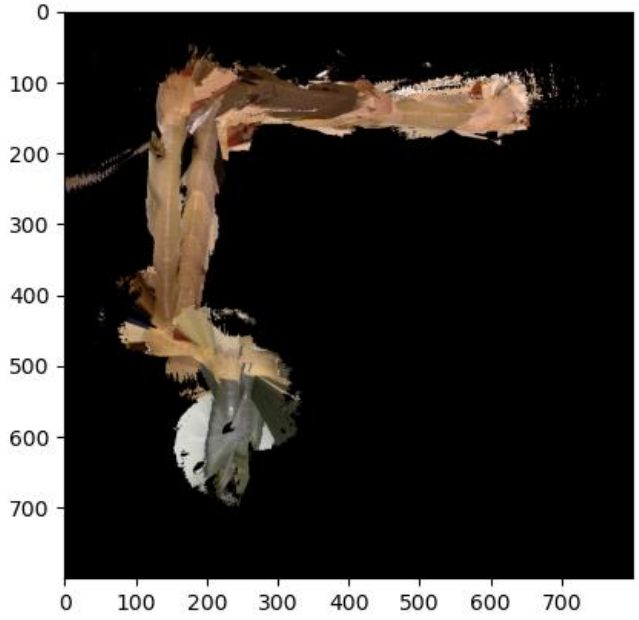
Particle Filter SLAM





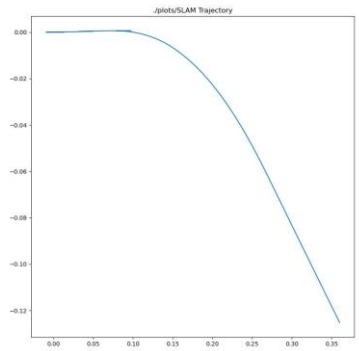
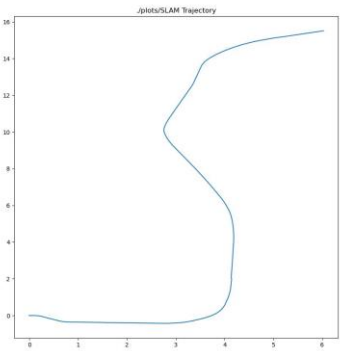
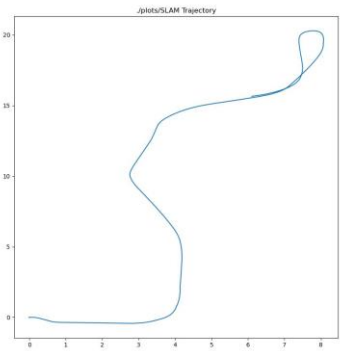


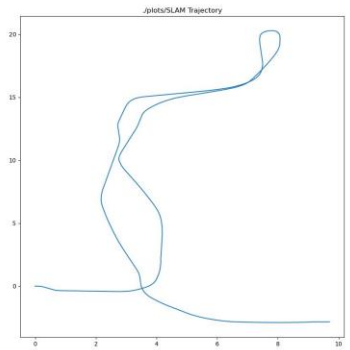




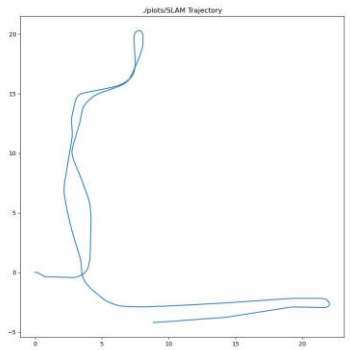
Plotting over time

Results are very clean and sharp initially for SLAM

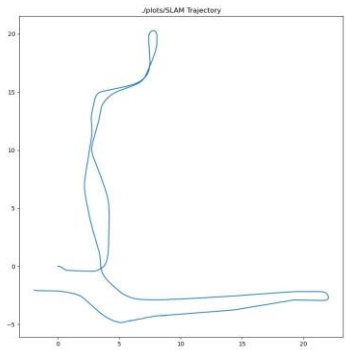
Dataset 20		
		
Timestamp 500	Timestamp 1500	Timestamp 2000



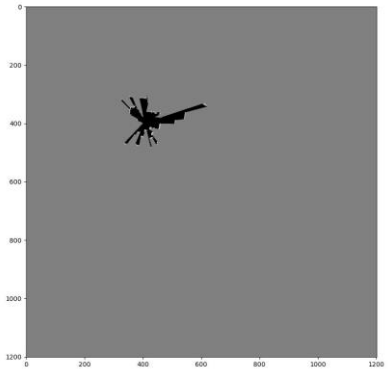
Timestamp 3000

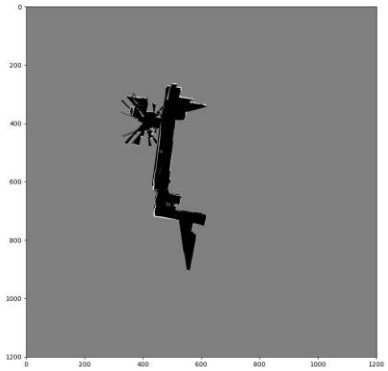


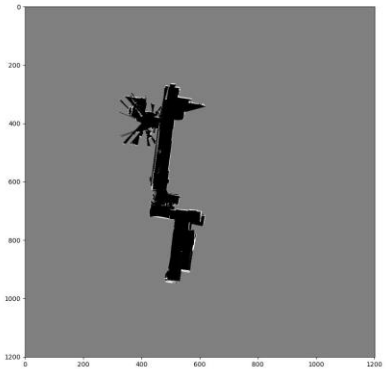
Timestamp 4000

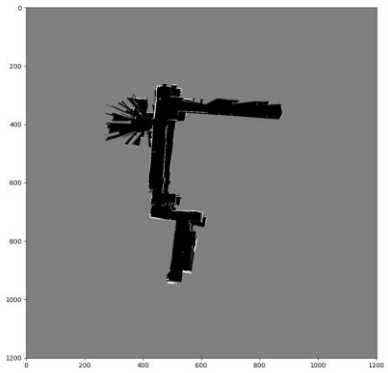


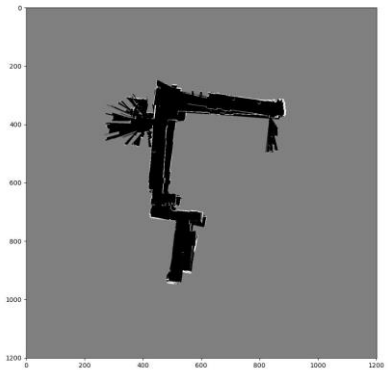
Timestamp 4500

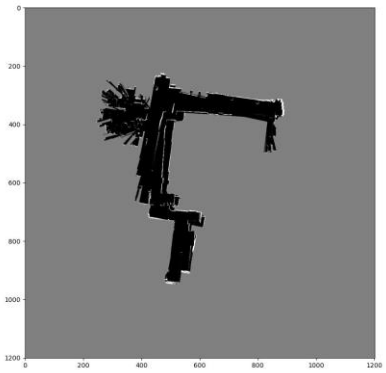


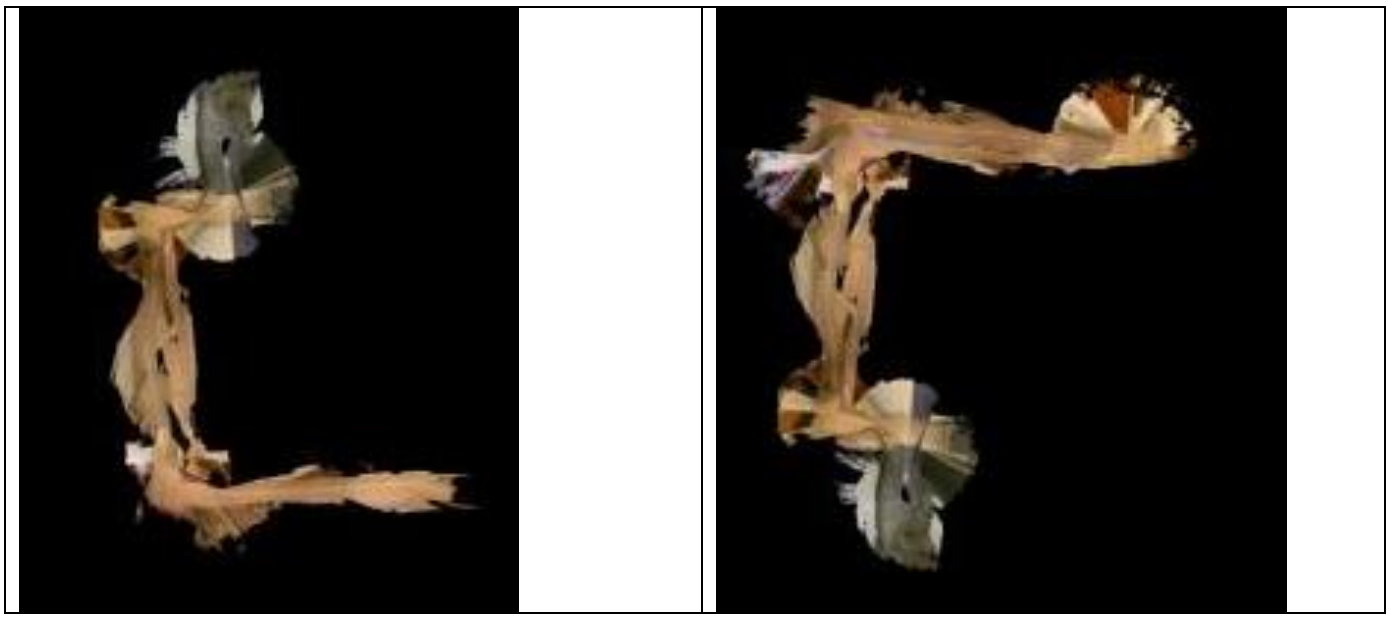












More images are attached with the code as I code not format them before the deadline.