Particle Filter SLAM and Texture Mapping

Saksham Jindal
Department of Electrical and Computer Engineering
University of California, San Diego

I. Introduction

SLAM, which stands for simultaneous localization and mapping, is a problem of constructing or updating a map of an unknown environment while simultaneously keeping track of an agent's location within it, without any external information available to the robot except for the observations and measurements made by the robot. Being able to create a map of the immediate surroundings of a robot, vehicle, or vessel, and locating the object within that map is important for self-driving cars, unmanned aerial vehicles, autonomous underwater vehicles, home robots etc.

There are many different approaches to SLAM which include the use of different sensors, observation models, motion models, and filtering techniques. One of the approaches, used in this paper, is implemented using Particle Filter, which utilizes physical sensors and a lidar, determine the position and orientation of a robot driving around and build a map of the surrounding. It is probabilistic inference technique that utilizes a group of particles to represent the probability distribution of a dynamic system. The Particle Filter based SLAM method involves two crucial stages – Prediction and Update. During the Prediction stage, the motion model is utilized to predict the movement of each particle. During the Update stage, the observation model is used updating the map based on the observations (measurements) by the sensors on the robot.

In this paper, we use data from 4 sensors – Encoder and Inertial Measurement Units (IMU) for pose update of the robot, LIDAR scan for building an occupancy grid and Kinect to create texture mapping of the ground plane. We discuss the problem formulation in Section II, Technical Approach in Section III and discussion of results in Section III.

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

II. Problem formulation

SLAM or "Simultaneous Localization and Mapping" is a state estimation problem to determine the environment map m and the robot pose or state x_t at each time step t, given the observations $z_0, \ldots, z_t$ and control inputs $u_0, \ldots, u_{t-1}$, in probabilistic form

$$p(m, x_t | z_{0:t}, u_{0:t-1}) \tag{1}$$

Motion Model: function f or equivalently probability density function p_f that describes the probability distribution over the possible future states of the system, given the current state x_t and the control input u_t. The output of the motion model is a probability distribution over the possible next states. Mathematically, the state x_{t+1} resulting from applying input u_t at state x_t is given by:

$$\begin{aligned} x_{t+1} &= f(x_t, u_t, w_t) \sim p_f(\cdot | x_t, u_t) \\ w_t &= \text{motion noise} \end{aligned} \tag{2}$$

The noise in the motion model takes into account the uncertainty in the prediction of the future state.

Observation Model: The robot senses the environment to get observation or measurement at each time step t and updates the map m of the environment . It is a function h or equivalently probability density function p_h that describes the probability density over the possible measurements z_t that can be obtained from the system, given the current state x_t:

$$\begin{aligned} z_t &= h(x_t, v_t) \ \sim p_h(\cdot | x_t) \\ v_t &= \text{observation noise} \end{aligned} \tag{3}$$

We have the following a sequence of following sensor measurements used for determining the most likely map and robot pose over time using the combination of motion and observation model

1) **Encoder**: gives us information about the linear and angular motion of the model. It is used to localize or determine the pose of the robot from the motion model.
2) **IMU**: gives us information about the angular velocity and linear acceleration of the robot. It is also used to determine the pose of the robot.
3) **Lidar**: gives up infromation of polar coordinates of the obstacle detected. It is used for as an input for the observation model and building the map of the environment.
4) **Kinect**: contains disparity and RGB maps and building a texture map of the environment.

A. Localization

Bayes filter [1], also known as recursive Bayesian estimation, is a general probabilistic approach for estimating an unknown probability density function (here, pdf of state x_t

of the robot), recursively over time using incoming measurements (observation model) and a mathematical process model (motion model) using markov assumptions and bayes rules. Bayes filter formulates the SLAM problem as a bayesian inference problem and uses Markov assumptions to induce a factorization of the joint probability density function of the states $x_{0:T}$, observations $z_{0:T}$, and inputs $u_{0:T-1}$

1) The state x_{t+1} only depends on the previous input u_t and state x_t and is independent of the history $x_{0:t-1}$, $z_{0:t-1}$, $u_{0:t-1}$
2) The observation z_t only depends on the state x_t

The Bayes Filter algorithm used for estimating probability distribution of state x_t of the robot consists of two main steps: the prediction step and the update step.

1) Prediction step: : In the prediction step, the algorithm uses the motion prior pdf $p_{t|t}(x)$ and motion model p_f to predict the state of the system at the next time step x_{t+1}, given the state at the current time step x_t and the control input u_t. The prediction step updates the prior probability distribution, which represents the estimate of the state at the current time step, to obtain the predicted probability distribution, which represents the estimate of the state at the next time step.

$$p_{t+1|t}(x) = \int p_f(x|s, u_t) p_{t|t}(s) ds$$

2) Update step: In the update step, the algorithm uses the observation model p_h to update the predicted probability distribution based on the sensor measurements obtained at the current time step. The update step computes the posterior probability distribution, which represents the updated estimate of the state given the current sensor measurements. This posterior distribution is used as the prior distribution in the next time step.

$$p_{t+1|t+1}(x) = \frac{p_h(z_{t+1}|x) p_{t+1|t}(x)}{\int p_h(z_{t+1}|s) p_{t+1|t}(s) ds}$$

B. Mapping

Mapping refers to the problem of generating a map of the environment m from noisy and uncertain sensor observations z given the state x of the robot. The goal of occupancy grid mapping is to estimate the posterior probability over time:

$$p(m|z_{0:t}, x_{0:t}) = \prod_i p(m_i|z_{0:t}, x_{0:t}) \tag{4}$$

In this equation, m represents the occupancy grid map, which is a discretized representation of the environment. The probability distribution $p(m|z_{0:t}, x_{0:t})$ represents our belief about the occupancy of each cell in the grid map at time steps 0 through t, given the sensor measurements and robot poses up to time t. The product over all cells in the occupancy grid map assumes that the cells are conditionally independent given the robot trajectory. The distribution of each cell m_i can be modeled individually using sensor observations and robot poses up to time t, denoted by $z_{0:t}$ and $x_{0:t}$, respectively.

C. Texture Mapping

Texture mapping refers to projecting 2D texture (3-channel RGB image) data from the Kinect to the cells of occupancy grid m of the environment.

In the next section, we will deep dive on the motion model, observation models and particle filter approach to solve the problem of SLAM.

III. Technical approach

We use Particle Filter approach to solve the SLAM problem discussed in the previous section. It uses a set of particles to represent the state of the system, each of which represents a possible state of the system. Each particle is typically represented by the state $\mu^{(k)}$ variable and weight that reflects its likelihood of being the true state of the system $\alpha^{(k)}$. A particle, represented by, ($\mu^{(k)}$, $\alpha^{(k)}$) is a hypothesis that the state x of the system is $\mu^{(k)}$ with probability $\alpha^{(k)}$.

We initialize N particles with equal weights 1/N. The particle filter uses particles with locations $\mu^{(k)}$ and weights $\alpha^{(k)}$ to represent the probability density functions $p_{t|t}$ and $p_{t+1|t}$:

$$p_{t|t}(x_t) = \sum_{k=1}^{N} \alpha_{t|t}[k] \delta(x_t - \mu_{t|t}[k]) \tag{5}$$

where, $\delta(\cdot)$ is the Dirac delta function. Here, the weight $\alpha^{(k)}$, assigned to each particle k as discrete probability mass function values evaluated at the location of the particle $\mu^{(k)}$. Using $\delta(\cdot)$ enables, us to represent the mixture as continuous space the probability density function.

A. Prediction

The prediction step uses the motion model to update the positions of each particle at a given time step t. The prediction step equation is given by

$$\begin{aligned} p_{t+1|t}(x) &= \int p_f(x|s, u_t) \sum_{k=1}^{N} \alpha_{t|t}^{(k)} \delta(x_t - \mu_{t|t}^{(k)}) ds \\ &= \sum_{k=1}^{N} \alpha_{t|t}^{(k)} p_f(x|\mu_{t|t}^{(k)}, u_t) \end{aligned} \tag{6}$$

We use differential drive kinematic model as the motion model $f(\cdot)$. Since, the object is moving in 2D space, The state of a system can be represented as $\mathbf{x} = (x, y, \theta) \in SE(2)$. The motion model can be expressed as:

$$\begin{aligned} x_{t+1} &= \begin{bmatrix} x_{t+1} \\ y_{t+1} \\ \theta_{t+1} \end{bmatrix} \\ &= x_t + \tau_t \begin{bmatrix} v_t \sin\left(\frac{\omega_t \tau_t}{2}\right) \cos(\theta_t + \frac{\omega_t \tau_t}{2}) \\ v_t \sin\left(\frac{\omega_t \tau_t}{2}\right) \sin(\theta_t + \frac{\omega_t \tau_t}{2}) \\ \omega_t \end{bmatrix} + w_t \end{aligned} \tag{7}$$

where v is the linear velocity and ω is the rotational velocity (yaw rate), τ_t is the time interval, and w_t is noise term adding to the state vector.

We use the encoder to get the encoder counts [FR, FL, RR, RL] corresponding to the four wheels. The right wheels travel a distance of $d_r = (FR + RR)/2 * 0.0022$, while the left wheels travel a distance of $d_l = (FL + RL)/2 * 0.0022$. We have approximation of speed of right wheels $v_r = d_r/\tau$ and left wheels $v_l = d_l/\tau$, where τ is the time interval between 2 consecutive encoder time stamps. The linear speed of the can be represented as $v_t = (v_r + v_l)/2$. We obtain the yaw rate ω from the IMU data. Since, the robot is moving in the 2D plane and do not consider the dynamics of the system, we can ignore the roll rate, pitch rate and linear acceleration values from the IMU data. We also notice that the data from encoder and IMU is not synchronised. During the prediction step, we find the yaw rate corresponding to the IMU timestamp corresponding to the encoder timestamp.

For each particle, we use the motion model $f(\cdot)$ to compute the probability density function $\mu^{(k)}_{t+1|t}$ under control input u_t with added Gaussian noise w_t and use this as an approximation of $p_f(x_{t+1}|\mu^{(k)}_{t|t}, u_t)$. Also, the prediction step changes only the particle positions but not their weights.

$$\mu^{(k)}_{t+1|t} = f(\mu_{t+1|t}, u_t, w_t) \tag{8}$$

$$\alpha^{(k)}_{t+1|t} = \alpha^{(k)}_{t|t} \tag{9}$$

We experiment with adding a Gaussian noise from the distributions $\mathcal{N}(0, 0.001)$, $\mathcal{N}(0, 0.01)$ and $\mathcal{N}(0, 0.05)$ to translation components and noise from the distribution from $\mathcal{N}(0, 0.001)$ for the orientation component of the state at each time step of prediction to account for noise in the encoder. Further, we also experiment with different noise levels by varying number of particles in the set $\{10, 100, 1000\}$.

B. Updation

The posterior in update step step is obtained by plugging the prediction pdf from (6) in the update step of Bayes filter and is given by

$$\begin{aligned}
p_{t+1|t+1}(x_{t+1}) &= \frac{p_h(z_{t+1}|x_{t+1}) \sum_{k=1}^{N} \alpha^{(k)}_{t+1|t} \delta(x_{t+1} - \mu^{(k)}_{t+1|t})}{\int p_h(z_{t+1}|s) \sum_{j=1}^{N} \alpha^{(j)}_{t|t} \delta(s - \mu^{(j)}_{t+1|t}) ds} \\
&= \sum_{j=1}^{N} \left[\frac{\alpha^{(k)}_{t+1|t} p_h(z_{t+1}|\mu^{(k)}_{t+1|t})}{\sum_{j=1}^{N} \alpha^{[j]}_{t+1|t} p_h \left(z_{t+1} \mid \mu^{[j]}_{t+1|t} \right)} \right] \\
&\cdot \delta(x - \mu^{(k)}_{t+1|t}) \\
&= \alpha_{t+1|t+1} \delta(x - \mu^{(k)}_{t+1|t})
\end{aligned} \tag{10}$$

where,

$$\begin{aligned}
\alpha_{t+1|t+1} &= \frac{\alpha^{(k)}_{t+1|t} p_h(z_{t+1}|\mu^{(k)}_{t+1|t})}{\sum_{j=1}^{N} \alpha^{[j]}_{t+1|t} p_h \left(z_{t+1} \mid \mu^{[j]}_{t+1|t} \right)} \\
&= \frac{p_h(z_{t+1}|\mu^{(k)}_{t+1|t})}{\sum_{j=1}^{N} p_h \left(z_{t+1} \mid \mu^{[j]}_{t+1|t} \right)} \\
&= \frac{p_h(z_{t+1}|\mu^{(k)}_{t+1|t})}{\eta} && (11) \\
&\propto p_h(z_{t+1}|\mu^{(k)}_{t+1|t}) && (12)
\end{aligned}$$

We use the lidar ranges values which give us the depth r and θ of the obstacle in the LIDAR frame. For LIDAR measurements z_i, after obtaining the valid range r and angle θ, we can calculate the coordinates (x_l, y_l) of each obstacle encountered by each LIDAR beam using the following equations:

$$\begin{aligned}
x_l &= r\cos(\theta) \\
y_l &= r\sin(\theta)
\end{aligned}$$

These equations give us the location of each LIDAR measurement in the 2D plane, relative to the LIDAR sensor itself. We first transform these LIDAR measurements from the lidar frame to the body frame and, further, we transform these points in the particle's body frame into our world frame using the pose estimate of the particle at time $t + 1$. We constraint our grid to be of size 50 m x 50 m and exclude any points encountered outside the grid. The resulting world coordinates (x_w, y_w) are then fed into the correlation function to find the most likely position of the robot in the occupancy grid. We assume that observation model is proportional to correlation function and compute the updated weights using correlation value of each particle based on how well it matches the LIDAR scan.

$$p_h(z_{t+1}|\mu^k_{t+1|t}, m_t) \propto \exp(\text{corr}(m^k_{t+1}, m)) \tag{13}$$

All k weights $\alpha^{(k)}$ are normalized to 1 by softmax over correlation values.

$$\alpha^{(k)}_{t+1|t+1} = Softmax(\text{corr}(m^k_{t+1}, m)) \tag{14}$$

We select particle k with the maximum weight $\alpha^{(k)}_{t+1|t+1}$ as state x_{t+1}

$$x_{t+1} = \mu^{(k)}_{t+1|t+1}$$

To overcome particle depletion, we use stratified resampling which adds new particles at locations with high weights and reduces the particles at locations with low weights. Specifically, given a particle set $(\mu^{(k)}_{t|t}, \alpha^{(k)}_{t|t}))$, resampling is applied if the effective number of particles is less than N/10

$$Neff := \frac{1}{\sum_{k=1}^{N} (\alpha^{(k)}_{t|t})^2} < N/10$$

is less than a threshold. It ensures that particles are distributed in regions with high probability density, resulting in a more accurate estimate of the true state.

C. Mapping

The environment is represented as $m \in \mathbb{R}^{\mathrm{M*N}}$ and maintain a probability density function $p(m|z_{0:t}, x_{0:t})$ over time to model the occupancy probabilities γ_t.

We model the map cells m_i as independent Bernoulli random variables and occupancy probabilities as $\gamma_{i,t}$ using Bayes rule and update the map by accumulating log odd ratio of $\gamma_{i,t}$.

$$m_i = \begin{cases} +1 \text{ (Occupied) with prob. } \gamma_{i,t} := p(m_i = 1|z_{0:t}, x_{0:t}) \\ -1 \text{ (Free) with prob. } 1 - \gamma_{i,t} \end{cases} \tag{15}$$

Using the above equation, we have

$$\begin{aligned} \gamma_{i,t} &= p(m_i = 1|z_{0:t}, x_{0:t}) \\ &= \frac{1}{\eta_t} p_h(z_t|m_i = 1, x_t) p(m_i = 1|z_{0:t-1}, x_{0:t-1}) \gamma_{i,t-1} \\ &= \frac{1}{\eta_t} p_h(z_t|m_i = 1, x_t) p(m_i = 0|z_{0:t-1}, x_{0:t-1})(1 - \gamma_{i,t-1}), \end{aligned}$$

where η_t is a normalization constant and $h(z_t|m_i, x_t)$ is the measurement model. The odds of m_i updated over time is

$$\begin{aligned} o(m_i|z_{0:t}, x_{0:t}) &= \frac{\gamma_{i,t}}{1 - \gamma_{i,t}} \\ &= \frac{p(h(z_t|m_i = 1, x_t))}{p(h(z_t|m_i = 0, x_t))} \frac{\gamma_{i,t-1}}{1 - \gamma_{i,t-1}} \\ &= g_h(z_t|m_i, x_t) o(m_i|z_{0:t-1}, x_{0:t-1}) \end{aligned} \tag{16}$$

We assume a confidence of 80% in the LIDAR measurement and take $g_h(1|z_i, x_t) = 4$ and $g_h(-1|z_i, x_t) = 0.25$. The update rule for log odds is given by taking logarithm of the above equation and factorizing the rule under the independence assumption. Taking the log for both sides of the equation, we get

$$\begin{aligned} \lambda(m_i|z_{0:t}, x_{0:t}) &= \log o(m_i|z_{0:t}, x_{0:t}) \\ &= \lambda(m_i|z_{0:t-1}, x_{0:t-1}) + \log g(h(z_t|m_i, x_t)) \\ &= \lambda_{i,t-1} + \log g_h(z_t|m_i, x_t) \end{aligned} \tag{17}$$

which is the update rule for log-odds. For each observed cell i, we decrease the log-odds by log4 if it was observed free or increase the log-odds by log4 if the cell was observed occupied. We clip the $\lambda_{i,t-1}$ between λ_{min} and λ_{max} to avoid increasing too much confidence or preventing too much decrease in log odds.

D. Texture Mapping

We have not covered texture mapping in this paper due to time constraints. However, we have attempted to work out the texture mapping and based on our understanding, the procedure for the same is a follows. We can use disparity and RGB images from the RGBD dataset to project the pixel corrdinates from the 2D images to 3D coordinates in the camera's optical frame (X_O, Y_O, Z_O). The depth camera is located at (0.18, 0.005, 0.36) m with respect to the robot center and has orientation with roll 0 rad, pitch 0.36 rad, and yaw 0.021 rad. We can use this information to generate transformation ${}_BT_C$ for mapping the points from camera frame to body and, further, projecting them into world frame using the pose estimate from the previous step. An intermediate step, would also involve mapping points from optical frame to camera frame, While we obtain optical coordinates, we can maintain the RBG color of each of (X_O, Y_O, Z_O) as C_O. Finally, we select points whose Z coordinate is below a threshold of 0.5 which we obtain from LIDAR z coordinate.

$$X_W = {}_WT_{BB}T_{CC}T_O X_O$$
$$C_W = C_O$$

IV. Results

We test our approach on 2 datasets - datasets "20" and "21" - and explore the effect of adding more particles and different Gaussian noise in the motion model.

A. Dead Reckoning Trajectories

The dead reckoning trajectories are obtained by simulating the motion of the robot with N=1 particle and no added noise. However, to gauge the effect of adding noise in motion model, we experiment by adding noise from the distributions $\mathcal{N}(0, 0.001)$, $\mathcal{N}(0, 0.005)$, $\mathcal{N}(0, 0.01)$, $\mathcal{N}(0, 0.05)$, $\mathcal{N}(0, 0.1)$, $\mathcal{N}(0, 0.5)$ to the translation component (x_t, y_t) while adding a noise from the distribution $\mathcal{N}(0, 0.001)$ to the orientation component θ_t of the state vector.

We observe that there is local drift and global divergence in the dead reckoning trajectories as the noise level increases. Particularly, we observed that there is a remarkable deviation in trajectories when noise in sampled from $\mathcal{N}(0, 0.1)$ or higher standard deviation. The reason for this instability is that the noise in the motion model introduces uncertainty in the robot's actual position and orientation, which accumulates over time as the robot moves. In comparison to trajectories of lower noise level, we observed only a local drift in trajectory when simulated with added gaussian noise from $\mathcal{N}(0, 0.05)$. However, we do not observe a global diverge and trajectory does not deviate with the path followed by earlier noise levels.

We have included the simulation results on dataset 20 and 21 in the Appendix A and B of this paper.

B. Simultaneous localization and mapping

While increasing the noise level can help to reduce bias in the robot's estimate of its position and orientation, it also increases the variance in the estimate, making it more uncertain. We believe it could be a good idea to sample noise from $\mathcal{N}(0, 0.1)$ or $\mathcal{N}(0, 0.5)$ for the simultaneous localization and mapping. Our hypothesis was that increasing the number of particles should reduce the variance in estimate of the final trajectory. However, adding particles came with additional bottleneck of linear increase in computational cost. We test our approach by adding noise levels from 3 distributions $\mathcal{N}(0, 0.001)$ (which is equivalent to adding no noise), $\mathcal{N}(0, 0.01)$ and $\mathcal{N}(0, 0.05)$ and test out our approach by increasing particles on log scale. We test our hypothesis by testing our algorithm by using N=100 and N=1000 particles for dataset 20 and N=10 and N=100 for the dataset 21.

We observed adding noise may decreases bias in the trajectoeries and occupancy map and increasing variance in the trajectory as we increase the noise levels. Particularly, the robot achieves a loop closure when we added noise from $\mathcal{N}(0, 0.01)$ and $\mathcal{N}(0, 0.05)$. However, we also observed that the adding more noise increases variance in the localization and mapping measurements. This can be observed by paying close attention to local drift experienced by the trajectories and the smoothening of occupany map (the map becomes more ”fat” and specks inside and around the map reduce).

We observed a deviation in trajectories as we increase the number particles from N=100 to N=1000 and also observe that some parts of the map (for N=1000) rotate relative to their counterpart with lesser number of particles. While increasing particles from N=10 and N=100 (in dataset 21) seem to have a better effect on convergence of the state estimate and occupancy map, the same can not be said about the effect of increasing particles from N=100 to N=1000 in the dataset 20. Overall, we believe that a balance between noise level and number of particles needs to be struck to achieve the best localization and mapping performance.

We have included the simulation results on dataset 20 and 21 in the Appendix C and D of this paper.

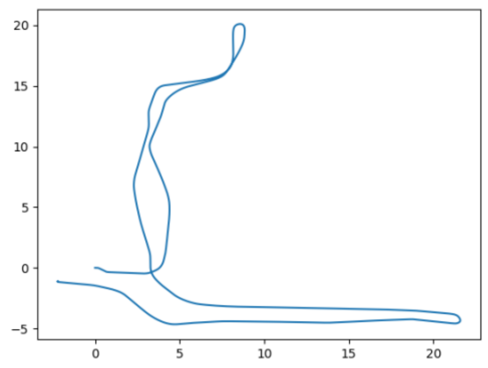
References

[1] S. Thrun, W. Burgard, and D. Fox., “Probabilistic robotics” MIT Press, 2005

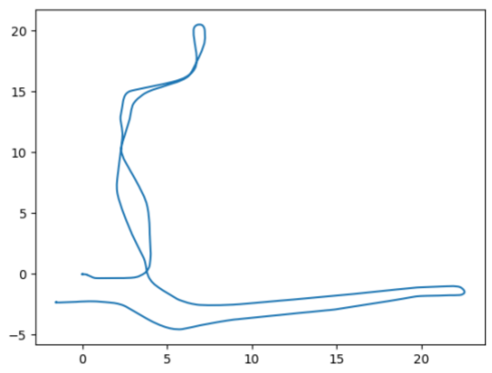
[2] N. Atanasov, ”Lecture 8: Particle Filter”, ECE276A: Sensing Estimation in Robotics, 2022

Appendix

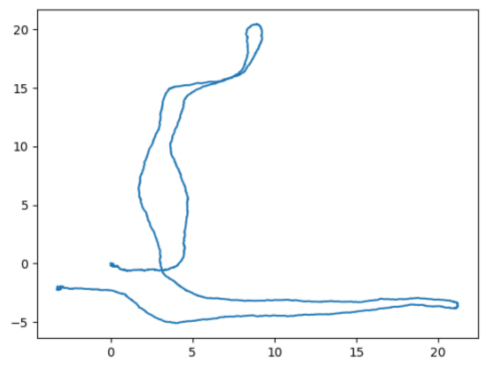
A. Dead Reckoning trajectories on dataset 20 with added gaussian noise



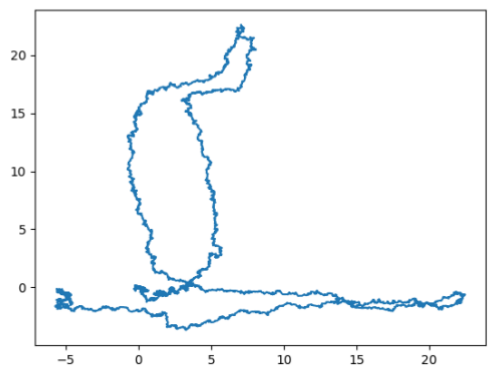
Gaussian Noise with mean 0 and standard deviation 0



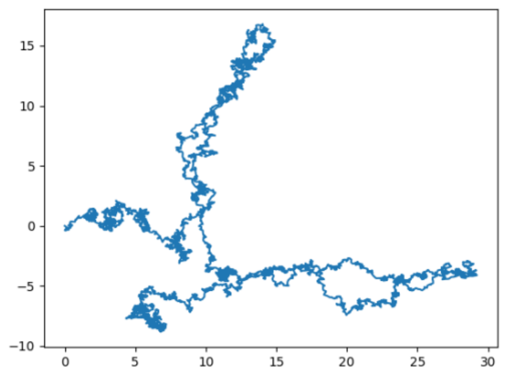
Gaussian Noise with mean 0 and standard deviation 0.001



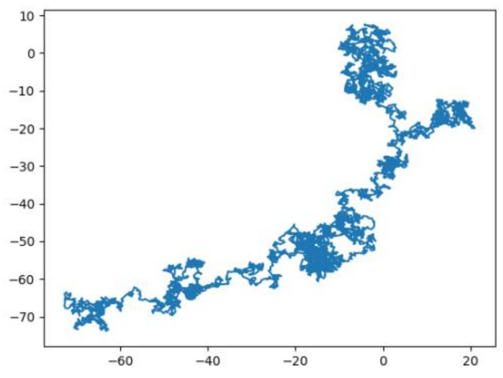
Gaussian Noise with mean 0 and standard deviation 0.01



Gaussian Noise with mean 0 and standard deviation 0.05

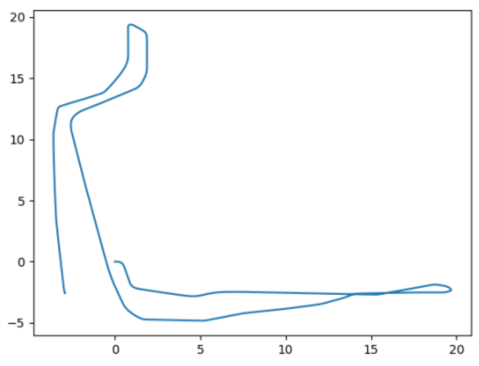


Gaussian Noise with mean 0 and standard deviation 0.1

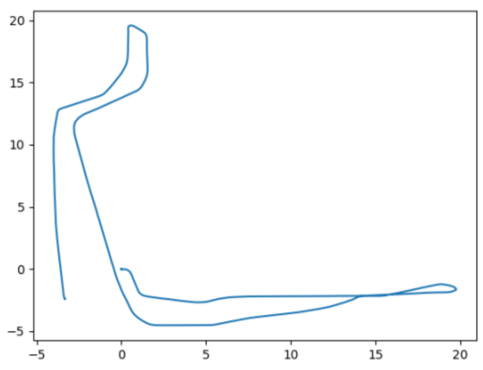


Gaussian Noise with mean 0 and standard deviation 0.5

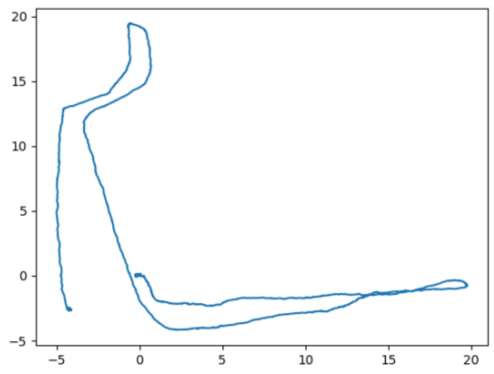
B. Dead Reckoning trajectories on dataset 21 with different gaussian noise



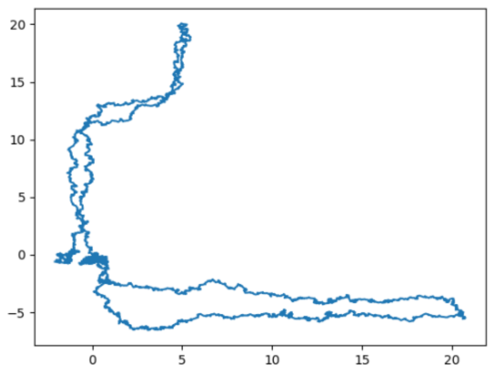
Gaussian Noise with mean 0 and standard deviation 0



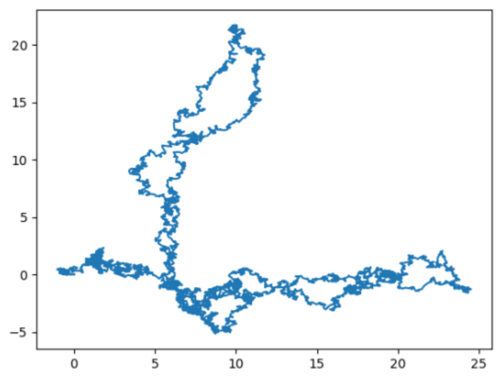
Gaussian Noise with mean 0 and standard deviation 0.001



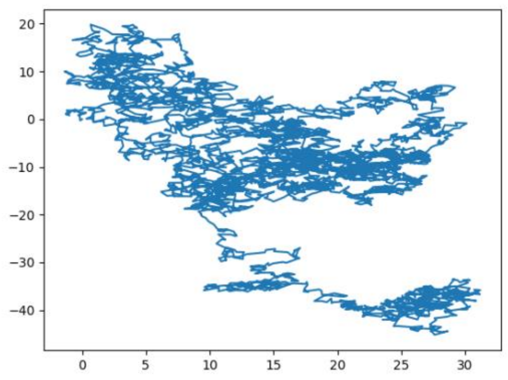
Gaussian Noise with mean 0 and standard deviation 0.01



Gaussian Noise with mean 0 and standard deviation 0.05

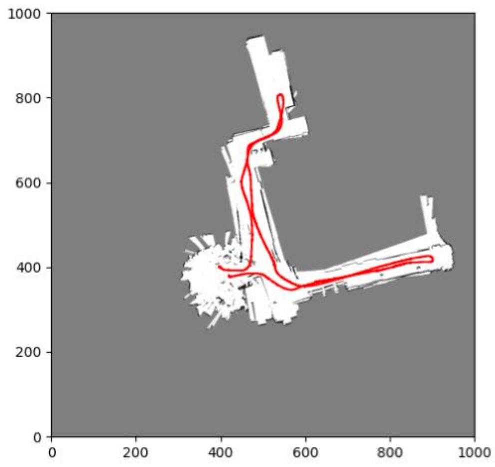


Gaussian Noise with mean 0 and standard deviation 0.1

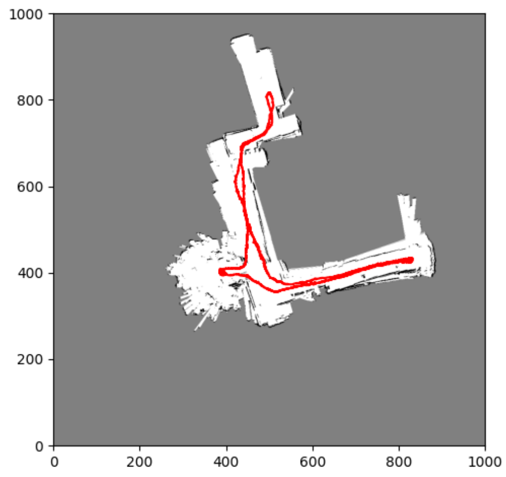


Gaussian Noise with mean 0 and standard deviation 0.5

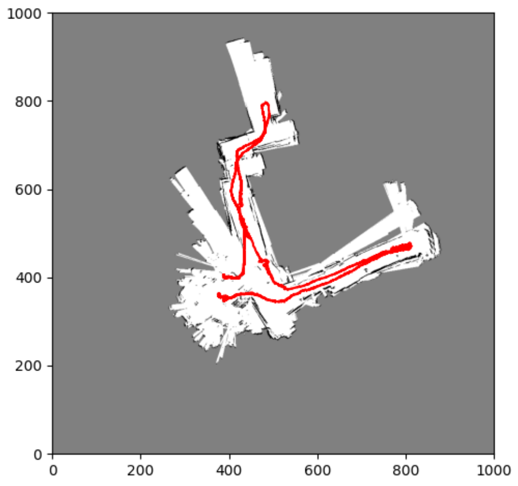
C. Occupancy Map of the environment for Dataset 20



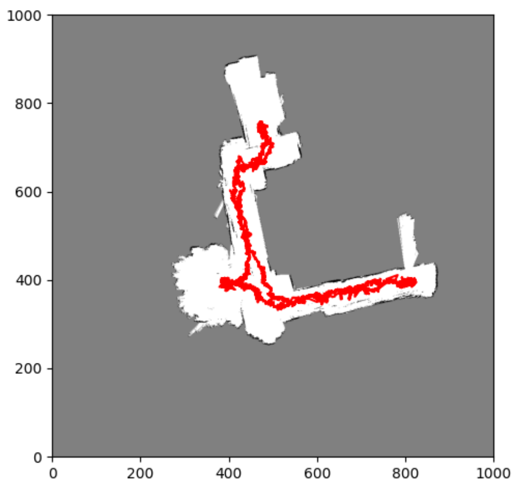
Final map with 100 particles and noise from N(0, 0.001)



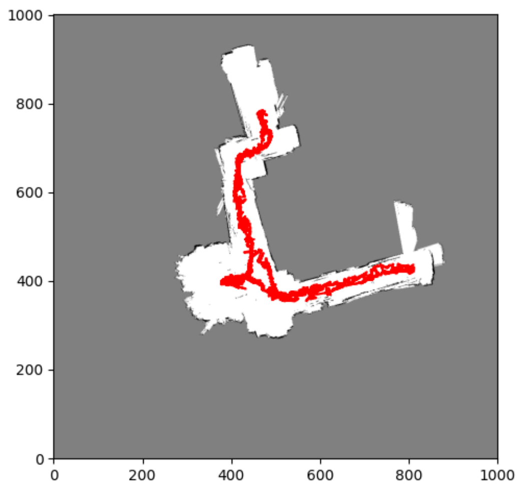
Final map with 100 particles and noise from N(0, 0.01)



Final map with 1000 particles and noise from N(0, 0.01)

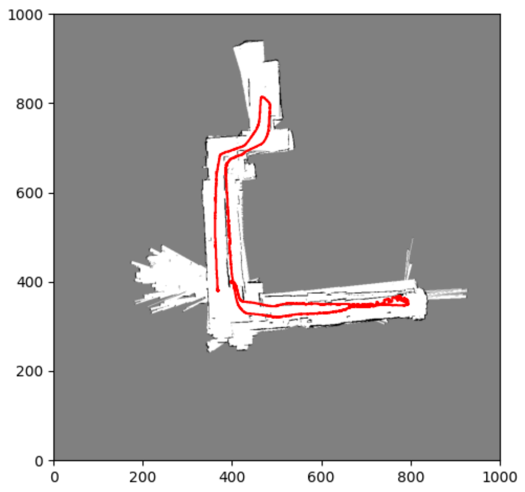


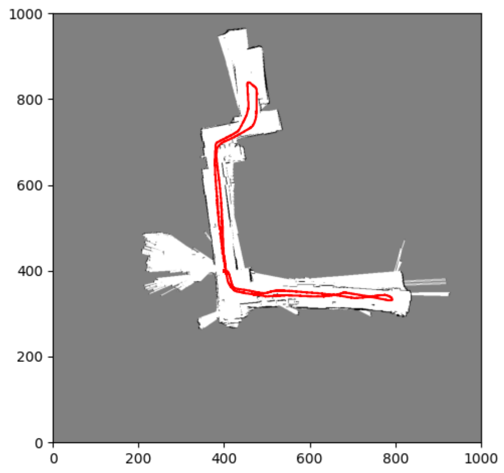
Final map with 100 particles and noise from N(0, 0.05)

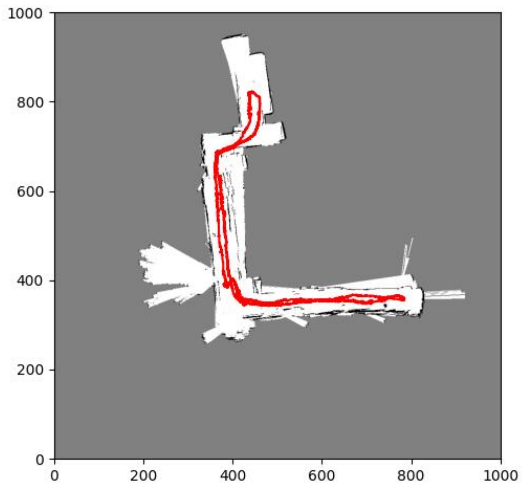


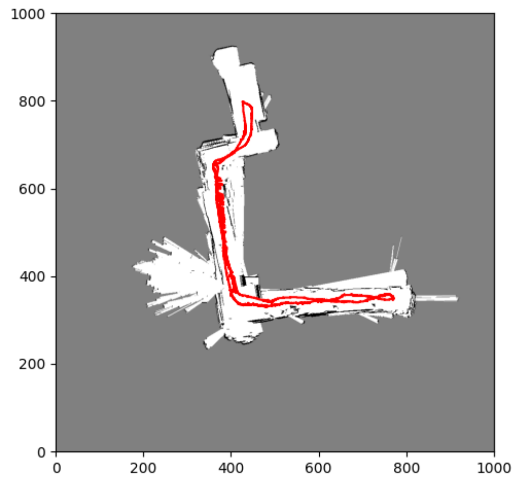
Final map with 1000 particles and noise from N(0, 0.05)

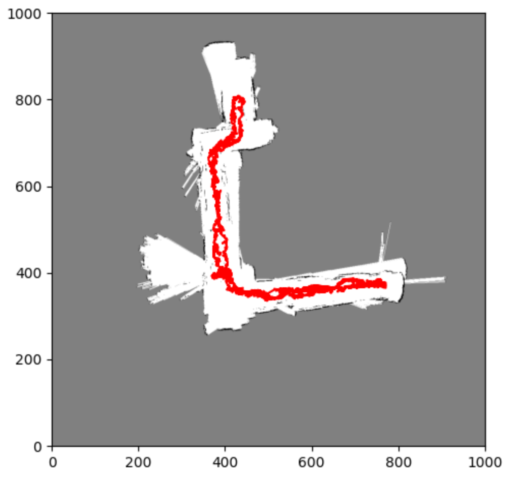
D. Occupancy Map of the environment for Dataset 21

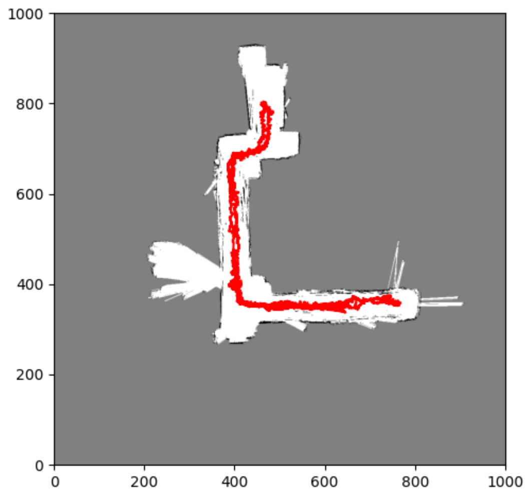

Final map with 10 particles and noise from N(0, 0.001)


Final map with 100 particles and noise from N(0, 0.001)


Final map with 10 particles and noise from N(0, 0.01)


Final map with 100 particles and noise from N(0, 0.01)


Final map with 100 particles and noise from N(0, 0.05)


Final map with 1000 particles and noise from N(0, 0.05)