Particle Filter SLAM

Rishabh Bhattacharya
Department of Mechanical & Aerospace Engineering
University of California, San Diego
ribhattacharya@ucsd.edu

***Abstract*—Simultaneous localization and mapping (SLAM) is a necessity in order to control any robot in an unknown environment. A deterministic approach fails in this regard, due to inherent sensor errors. Also, with no prior information about the environment, mapping and estimation have to be carried out hand-in-hand, which can stress our computational resources. This project aims to present an approach in SLAM, using particle filter.**

***Index Terms*—particle filter, bayesian filter, localization, mapping, SLAM, occupancy grid, odometry, stereo camera, texture mapping**

I. Introduction

Any robot in a complex dynamic system has to be able to localize itself w.r.t the environment, in order to ensure operability and safety guarantees. In most cases, prior information about the environment is not known. So the task now involves estimating the position of the robot and simultaneously mapping the environment. This needs to be done in real time, which adds further computational challenges to this problem.

This can only be achieved by providing some sort of sensory information about the environment to the robot, and make some predictions about the robot state. Light detection and ranging (Lidar) sensors can be used to provide obstacle data, while Fiber Optic Gyroscope (FOG) sensor and magnetic encoders can provide odometry related data for the robot (eg. change in roll/pitch/yaw angles, total rotations of the wheel, etc).

In this project, we will use 2D Lidar data to map the environment. We will predict the state of the robot using encoders (linear velocity) and FOG data (angular velocity). We will simultaneously map the environment using the Lidar observations, while also estimating the pose of the robot that correlates well with the observed environment.

A crucial aspect to note here is the fact that we avoid any sort of deterministic approaches, since any and all sensors have inherent errors. Also, any control input given to the robot (eg. move ahead by 1 m) will not be implemented perfectly since the distance travelled depends on the wheel diameters (which can vary according to tire pressure, temperature, etc). This is the reason we approach this problem, with a probabilistic approach.

Bayes filter is a probabilistic inference approach for estimating a state $\mathbf{x_t}$ given a set of controls $\mathbf{u}_{0:t}$ and observations $\mathbf{z}_{0:t}$. Particle filter is just a discrete formulation of the same problem. For the purposes of SLAM in this project, we will use a particle filter implementation.

After having successfully obtained the robot trajectory, we will attempt a texture mapping of the ground onto the path obtained. This shall be approached by forming a stereo-camera model of our stereo sensors, and projecting the RGB values from the images onto our map.

The report has been organized as follows; we begin with the problem formulation (II), which talks about how to setup the SLAM problem (II-A) and texture mapping (II-B). We then elaborate on the technical approach (III) which elaborates our solution approach for each segment of the problem, which are mapping (III-A), prediction (III-B), update (III-C) and ending with texture mapping (III-D). Finally we present our results and observations (IV), which includes discussions on the visuals obtained as well.

II. Problem Formulation

The entire problem consists of two parts, SLAM and texture mapping. We use a Bayesian inference based particle filter approach for SLAM, which involves iterative prediction and update steps, while using a stereo camera model for the texture mapping.

A. SLAM

We want to generate a map of the environment, given lidar observations ($\mathbf{z_t}$) and encoder/fog data ($\mathbf{u_t}$).

Particle: N particles are initialized with equal weights $\alpha^k_{0|0} = \frac{1}{N} \forall\ k \in \{1, 2...N\}$. At any time t, we hypothesize that the value of $\mathbf{x}$ is one of the assumptions μ^k with probability α^k.

Prior:

$$\mathbf{x}_t \mid \mathbf{z}_{0:t}, \mathbf{u}_{0:t-1} \sim p_{t|t}\left(\mathbf{x}_t\right) := \sum_{k=1}^{N} \alpha^{(k)}_{t|t} \delta\left(\mathbf{x}_t - \mu^{(k)}_{t|t}\right)$$

Prediction: given $p_{t|t}, \mathbf{u}_t$, compute $p_{t+1|t}$

$\mu^{(k)}_{t+1|t} \sim p_f\left(\cdot \mid \mu^{(k)}_{t|t}, u_t\right)$ and $\alpha^{(k)}_{t+1|t} = \alpha^{(k)}_{t|t}$ so that:

$$p_{t+1|t}(\mathbf{x}) \approx \sum_{k=1}^{N} \alpha^{(k)}_{t+1|t} \delta\left(\mathbf{x} - \mu^{(k)}_{t+1|t}\right)$$

We obtain $p_{t|t}$ from the prior, while we obtain $\mathbf{u_t}$ (linear and angular velocity for the differential drive model) from the sensory data (encoder and FOG sensor).

Update: given $p_{t+1|t}, \mathbf{z}_{t+1}$, compute $p_{t+1|t+1}$

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

The observations are made available using 2D-lidar scan data. We also need to update the occupancy grid map based on the particle weights (III-A). The prediction and update steps are iterated to generate the desired map after we exhaust our observations/control inputs.

B. Texture Mapping

Given a sequence of images from the left and right stereo cameras, we need to project the *ground* RGB values from the image to the occupancy grid map, using the following stereo camera model.

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

where,

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = {}_oR_r R^\top (\mathbf{m} - \mathbf{p}); \quad d = u_L - u_R = \frac{1}{z} fs_u b$$

III. Technical Approach

SLAM is basically a chicken-and-egg-problem, wherein we try to i) build a map $\mathbf{m}$ of an environment given the robot trajectory $\mathbf{x}_{0:T}$ and ii) estimate the robot trajectory $\mathbf{x}_{0:T}$ given a map $\mathbf{m}$ of the environment. It can be modeled as a parameter estimation problem for $\mathbf{x}_{0:T}$ and $\mathbf{m}$ given a dataset of the robot inputs $\mathbf{u}_{0:T}$ and observations $\mathbf{z}_{0:T}$. For this project, we will use Bayesian inference to maintain a posterior pdf for the parameters given the data, implemented through a particle filter. The particle filter is a histogram filter which allows its grid centers to move around and adaptively concentrate in areas of the state space that are more likely to contain the true state.

A. Mapping

For mapping we use the lidar scans to determine the obstacles and free path cells using the *bresenham2D* function provided. We keep the obstacles based on their distance to our vehicle ($0.1 <$ endpoint < 80 in m) to get rid of erroneous/out of limit readings, as the lidar range is 80 m. We then transform the lidar scan into our world frame using the transformation matrix provided,

$$l_w = T_{wp} \times T_{ps} \times l_s \tag{1}$$

where l_w are the lidar coordinates in world frame, T_{wp} is the transformation matrix of the particle w.r.t world, T_{ps} is the transformation of the lidar sensor w.r.t particle/vehicle and l_s are the lidar cartesian coordinates in lidar sensor frame. We then again filter out points where height ≤ 1.5m, since those are the obstacles that affect thew mobility of our robot, not the ones that are too high.

Having the lidar information in the world frame, we initialize two simultaneous maps i) a *bitmap* (occupancy grid map) 2D array of $\{0, 1, 2\}^{3001 \times 3001}$ and ii) a real valued *logmap* $\mathbb{R}^{3000m \times 3000m}$, both containing zeros initially. The bitmap array stores the independent Bernoulli variables $\{0, 1, 2\}$ as

$$\begin{cases} 0 \rightarrow \text{unexplored (default value at the beginning)} \\ 1 \rightarrow \text{obstacle} \\ 2 \rightarrow \text{free space} \end{cases}$$

The *bitmap* is obtained from the *logmap* using,

$$bitmap = \begin{cases} 0 \rightarrow \textit{logmap} = 0 \\ 1 \rightarrow \textit{logmap} > 0 \\ 2 \rightarrow \textit{logmap} < 0 \end{cases}$$

The reason this is done is because the *probability map* (probability of free space/obstacle for each cell) can be obtained from the sigmoid function,

$$p = \frac{1}{1 + e^{-logmap}}$$

which validates the *bitmap* - *logmap* relation.
We use the *logmap* to update the log-odds for each lidar scan, by increasing the log-odds for obstacles and decreasing it for the empty cells.

$$\lambda_{i,t+1} = \lambda_{i,t} \pm \Delta\lambda_{i,t} \tag{2}$$

where

$$\Delta\lambda_{i,t} = \log \frac{p\left(m_i = 1 \mid \mathbf{z}_t, \mathbf{x}_t\right)}{p\left(m_i = -1 \mid \mathbf{z}_t, \mathbf{x}_t\right)} = \log(\text{sensor confidence})$$

B. Prediction

1) We initialize N particles with equal weights $\alpha_0^k = \frac{1}{N}$ and corresponding pose $\mu_0^k = [x_0^k, y_0^k, \theta_0^k]^T$.
2) The inputs to our vehicle can be obtained from the encoder values (v - linear velocity) and fog data (ω - angular velocity). These inputs are provided to our particle with a gaussian noise $\phi(v; 0, \sigma_v^2)$ and $\phi(\omega; 0, \sigma_\omega^2)$, to mimic the real life error for these sensors. Each particle gets a different input and thus a different motion, based on σ_v and σ_ω. (Note: encoders are usually noisy compared to fog, which are fairly accurate).
3) The input data obtained is used to find the particle states using the discrete time differential drive model,

$$\mathbf{x}_{t+1} = \begin{bmatrix} x_{t+1} \\ y_{t+1} \\ \theta_{t+1} \end{bmatrix} = \mathbf{x}_t + \tau \begin{bmatrix} v_t \cos(\theta_t) \\ v_t \sin(\theta_t) \\ \omega_t \end{bmatrix} \tag{3}$$

We do this for each particle in our framework.

Notice that only the particle states change during this step, while α^k remains the same.

C. Update

1) We convert the next lidar scan (l_s) to the world frame using Eqn. 1 for each particle pose in the system. The new T_{wp} (particle w.r.t world) is given by

$$T_{wp}^k = \begin{bmatrix} cos(\theta_{t+1}^k) & -sin(\theta_{t+1^k}) & x_{t+1}^k \\ sin(\theta_{t+1}^k) & cos(\theta_{t+1}^k) & y_{t+1}^k \\ 0 & 0 & 1 \end{bmatrix}$$

where $k \in \{1, 2...N\}$

2) We determine a correlation between each lidar scan and the current map using the *mapCorrelation* function provided.

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

where $\mathbf{y}, \mathbf{m}$ are the discretized lidar scan and occupancy grid respectively and k is the particle index. The correlation is large if the scan $\mathbf{y}$ and map $\mathbf{m}$ agree with each other.

3) We update the weights of the particles according to their corresponding correlation score.

$$\alpha_k = \frac{\alpha_k \times \text{corr}_k}{\sum_{i=1}^N \alpha_i \times \text{corr}_i}$$

Then we pick the particle with the largest weight and update the map-log-odds (Eqn. 2) by projecting the laser scan to this particle pose.

$$k = \arg\max \alpha$$

$$l_w = T_{wp}^k \times T_{ps} \times l_s$$

4) During these iterations, our particles might become more/less confident about the robot location, which might lead to a large skew in the weight values (α). In such cases, a resampling is required.
If $N_{\text{eff}} := \frac{1}{\sum_{k=1}^N \left(\alpha_{t|t}^{(k)}\right)^2} \leq N_{\text{threshold}}$, resample particles $\{\mu_{t|t}^k, \alpha_{t|t}^k\}$ via importance resampling as follows,
   a) Draw $k \in \{1, 2...N\}$ independently with replacement and probability α^k
   b) Add the sample μ^k with $\alpha^k = \frac{1}{N}$ to the new particle

Note that in the update step, the particle locations remain same, but the weights change according to the correlation found. This is in contrast to the prediction step.

D. Texture Mapping

We are provided with two stereo images from the left and right cameras mounted on the robot. We can use the *compute_stereo* function provided, which calculates the disparity between the images.

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

MAP['xmin']	-1500 m
MAP['ymin']	-1500 m
MAP['xmax']	1500 m
MAP['Ymax']	1500 m
MAP['res']	1 m
lidar_confidence	4 (80%)
z_thresh	1.5 m
resample_thresh	60% of N_particles
N_particles	30
ratio (prediction - update ratio)	20
v_var	1E-1
w_var	1E-3

TABLE I: Input parameters

where,

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = {}_oR_r R^\top (\mathbf{m} - \mathbf{p}); \quad d = u_L - u_R = \frac{1}{z} f s_u b$$

- u_L, v_L, d are the pixel row, pixel column and calculated disparity.
- ${}_oR_r, R = R_{wp} R_{pc}$ are the optical rotation matrix and stereo camera orientation w.r.t world frame respectively.
- b is the baseline, while $\mathbf{p}$ is the camera position w.r.t world.

The real world coordinate of the pixel is thus given by $\mathbf{m}$, which we want to calculate. Since we want to map the ground, we can filter coordinates with $z \leq 0.5$ m, such that we get ground coordinates. The corresponding RGB values are then superimposed onto our occupancy grid cells, by converting them from pixels coordinates $\rightarrow$ world frame coordinates $\rightarrow$ occupancy grid cell coordinates.

IV. Results

The input parameters have been stated in Table I, for which the corresponding figures are 1, 2, 3 & 4. We change the N_particles and prediction ratio, to perform a comparative analysis in figures 5, 6 & 7. We then perform a study on the effect of changing noises in figures 8, 9 & 10. v (linear velocity is of the order of 4-6 m/s, thus an error of 1E-1 seems justified. Similarly, ω is of the order of 0.005-0.01 rad/s, thus an error of 1E-3 seems realistic.)

V. Conclusions

We were able to successfully implement particle filter to localize our robot and map the environment with high accuracy. For this project, we implemented prediction, update and mapping of the robot and its environment using sensor data (provided beforehand). The next steps in the project could involve implementation of Kalman filter, which is a more robust implementation.

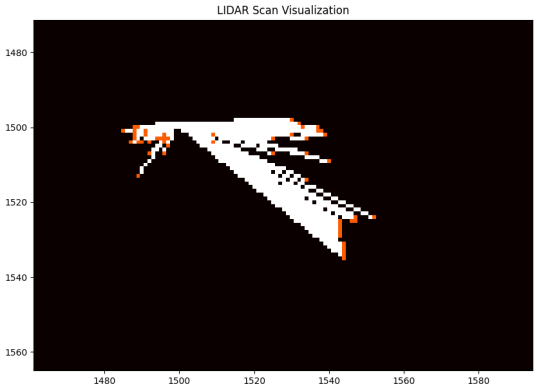

Fig. 1: LIDAR scan for the initial time step. Notice the red pixels (end points) and the white pixels (empty cells). Black region is the unexplored area. LIDAR range is 80 m, which is evident from the plot.

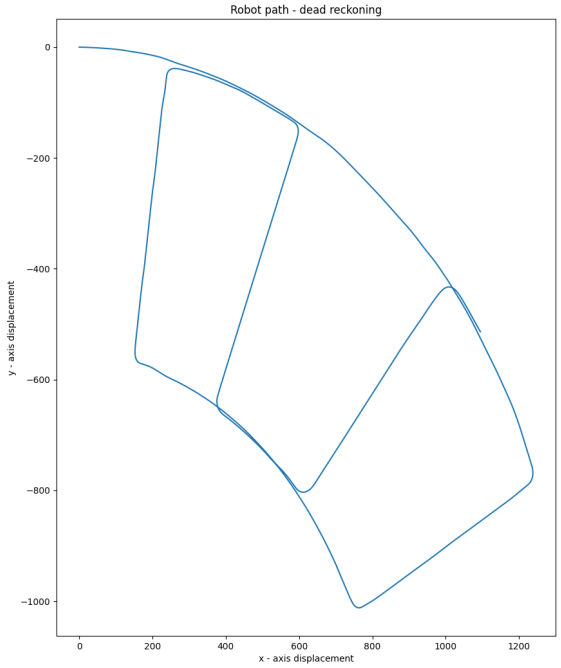

Fig. 2: Robot dead reckoning plot for for the input conditions in Table I

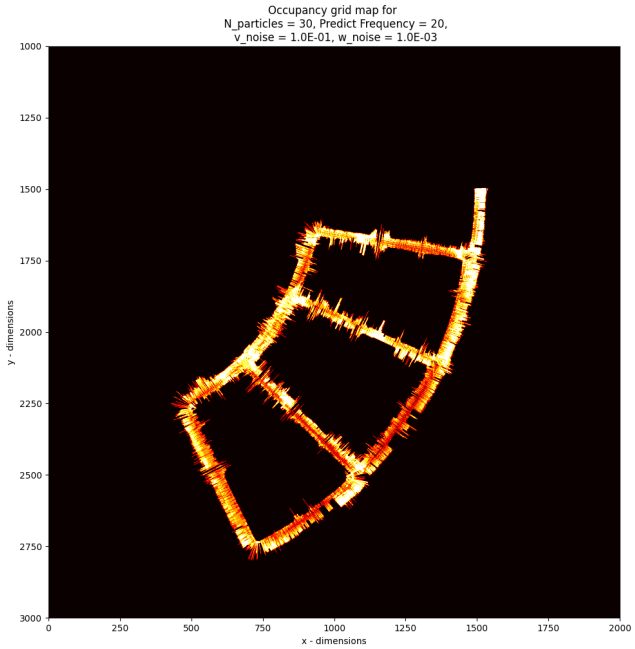

Fig. 3: Occupancy grid map for the input conditions in Table I

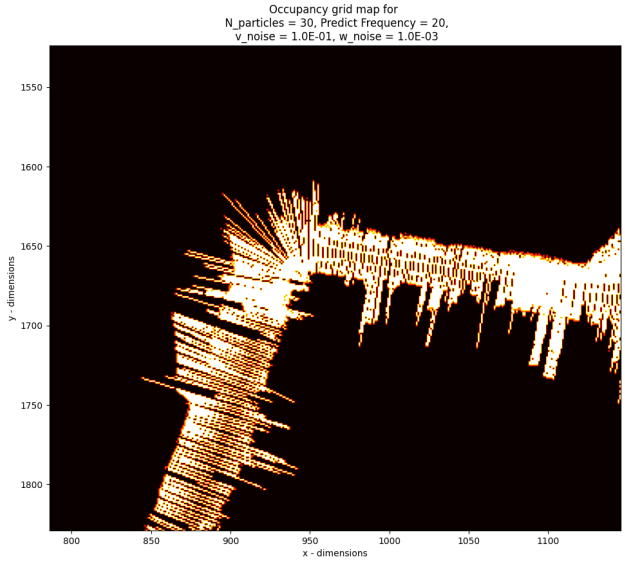

Fig. 4: Zoomed in figure for Fig. 3. Notice that the walls/roadside is red in colour, validating that those are obstacles. The white road/path is empty, while the black region is unexplored.

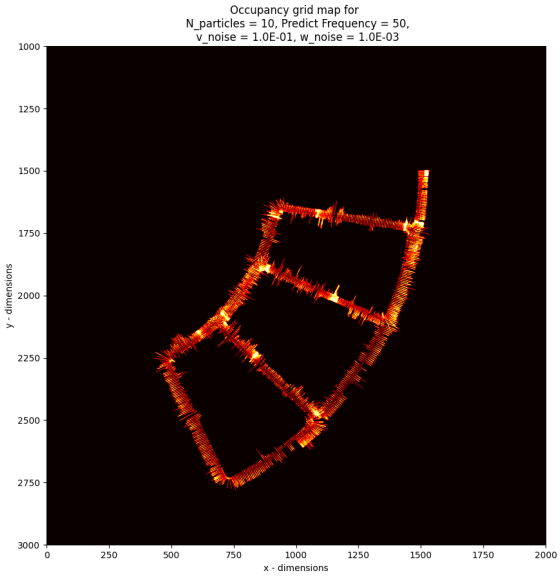

Fig. 5: Reducing the N_particles and prediction ratio w.r.t Fig. 3, we find that we have very sparse lidar scans, which is visible in the lower density of free space

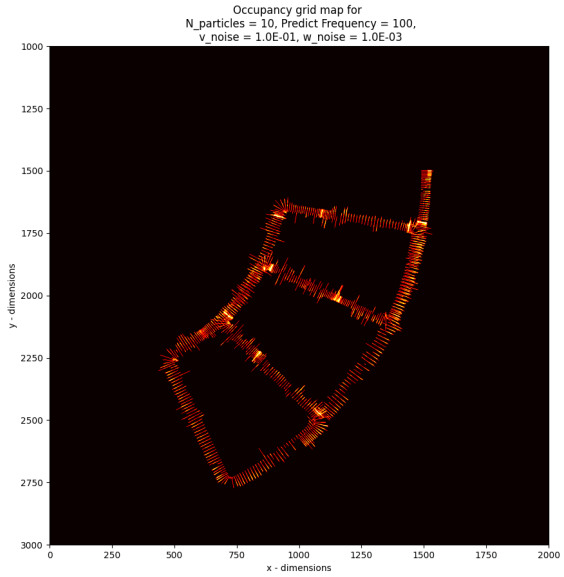

Fig. 6: Changing the N_particles and prediction ratio, we find that we have very sparse lidar scans, which is visible in the lower density of free space w.r.t Figs 5 and 3

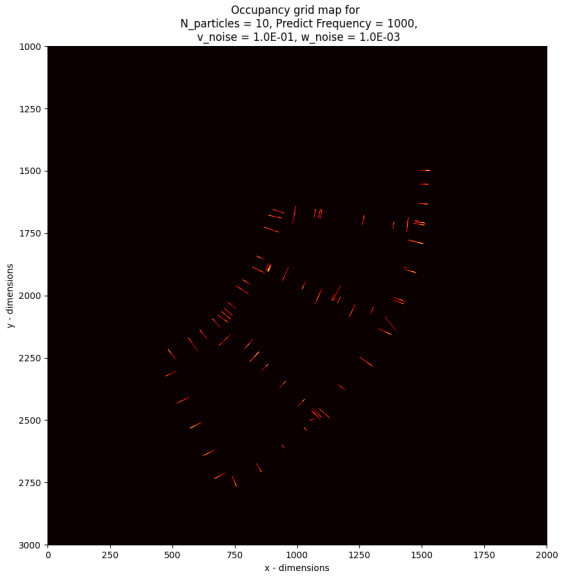

Fig. 7: With a prediction frequency of 1000, we see that the free space is barely mapped, and this is not a suitable scenario for any SLAM problem

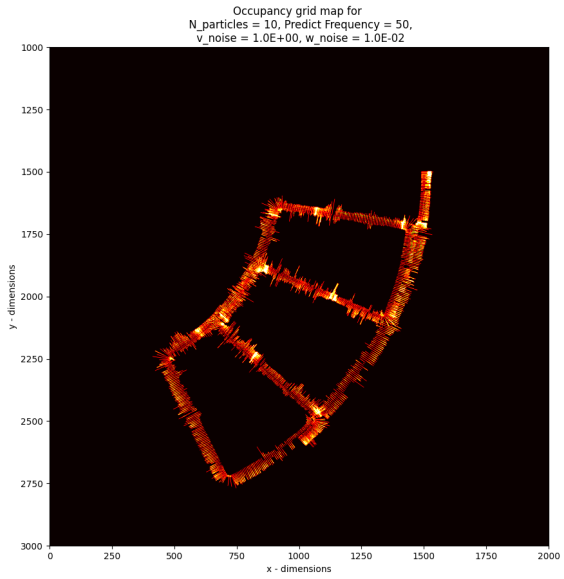

Fig. 8: Now we increase the noises by a factor of 10. We do not observe much deviation from the path

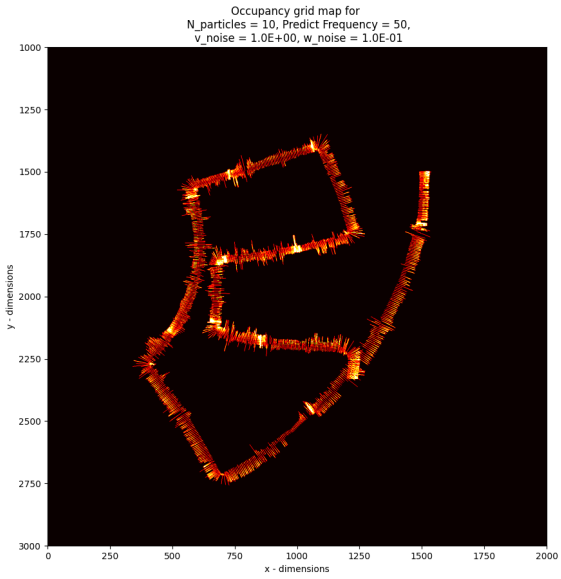

Fig. 9: Increasing the noise in ω by another order of magnitude leads to some interesting results.

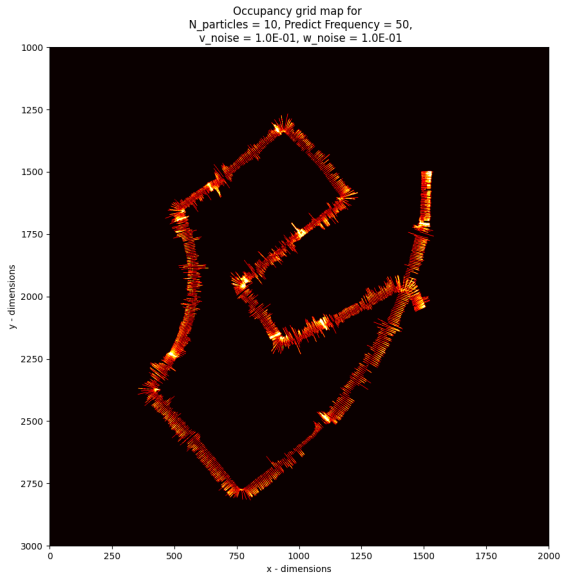

Fig. 10: Resetting the noise to the original value, we have the noise in ω increased my 2 orders of magnitudes.