Particle Filter SLAM

Vaibhav Bishi
Department of Electrical and Computer Engineering
University of California, San Diego
La Jolla, USA
vbishi@ucsd.edu

***Abstract*—This project presents an approach for solving the Simultaneous Localization and Mapping (SLAM) problem for a moving vehicle. In this project, we have used a particle filter with a differential-drive motion model and scan-grid correlation observation model for implementing SLAM. The results show accurate estimation of the vehicle trajectory and the map around it.**

Keywords—SLAM, particle filter, openCV

I. Introduction

In this modern age of science and technology, the presence of autonomous vehicles and robots is becoming ever more profound by the day. With increasing complexities in task, we are inclined to develop better models and techniques for these autonomous agents to deal with different situations. One such problem which is associated with autonomous navigation is the Simultaneous Localization and Mapping (SLAM) problem. SLAM is a technique that utilizes the raw data obtained via sensors from the outside world directly to perform mapping and localization simultaneously. In this problem, an autonomous agent has to be able to identify its own state in an environment while simultaneously having to build the map around it. This is much like a chicken and an egg problem, giving it is a form of parameter estimation problem in some sense as we have to deal with probabilities.

In this project, we solve the SLAM problem for a vehicle using sensor data from 2D LiDAR, odometry and stereo camera measurements. We use differential-drive model and scan grid map correlation to estimate the vehicle's location and build an occupancy grid map. We implement the particle filter which is a special case of a more general Bayes filter to achieve our objective.

II. Problem Formulation

A. Problem Statement

We are given multiple sets of sensor data from a moving vehicle. We particularly have readings from 3 different kind of sensors mounted on the vehicle: wheel encoder, fiber optic gyroscope (FOG) and front scanning 2D LiDAR. Our objective is to estimate the localized state of the vehicle in the map while simultaneously generating said map over time.

B. Dataset

We are given the number of ticks for the left and right wheels from the encoder along with the timestamps. The wheel diameters are around $0.62\ m$ with a resolution of 4096. The FOG data gives the change in roll, pitch and yaw of the vehicle in radians along with the timestamps. We are given LiDAR scans with angular range of $190°$ from $-5°$ to $185°$. The angular resolution is $0.666°$ with maximum range of $80\ m$.

C. SLAM Problem

The localization problem deals with predicting the position of an agent in a given map, while the mapping problem deals with generating a map of the agent's surroundings as it progresses. The SLAM problem is a culmination of both localization and mapping with a limited understanding of both. Thus, given a sequence of control inputs $u_{0:T-1}$ and observations z_T, we are required to generate a sequence of the agent's state**s** $x_{0:T}$ and the map m surrounding it. We can relate these parameters by exploiting the decomposition of the joint probability distribution with Markov independence assumption as:

$$p(x_{0:T}, m, z_{0:T}, u_{0:T-1}) = p_0(x_0, m)\prod_{t=0}^{T} p_h(z_t|x_t, m)\prod_{t=1}^{T} p_f(x_t|x_{t-1}, u_{t-1})\prod_{t=0}^{T-1} p(u_t|x_t)$$

D. Particle Filter

Our task is to solve the SLAM problem using the particle filter, which is a special kind of Bayes filter. The Bayes filter is a probabilistic inference technique for estimating the state of a dynamical system. Here, we will have to keep track of $p_{t|t}(x_t)$ and $p_{t+1|t}(x_{t+1})$ using a prediction step to incorporate the control inputs and an update step to incorporate the measurements.

Prediction step:

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

Update step:

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

The particle filter that we will implement is a special kind of Bayes filter that does not assume anything about the world.

E. Occupancy Mapping

An occupancy grid mapping is to be used to represent the environment around the agent by dividing it into cells. It helps keep track of the state of each map cell by assigning values as 1 for occupied cells and -1 for free cells. The cell values are probability mass functions defined as:

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

F. Motion Model

A differential-drive motion model is to be implemented to determine the movement of the vehicle over time. The Euler discretization gives the equation as:

$$x_{t+1} = \begin{bmatrix} x_{t+1} \\ y_{t+1} \\ \theta_{t+1} \end{bmatrix} = f(x_t, u_t)$$

Here, we need to use the encoder and FOG data to model the motion trajectory.

G. Observation Model

An observation model is to be used to determine the relation between the vehicle's observation z_t, its state x_t and map m_t.

$$z_t = h(x_t, m_t, v_t)$$

Here, we need to use the LiDAR data to model the observations.

III. Technical Approach

A. Data Cleaning

In order to make sense of the data and make it directly usable for the SLAM implementation, we firstly need to process the data from the raw sensor readings.

Firstly, for the wheel encoders, we are given the tick counts for both the wheels and we need to convert these values into distances travelled by the vehicle across different timestamps. We determine the distance Δx between consecutive timestamps Δt as:

$$\Delta x = \frac{\pi \times diameter \times \Delta ticks}{resolution}$$

Secondly, for the FOG, we only need the change in yaw angle as we can assume that the vehicle is moving in 2D. However, this data is sampled roughly 10x faster than the encoder and LiDAR. So, we compute the net change in angle in FOG readings during a particular time interval corresponding to two consecutive encoder readings by summing up all the 10 delta yaw angles in that interval.

Lastly, for the LiDAR readings, we are given the scans that correspond to distances where the vehicle observes an obstacle. We see that these are sampled almost at the same rate as the encoder although with around 200 readings less. Thus, we do not require much pre-processing here. However, we would still need to synchronise the readings which we shall deal with during map update.

B. Dead Reckoning

Dead reckoning is an important step to assess if we are on the right track. Dead reckoning simply means the trajectory that can be obtained considering a single body (particle) using the differential-drive model dynamic equations. We can get an estimate of how our vehicle's trajectory is supposed to look like all while we are using just the encoder and FOG data.

The aforementioned motion model give the equation:

$$\Delta X = \begin{bmatrix} x_{mean}\cos(\theta + \Delta\theta) \\ x_{mean}\sin(\theta + \Delta\theta) \\ (\theta + \Delta\theta) \end{bmatrix}$$

Here, x_{mean} is the mean distance travelled by the car using data from both left and right wheels, θ and $\Delta\theta$ are the cumulative angle and the change in angle respectively.

Using Euler discretization, we can compute the next state X_{t+1} as:

$$X_{t+1} = X_t + \Delta X$$

This gives us the coordinates of the vehicle in the world frame.

C. Mapping

Once we have the deterministic single particle trajectory, we now have to create a map of the surroundings of the vehicle. In order to achieve this, we fist initialize a $1400\text{x}1400$ occupancy grid map with a resolution of $1m$.

From the LiDAR scans at each timestamp, we firstly remove the points that are extremely close to the vehicle ($< 2m$) or very far from the vehicle ($> 75m$). Once we convert these readings (scan end-points) from polar to Cartesian coordinates, we apply pose transformations to represent these points from LiDAR frame to vehicle frame to finally in the world frame. It is easy to note that the start points are the vehicle coordinates (in world frame) obtained from the motion model earlier. Then we convert these coordinates into grid cell units and determine all the cells between the LiDAR start and end-points using the bresenham2D ray tracing algorithm. The cells that are between the start and end points are considered free and the last cell detected is assumed blocked. Finally, we update the log-odds map cells by increasing the cell probability mass function by $\log 4$ for free cells and decreasing the same by $2 \times \log 4$ for occupied cells. The reason for including a 2 factor for occupied cells is because we want to give more preference to avoiding obstacles from a safety perspective. Additionally, we clip the range for cell values to prevent over-confidence.

$$\lambda_{i,t+1} = \lambda_{i,t} + \log 4$$

$$-10 \log 4 \leq \lambda_{i,t} \geq 10 \log 4$$

We take $\log 4$ to assert that we choose to believe that our sensors have 80% accurate. Once we are finished with the mapping, we can plot the log-odds occupancy grid map. We can also recover the map probability mass function from the log-odds map using the logistic sigmoid function to generate a binary map of the world.

$$\gamma_{i,t} = \frac{e^{\lambda_{i,t}}}{1 + e^{\lambda_{i,t}}}$$

D. Particle Filter SLAM: Prediction

Now that we have implemented the trajectory and mapping for a single particle without any probabilistic approach, we can start implementing particle SLAM. Here, we are going to take a probabilistic approach to solving this localization and mapping problem by adding multiple particles with certain noise that can represent the best possible trajectory.

We take a set of $n = 5$ particles initially, all of which have their own 3D states $p = [x, y, \theta]$ just like the single vehicle before. They also have their own assigned weights α which are initialized as $1/n$.

Using the $\Delta\theta$ obtained from the FOG readings and the differential-drive model, we predict the updated pose of each particle as:

$$\mu_{t+1} = \mu_t + \begin{bmatrix} \Delta x \\ \Delta y \\ \Delta\theta \end{bmatrix}$$

On top of that, we add Gaussian noise to each of the particle state in all degrees using zero mean and a variance of $\max(\Delta x/10)$, $\max(\Delta y/10)$ and $\max(\Delta\theta/10)$ similar to some form of noisy envelope around the particle trajectory.

E. *Particle Filter SLAM: Updation*

Once we have completed the prediction step, we update the pose and weight of each particle using map correlation. For this, we convert the LiDAR scan points to world frame coordinates just as before and create a 9x9 grid around each particle's pose. Then, we find the map cells that correspond to this pose and compute map correlation for each particle. After that, we find the particle that gives the best correlation of its projected scan with the prior map we have so far. We get the particle weights using the softmax function and the particle state with the highest weight becomes the new vehicle position which we consider for our optimum trajectory. Once we have this best particle state, we again update the log-odds map.

In order to deal with possible depletion of particles, we can implement particle resampling when our effective number of particles become less than our assigned threshold which is set at 40% of the particle count. We have implemented Stratified resampling in particular for this case.

Color Classification

Given the labelled training images, we implement the Gaussian Discriminant Analysis model discussed above to train our model and get the required parameters θ, μ and Σ. Using these parameters, we classify the images based on the Bayes decision rule as shown in the last equation. We implemented our code for classifying blue pixels on the validation set and tested it on unknown images.

F. *Recycling Bin Detection*

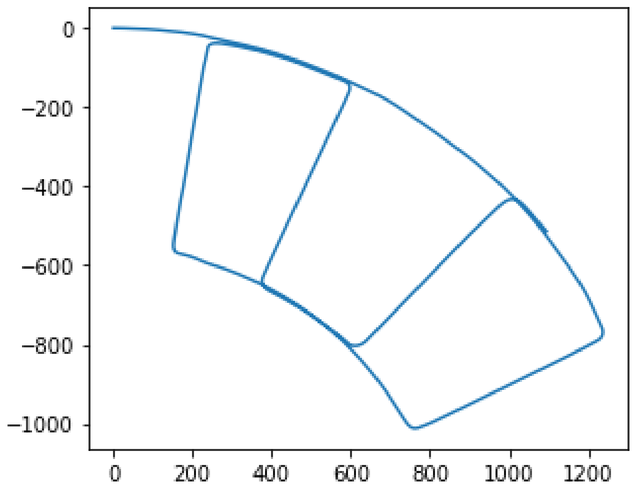
Given the training set of images containing blue recycling bins and other objects, we use the roipoly function in python to label multiple sets of RGB pixels extracted from these images into 4 broad classes: bluebin=1 (for recycling-bin blue), otherblue=2 (for other shades of blue), green=3 (for green, yellow, white) and brown=4 (for brown, black, red). We implement the same Gaussian Discriminant Analysis model as in color classification case, but now with four classes instead of three. Using the trained parameters, we then segment the recycling-bin blue-like regions from the validation images to get binary masks of bin-blue and not bin-blue regions. Afterwards, we detect the recycling bins using regionprops and label functions from scikit-image by drawing bounding boxes on the segmented regions while considering a similarity index for the typical recycling bin shape and size by checking for area of the box to be greater than 5000 pixels so that it doesn't take other smaller patches of blue regions and also for the height-width ratio to be within a range of 0.8 and 2.5. We implemented our code for bin detection on the validation set and unknown test images.

IV. Results

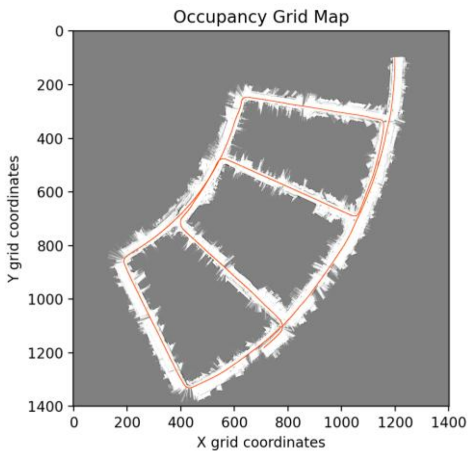
In our project, we have implemented the particle filter SLAM for $n = 5, 10, 20, 40$ particles with threshold for resampling at 40% of particle size. We update the map only at every 5 LiDAR readings for every case. We update the log-odds by adding log 4 for the free cells and subtracting $2 \times \log 4$ for the occupied cells. We have used the Gaussian noise for each particle state using zero mean and variance of $\max(\Delta x/10)$, $\max(\Delta y/10)$ and $\max(\Delta\theta/10)$.

We have plotted both the log-odds occupancy map and the binary map at various intervals along with its final wall map for $n = 5$ particles case. We have plotted the final binary map for all the other particle cases. We have also plotted the dead-reckoning trajectory path prior to using particle filter.

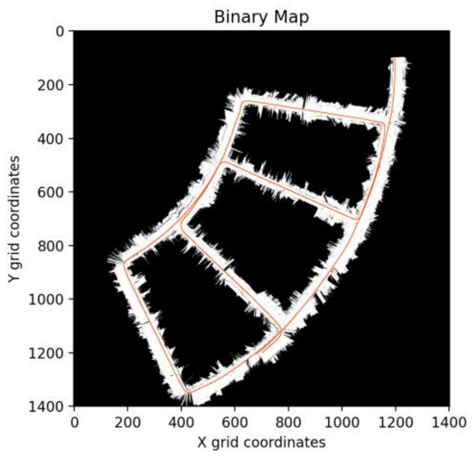
We notice that particle filter SLAM gives almost the same trajectory as the dead-reckoning trajectory with some variations. One particular thing to note is the realization of the lane (right) on which the vehicle traverses which becomes very conspicuous from the SLAM implementation. We can observe that the change in number of particles does not have much of an effect on the map or the car trajectory. Although in some cases, the trajectory seems to misalign at the very end of the journey. Also, it seems that the trajectory changes ever so slightly when run multiple times even for the same conditions. These can be attributed to high noise values that have been added which may affect the trajectory at some instances. All the different plots obtained have been attached below.



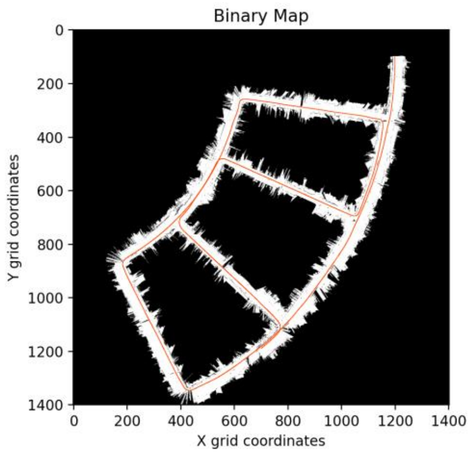
Dead reckoning trajectory plot



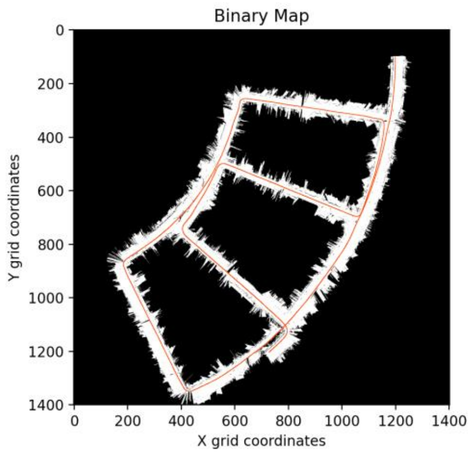
Occupancy grid map for n=5 particles



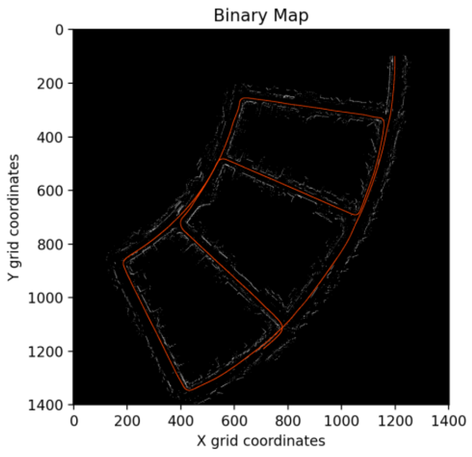
Binary map for n=10 particles



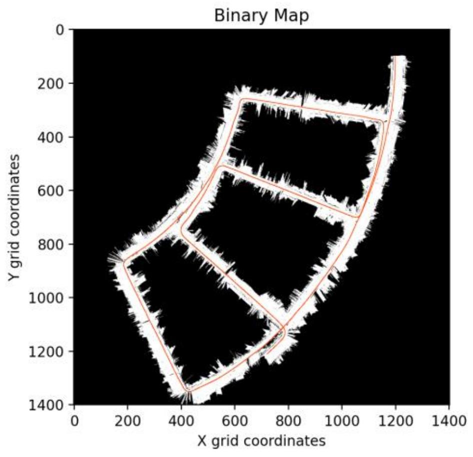
Binary map for n=5 particles



Binary map for n=20 particles

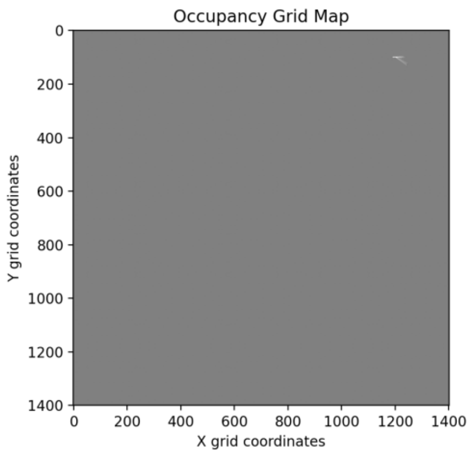


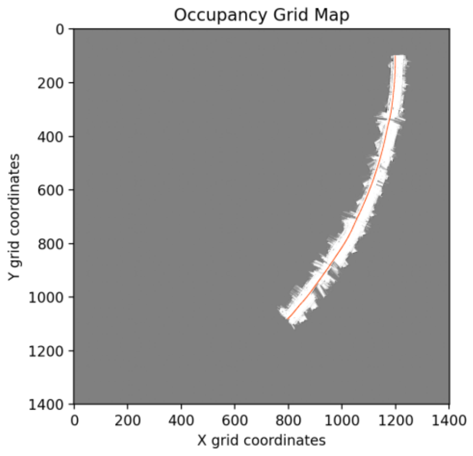
Wall map for n=5 particles

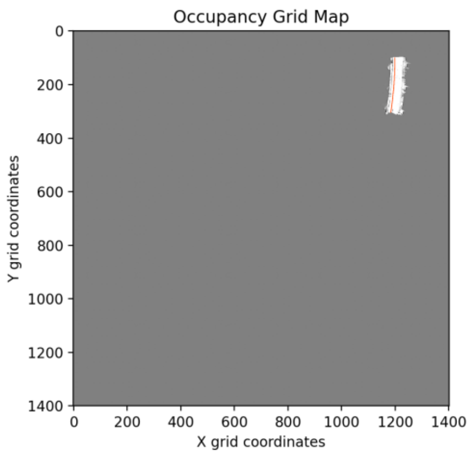


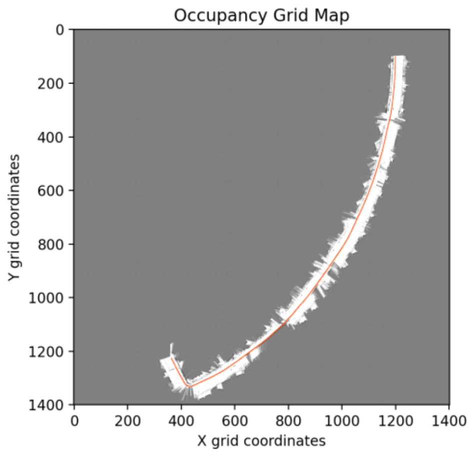
Binary map for n=40 particles

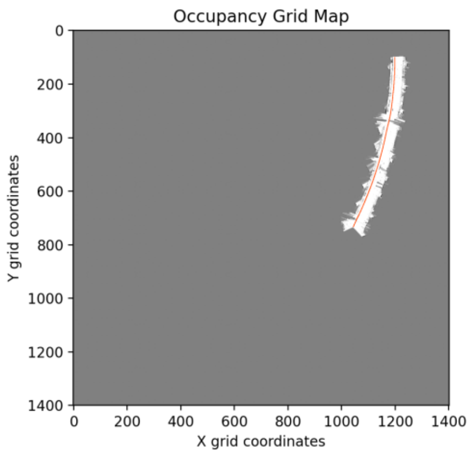
Occupancy grid maps for n=5 over time (per 10k iterations)

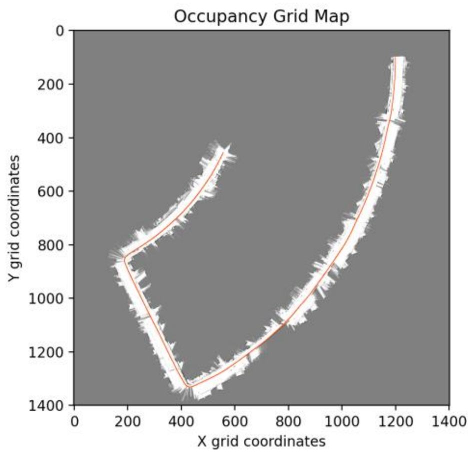


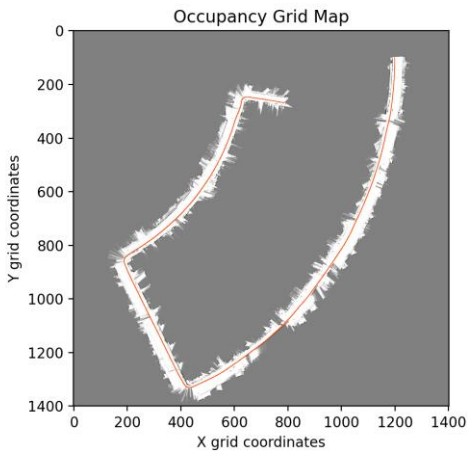


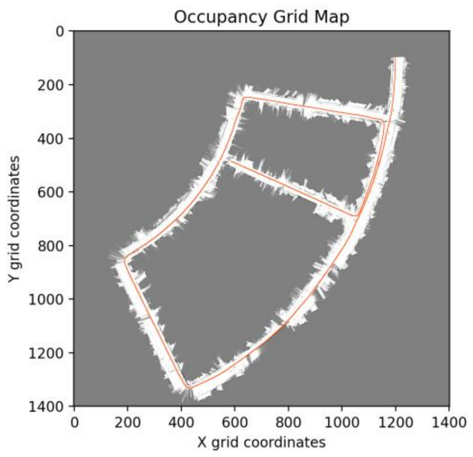


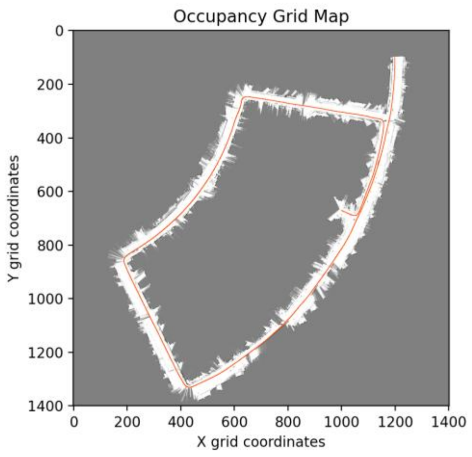


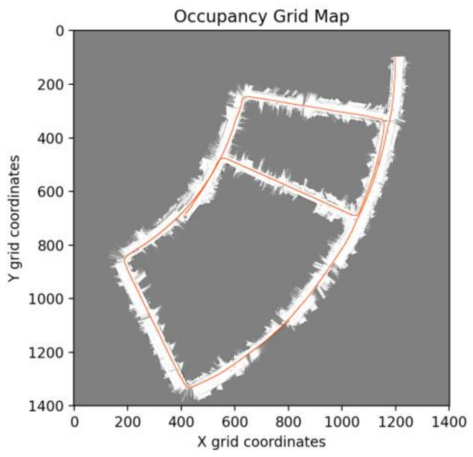


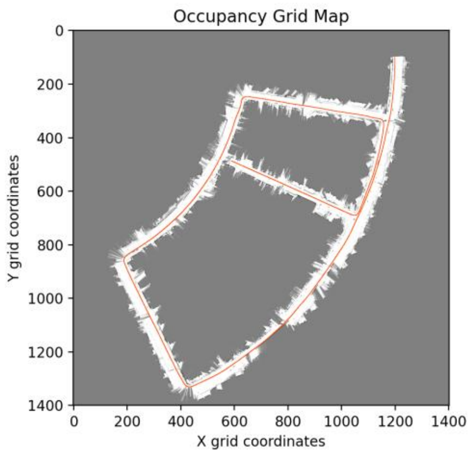


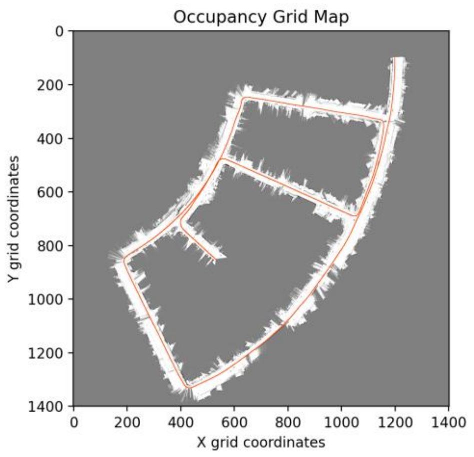

Occupancy Grid Map
Y grid coordinates
X grid coordinates
0
200
400
600
800
1000
1200
1400

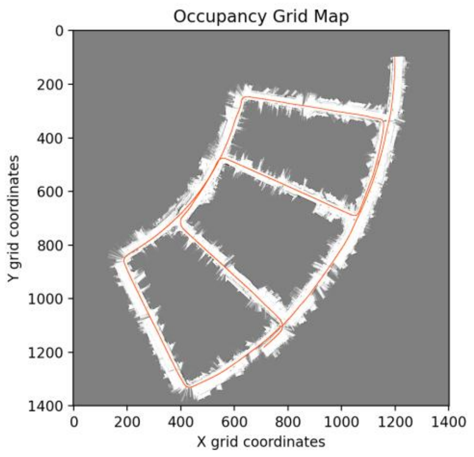

Occupancy Grid Map
Y grid coordinates
X grid coordinates
0
200
400
600
800
1000
1200
1400


Occupancy Grid Map
Y grid coordinates
X grid coordinates
0
200
400
600
800
1000
1200
1400

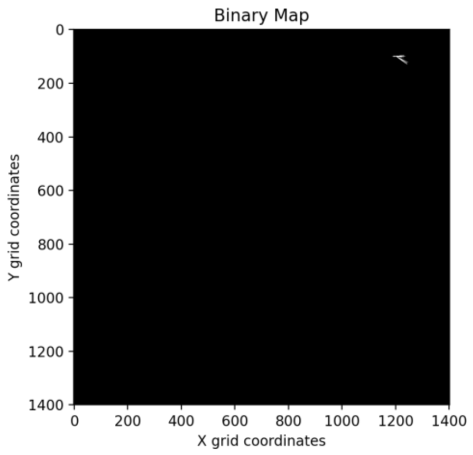

Occupancy Grid Map
Y grid coordinates
X grid coordinates
0
200
400
600
800
1000
1200
1400

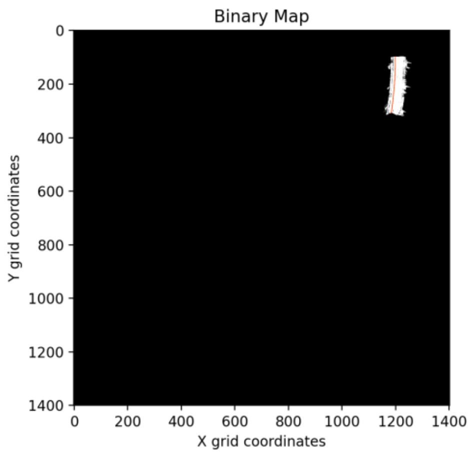

Occupancy Grid Map
Y grid coordinates
X grid coordinates
0
200
400
600
800
1000
1200
1400

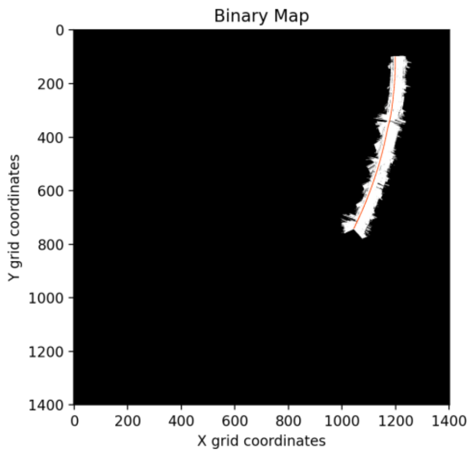


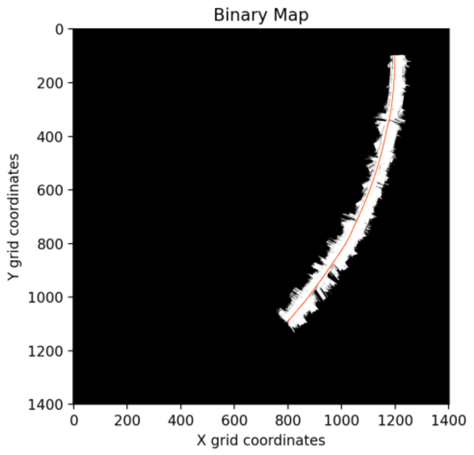


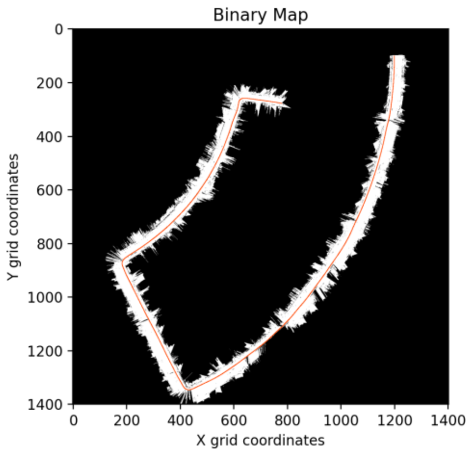
Binary maps for n=5 over time (per 10k iterations)

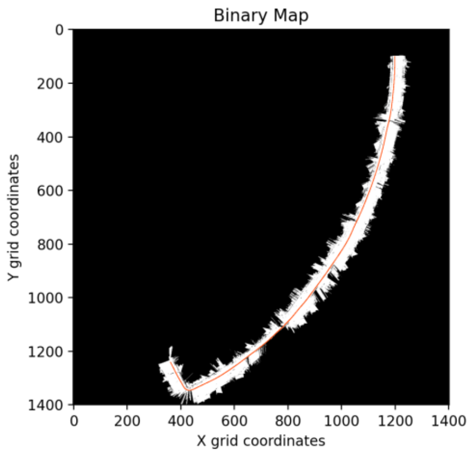


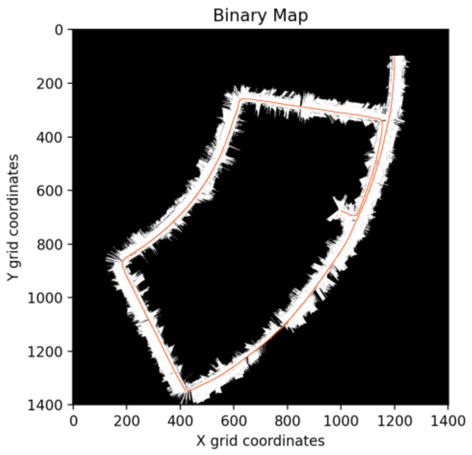


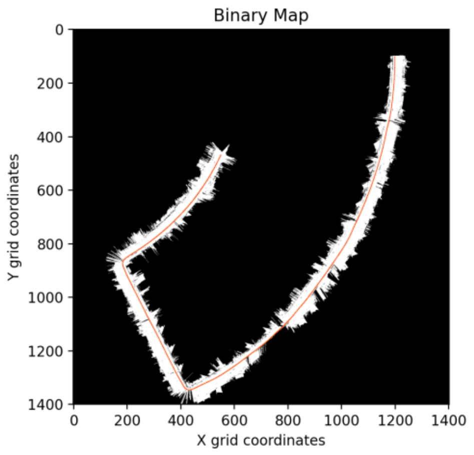


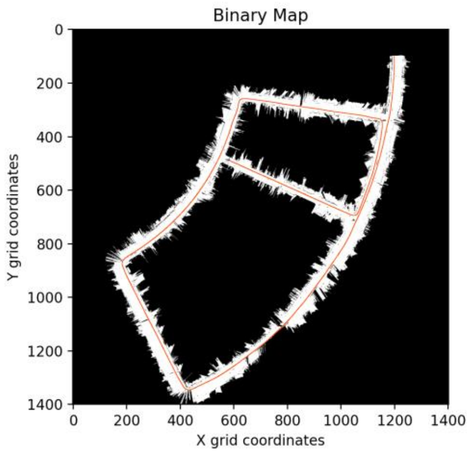

Binary Map
Y grid coordinates
X grid coordinates
0
200
400
600
800
1000
1200
1400

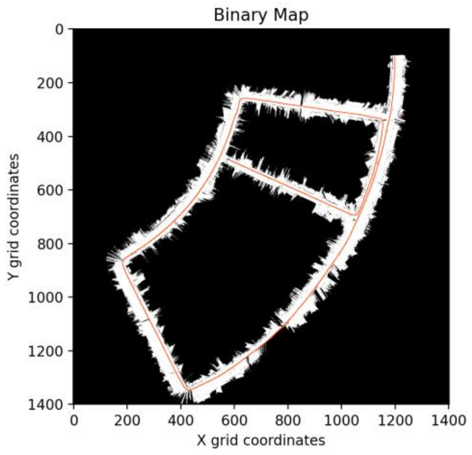

Binary Map
Y grid coordinates
X grid coordinates
0
200
400
600
800
1000
1200
1400

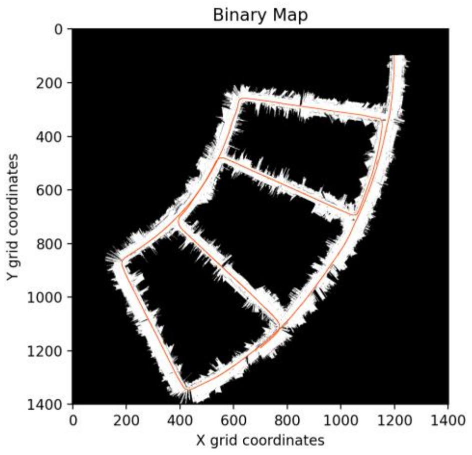

Binary Map
Y grid coordinates
X grid coordinates
0
200
400
600
800
1000
1200
1400

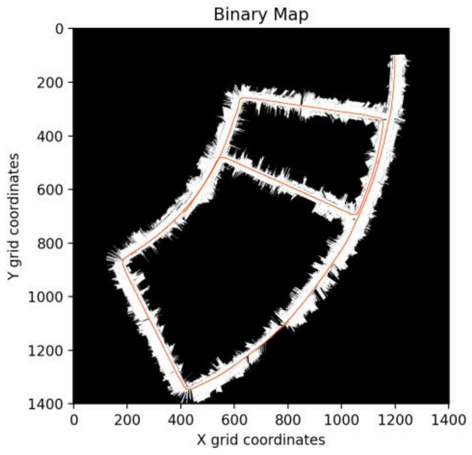

Binary Map
Y grid coordinates
X grid coordinates
0
200
400
600
800
1000
1200
1400

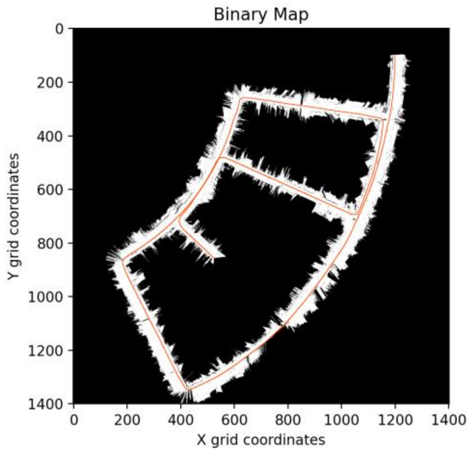

Binary Map
Y grid coordinates
X grid coordinates
0
200
400
600
800
1000
1200
1400


Binary Map
Y grid coordinates
X grid coordinates
0
200
400
600
800
1000
1200
1400


Binary Map
Y grid coordinates
X grid coordinates
0
200
400
600
800
1000
1200
1400


Binary Map
Y grid coordinates
X grid coordinates
0
200
400
600
800
1000
1200
1400


Binary Map
Y grid coordinates
X grid coordinates
0
200
400
600
800
1000
1200
1400


Binary Map
Y grid coordinates
X grid coordinates
0
200
400
600
800
1000
1200
1400