Particle Filter SLAM

Eskil Berg Ould-Saada
University of California San Diego
La Jolla, CA92093
`eskilbo@stud.ntnu.no`

1 Introduction

In the latter years, Simultaneous Localisation and Mapping (SLAM) has been an important part of computer vision and AI when it comes to autonomous systems with motion planning and estimation in general. Especially for self-driving drones discovering an undiscovered area, a good system for localisation and mapping is immense. In short, SLAM is the problem of creating and updating a map of an unknown area whilst keeping track of an agent's localisation at the same time. To solve this problem, several sensors and different techniques are used.

In this project, a car driving on a road equipped with wheel encoders, a LiDAR sensor and a fiber optic gyro (FOG) was used to create the map and localise the car using a particle filter. A model of the autonomous car with the sensors can be found in Figure 1.For the mapping part, the sensor measurements and the car's state were used to update the map, and the localisation part uses the map of the environment to estimate the car's trajectory. Throughout this report, a problem formulation will be presented in mathematical terms, as well as a description of the technical approach, before the results will be presented.

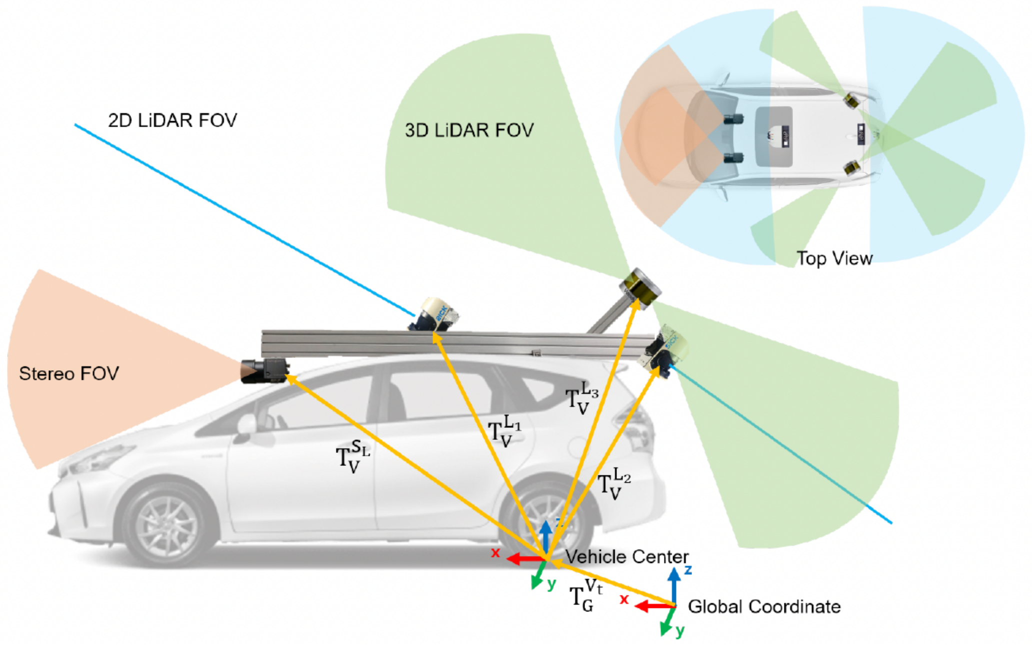

Figure 1: The antonomous car with its sensors.

2 Problem Formulation

The problem wanting solving in this project is localization of the driving car as well as creating an occupancy map and a texture map.

2.1 Dataset

The dataset given in this project consists of four parts. Firstly, the encoder values for the left and right wheel are given with resolution of 4096 and wheel diameters of respectively 0.6235 and 0.6228 meters. The next dataset is the FOG data, consisting of approximately ten times as many timestamps as the other datasets, and thus the data needed to be synchronized to match the other data. The FOG data consists of the changes in roll, pitch and yaw between timestamps. Furthermore, the 2D LiDAR sensor consisted of sensor values of ranges with a maximum range of 80 meters and between the angles of -5°and 185°. Lastly, stereo images are provided for texture mapping.

2.2 SLAM Problem Formulation

SLAM is a state estimation problem, and can be formulated as a probabilistic Markov Chain. Given a robot's state $\mathbf{x}_t$ and control input $\mathbf{u}_t$ at time steps t, and using the Markov assumptions of factorization of joint pdf's of the states, inputs and observations, the SLAM problem can be formulated as follows:

$$p(\mathbf{x}_{0:T}, \mathbf{z}_{0:T}, \mathbf{u}_{0:T-1}) = p(\mathbf{x}_0) \prod_{t=0}^{T} p_h(\mathbf{z}_t|\mathbf{x}_t) \prod_{t=1}^{T} p_f(\mathbf{x}_t|\mathbf{x}_{t-1}, \mathbf{u}_{t-1}) \prod_{t=0}^{T-1} p(\mathbf{u}_t|\mathbf{x}_t) \tag{1}$$

where the first part is the prior probability, the second is the observation model, the third is the motion model and the fourth is the control policy.

2.3 Particle Filter Problem Formulation

To use the particle filter, a special case of the Bayesian filter, one must define the different parts of the particle filter algorithm in mathematical terms. The particle filter consists of four parts: prior probability, re-sampling, prediction and updating. Recall that a particle is a hypothesis that the value $\mathbf{x}$ is $\mu^{(k)}$ with a probability $\alpha^{(k)}$.

The prior is given as

$$\mathbf{x}_t|\mathbf{z}_{0:t}, \mathbf{u}_{0:t-1}\ p_{t|t}(\mathbf{x}_t) := \sum_{k=1}^{N} \alpha_{t|t}^{k} \delta(\mathbf{x}_t; \mu_{t|t}^{(k)}) \tag{2}$$

If we let $\mu_{t+1|t}^{(k)}\ p_f(*|\mu_{t|t}^{(k)}, u_t)$ and $\alpha_{t+1|t}^{(k)} = \alpha_{t|t}^{(k)}$, the prediction step in the particle filter algorithm becomes

$$p_{t+1|t}(\mathbf{x}) \approx \sum_{k=1}^{N} \alpha_{t+1|t}^{(k)} \delta(\mathbf{x} - \mu_{t+1|t}^{(k)}) \tag{3}$$

The update step is

$$p_{t+1|t+1}(\mathbf{x}) = \sum_{k=1}^{N} \left[\frac{\alpha_{t+1|t}^{(k)} p_h\left(\mathbf{z}_{t+1} \mid \mu_{t+1|t}^{(k)}\right)}{\sum_{j=1}^{N_{t+1|t}} \alpha_{t+1|t}^{(j)} p_h\left(\mathbf{z}_{t+1} \mid \mu_{t+1|t}^{(j)}\right)} \right] \delta\left(\mathbf{x} - \mu_{t+1|t}^{(k)}\right) \tag{4}$$

2.4 Occupancy Grid Mapping

The problem of occupancy grid mapping consists of generating a map $\mathbf{m}$ from noisy sensor observations $\mathbf{z}$ given a robot's state $\mathbf{x}$. The different cells in the map are either positive or negative depending if they are free or occupied. The goal of this map is to estimate the posterior probability $p(\mathbf{m}|\mathbf{z}_{0:T}, \mathbf{x}_{0:T})$ over time. As the different cells are independent, the distribution of each cell can be expressed individually so one gets

$$p(\mathbf{m}|\mathbf{z}_{0:t}, \mathbf{x}_{0:t}) = \prod_i p(\mathbf{m}_i|\mathbf{z}_{0:t}, \mathbf{x}_{0:t}) \tag{5}$$

2.5 Texture Mapping

Given a map $\mathbf{m}$, texture mapping aims at forming a vector $\mathbf{m} \in \mathbb{R}^{nx3}$ where each element in the vector is an RGB color corresponding to the cell in the occupency grid map.

3 Technical Approach

This section will contain all the technical approach done in this project. In general, when iterating through all the time steps, if there is an encoder reading, the robot's next position is predicted, and if there is a LiDAR reading, the map and the robot's position is updated.

3.1 Pre-processing the data

The first thing done after observing that the FOG values were sampled at a rate about ten times higher than the encoder data was to synchronize the FOG data to the same sampling rate as the rest of the data. This was done by summing up ten FOG values at a time and assigning them to as many time stamps as the encoder data. Furthermore, as the angular velocity is to be used later, all ω's were pre-computed by dividing the summed up FOG values by the difference in time between the first and the last summed FOG value.

For the LiDAR data, scanned points too close to the sensor and too far from the sensor were removed from the dataset to reduce computing time and removing unnecessary scan points. The range of LiDAR points considered in this project is $[1.5, 55]$ meters.

3.2 Particle Prediction

While iterating through the time steps, if an encoder value is present at a time step, a prediction using the differential-drive motion model given in Equation 6 is used, taking in account an added Gaussian noise to make the model more robust. Recall that $[x_t, y_t, \theta_t]$ is the particle's state at a given time t.

$$\begin{bmatrix} x_{t+1} \\ y_{t+1} \\ \theta_{t+1} \end{bmatrix} = \begin{bmatrix} x_t \\ y_t \\ \theta_t \end{bmatrix} + \begin{bmatrix} dx \\ dy \\ d\theta \end{bmatrix} + \begin{bmatrix} N(0, \sigma_x) \\ N(0, \sigma_y) \\ N(0, \sigma_\theta) \end{bmatrix} \tag{6}$$

where

$$\begin{bmatrix} dx \\ dy \\ d\theta \end{bmatrix} = \tau \begin{bmatrix} v_t \cos(\theta_t) \\ v_t \sin(\theta_t) \\ \omega_t \end{bmatrix} \tag{7}$$

and v_t is computed using the following equation

$$\tau_t v_t \approx \frac{\pi d z_t}{4096} \tag{8}$$

where τ is the time between two encoder readings, d is the diameter of the wheel, z is the encoder reading and 4096 is the encoder resolution. This is done for both wheels before an average is computed to find the linear velocity. The first velocities are around $4.6\frac{\mathrm{m}}{\mathrm{s}}$.

The choice of noise ended up in being the maximum value of each of the particle's state's components divided by 10. For confirmation that the prediction step was implemented correctly, see Figure 4 in subsection 4.1.

3.3 Particle Update

The particles are updated by usage of the already provided correlation function. Firstly, the new LiDAR scan is converted to world frame from LiDAR frame using the transformation matrices from LiDAR to Body, then Body to World.

A 9x9 grid around the points is then made, and is used in the correlation function for each of the particles. For each of the spawned particles with the 9x9 grids, a correlation to the map is computed to find the particle with the best correlation. This correlation vector is then used to update the different particle's weights, by first using softmax then normalizing the weights again so that they sum to 1. The particle that has the new highest particle weight becomes the agent's current location and is used to update the log-odds map. When updating the map (see next part), the start state will be the state of the previously highest weighted particle.

To avoid weight collapse caused by weight disparity, resampling is done where weights are replaced with better weights if they are negligible. If at a time, the number of effective particles is under 20 (that is half of all particles in this project), the particles are updated with a stratified resampling step. This is done by following the algorithm in Figure 2.

```
Stratified (low variance) resampling
1: **Input**: particle set $\left\{\boldsymbol{\mu}^{(k)}, \alpha^{(k)}\right\}_{k=1}^{N}$
2: **Output**: resampled particle set
3: $j \leftarrow 1,\ c \leftarrow \alpha^{(1)}$
4: **for** $k = 1, \ldots, N$ **do**
5:     $u \sim \mathcal{U}\left(0, \frac{1}{N}\right)$
6:     $\beta = u + \frac{k-1}{N}$
7:     **while** $\beta > c$ **do**
8:         $j = j + 1,\ c = c + \alpha^{(j)}$
9:     add $\left(\boldsymbol{\mu}^{(j)}, \frac{1}{N}\right)$ to the new set
```

Figure 2: Resampling.

3.4 Map Update

The map is initialized as a $1350x1350$ grid with 1 meter resolution, and spawn a set of 40 particles with states as given in subsection 3.2 that are initialized with zeros. A set of weights belonging to each of these particles were also initialized, each having the weight $\frac{1}{40}$ to begin. When updating the map, the particle with the largest weight is used as the best particle and is used for the map update.

Given a LiDAR scan update, the points need to be converted to grid cells from the LiDAR frame. This is done by transforming the particle state points from the LiDAR frame to the body frame and then to world frame. They are then converted into grid cells and sent through the bresenham2D algorithm for finding occupied and free cells. The values of the cells in the log-odds map marked as free are decreased by $\log(4)$ while the occupied are increased by $\log(4)$. To recover the maps, the logistic sigmoid function is used and the log-odds map is recovered.

The first LiDAR scan gives the output found in Figure 3.

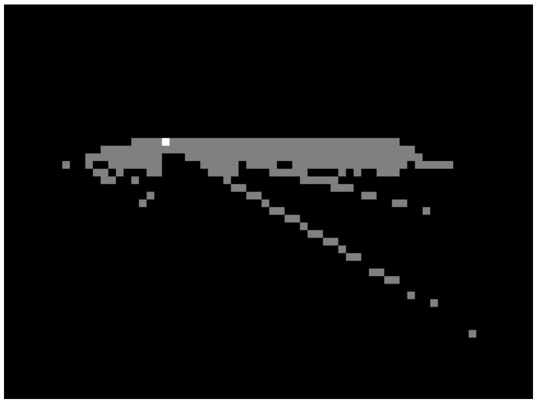

Figure 3: First LiDAR scan.

3.5 Texture Mapping

This part was not completed due to shortage of time.

4 Results

In this part, the results of the SLAM will be presented.

4.1 Dead Reckoning

To ensure that the prediction was done correctly and that the motion model was implemented correctly without noise, dead reckoning of the trajectory of the car was done. The result can be seen in Figure 4, and shows that the implemented was correct, as the path seems close to perfect.

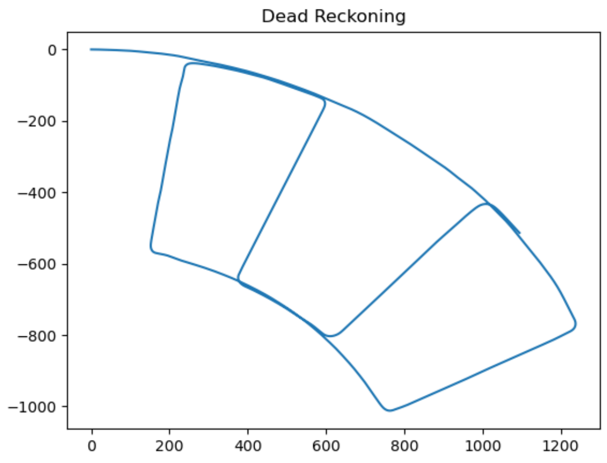

Figure 4: Dead reckoning trajectory.

4.2 Occupancy Map and Trajectory

In Figure 5-Figure 8 one can observe the occupation map and the trajectory while the program was running, and the final map can be found in Figure 9. One can observe that the trajectory of the vehicle in all these plots turn prematurely for some of the turns, which can be a sign that the added noise is too high. An experiment could be to decrease the noise and observe if the trajectory and map gets closer to the actual path. However, all things considered, the model performed rather well, even though the estimated path was a bit off for some of the turns.

It is clear from Figure 4 how the path of the vehicle should be, and one can observe that the paths in the occupancy map are flipped, probably due to the fact that the x and y coordinates are flipped, but it is still clear that the occupancy map and trajectory follow approximately the right way.

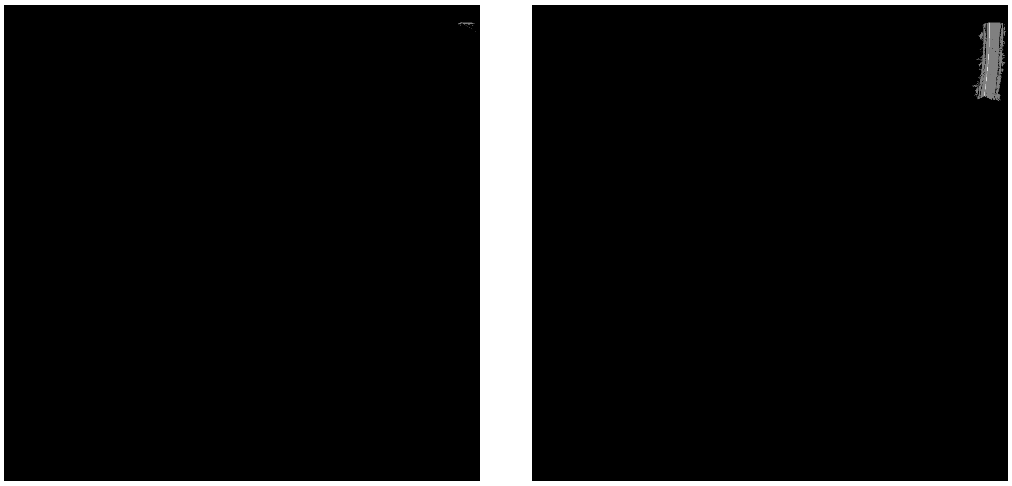

Figure 5: Occupancy map after 0 and 30000 iterations.

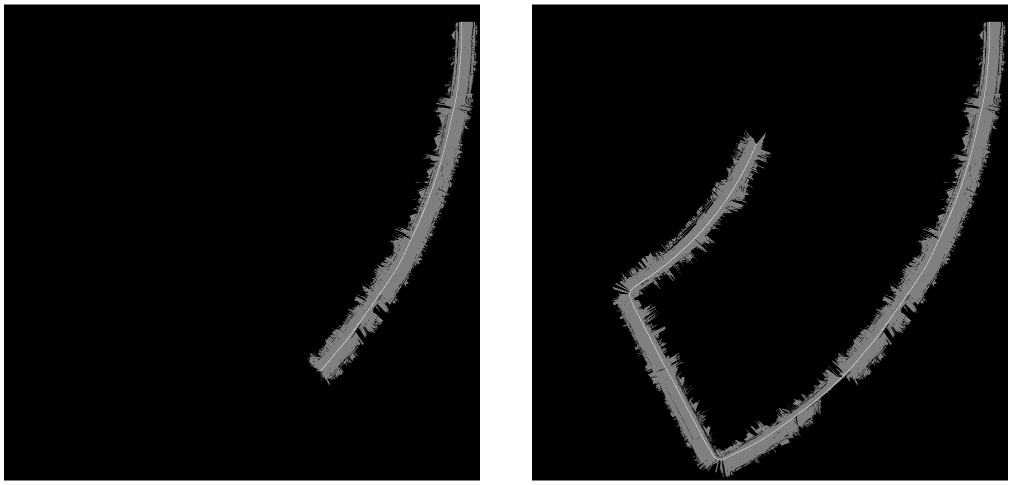

Figure 6: Occupancy map after 60000 and 100000 iterations.

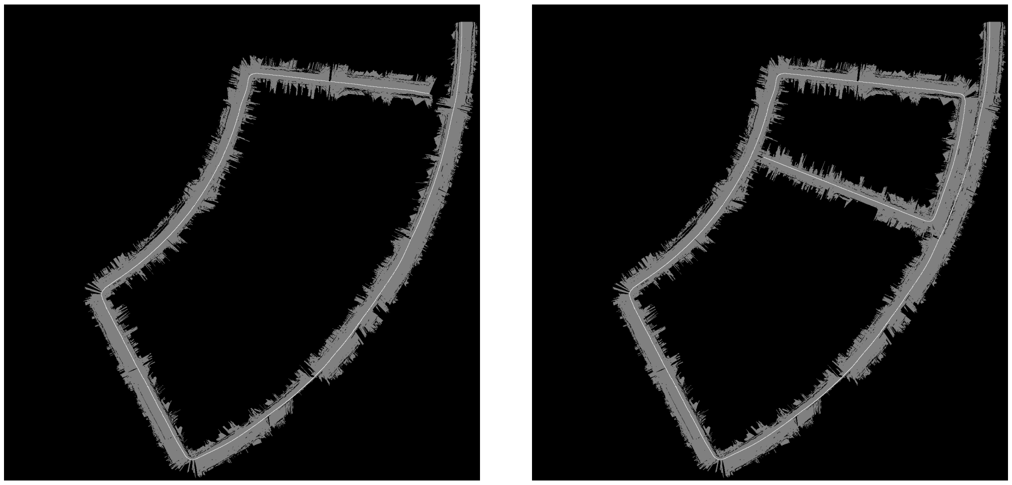

Figure 7: Occupancy map after 130000 and 160000 iterations.

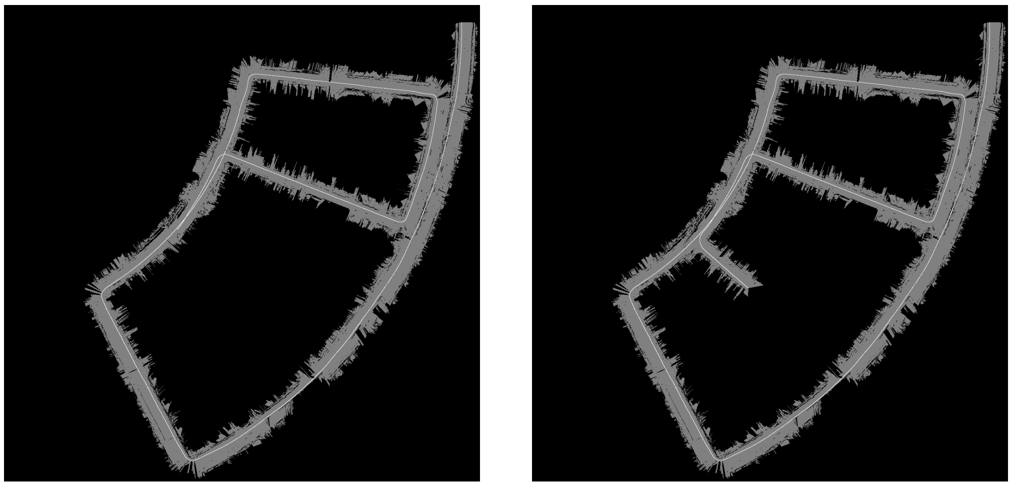

Figure 8: Occupancy map after 190000 and 220000 iterations.

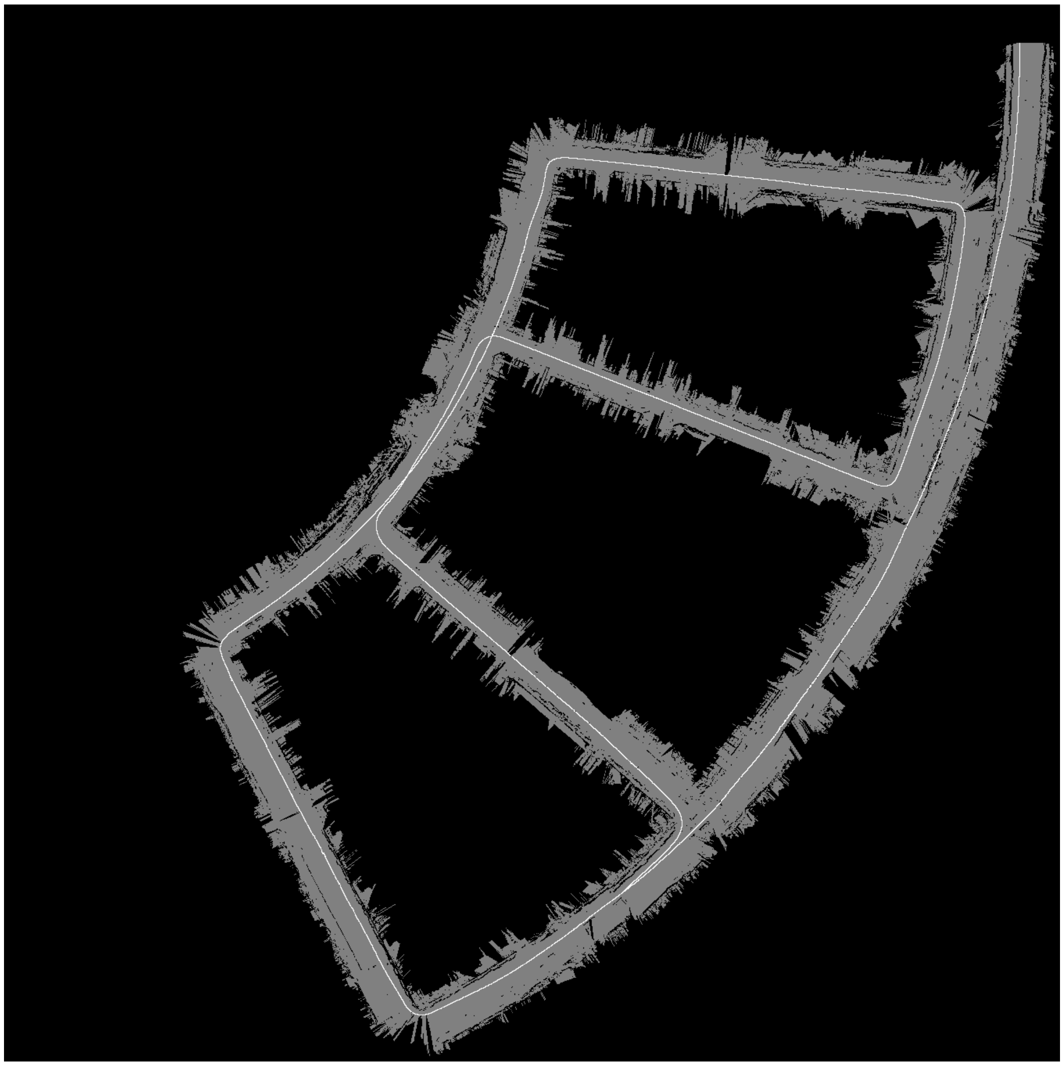

Figure 9: The final occupancy map.