Particle Filter SLAM

Yuanjun “Dastin” Huang

Introduction

Simultaneous localization and mapping (SLAM) is an important problem in robotics which aims to building a map representing the environment and tracking the trajectory of the robot at the same time. SLAM can be found in many real-world robotics applications. For example, a Roomba vacuum cleaner utilizes SLAM to build a map of an unseen floorplan and navigates itself among the household. Another commonly seen application is autonomous driving vehicle.

In many SLAM problems, two types of data are fed to the algorithm. The first type of data is related to the kinematic model of the robot, which is utilized to predict the relative pose offset between two consecutive steps. The other type of data represents the environment perceived at the current robot location, which can be utilized to build the map representation of the environment and revise the pose estimation based on the first type of data. In many cases, IMU measurement is adopted as the first type of data. For the second type of data, visual measurement provided by cameras and/or point cloud measurement provided by LIDAR are widely adopted.

Particle Filter and Extended Kalman Filter are two algorithms widely adopted to propagate SLAM across time. In this project, we focus on implementing a particle filter with IMU measurement and visual measurement as input.

Problem Formulation

At time step t, the $SE(3)$ state of the vehicle can be separated into 2 parts: R_t representing the rotation (orientation) and T_t representing the translation of the vehicle. In 3D environment, $R_t \in SO(3)$ and $T_t \in \mathbb{R}^3$. Since the vehicle doesn't have significant movement along z-axis, we can simplify the state space to $R_t \in SO(2)$ and $T_t \in \mathbb{R}^2$. We use $V_t = \{R_t, T_t\}$ to denote the state of vehicle at time t.

We denote the IMU measure at time t as $I_t = \{s_t, \Delta\theta_t\}$; the 3D point cloud representation of the environment in vehicle frame at time t as $L_t = \{p_{tk} \in \mathbb{R}^3, k = [1, K]\}$, where K denotes the total number of points; the stereo image set at time t as $SC_t = \{SC_{tl}, SC_{tr}\}$.

In order to build a 2D representations of the street environment, we adopt the following 2 approaches: 1) grid occupying map, i.e., a binary map M_o denoting whether each cell is occupied or not; 2) colored texture map, i.e., a 2D image M_c denoting the RGB color of each cell in the grid occupying map.

Our goal is to estimate $V_{1:Tl}, M_{o\,1:Tl}, M_{c\,1:Tl}$ based on $I_{1:Ti}$, $L_{1:Tl}, SC_{1:Ts}$ where T_i, T_l, T_s denotes the number of time steps of IMU measurement, LIDAR measurement, stereo camera image pairs, respectively.

Technical Approach

We maintain N number of particles to represent V.

At each updating step, we first predict each particle's pose at next time step using its IMU measurement by

$$[x_{t+1}, y_{t+1}, \theta_{t+1}] = [x_t + ds\cos(\theta_t), y_t + ds\sin(\theta_t), \theta_t + \Delta\theta_t]$$

where $ds, \Delta\theta_t$ can be derived from raw IMU measurement with simple calculation.

Then, we extract the 3D point cloud representation of the environment from its LIDAR measurement. We apply $p_w = R_t R_L p_L + T_{nt}$ to acquire the point coordinates in world frame, where R_t, R_L denote the 3D orientation of vehicle and relative orientation between LIDAR and vehicle, respectively. For each particle, we calculate the correlation between observed LIDAR measurement and inferred LIDAR measurement. This is approached by adding a number of small perturbations around current vehicle position and taking the maximum of the similarity between measurement and each inferred measurement using

$$c = \max\{similarity(M_o, L_t; \Delta T)\}$$

Then each particle's weight will be updated by

$$w_n = \frac{\exp(c_n)}{\sum \exp(c_i)}$$

In the next step, we update the grid occupancy map. We pick the particle with largest weight and evaluate every cell of the grid occupying map on the route from the particle's position to that point: if the LIDAR beam can pass the cell, then the cell's logit will be decreased by 1; otherwise, its logit will be increased by 1. After that, we threshold the cell logits by 0 to get the grid occupying map.

The last step is to calculate the colored map. With each stereo image pair, we can utilize OpenCV's disparity computing method to acquire a disparity map. Based on the disparity map and left camera's acquired image, we can restore the world frame 3D coordinate of each eligible pixel by

$$\begin{bmatrix} u_L \\ v_L \\ d \end{bmatrix} = \begin{bmatrix} fs_u & 0 & c_u & 0 \\ 0 & fs_v & c_v & 0 \\ 0 & 0 & 0 & fs_u b \end{bmatrix} \frac{1}{z} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \qquad \begin{bmatrix} x \\ y \\ z \end{bmatrix} = {}_oR_r R^\top (\mathbf{m} - \mathbf{p})$$

We then select all pixels with world-frame z-axis coordinate in [0,1], and find the associated cells in the colored map. Each cell's RGB value is set to be the same as the associated pixel.

Result

We run a prediction step at every IMU measurement and run an update step at every LIDAR measurement. The colored map is updated at every stereo camera measurement. All data is synced before update step and colored map calculation step. We report the grid occupying map, colored texture map and vehicle trajectory every 100 images.

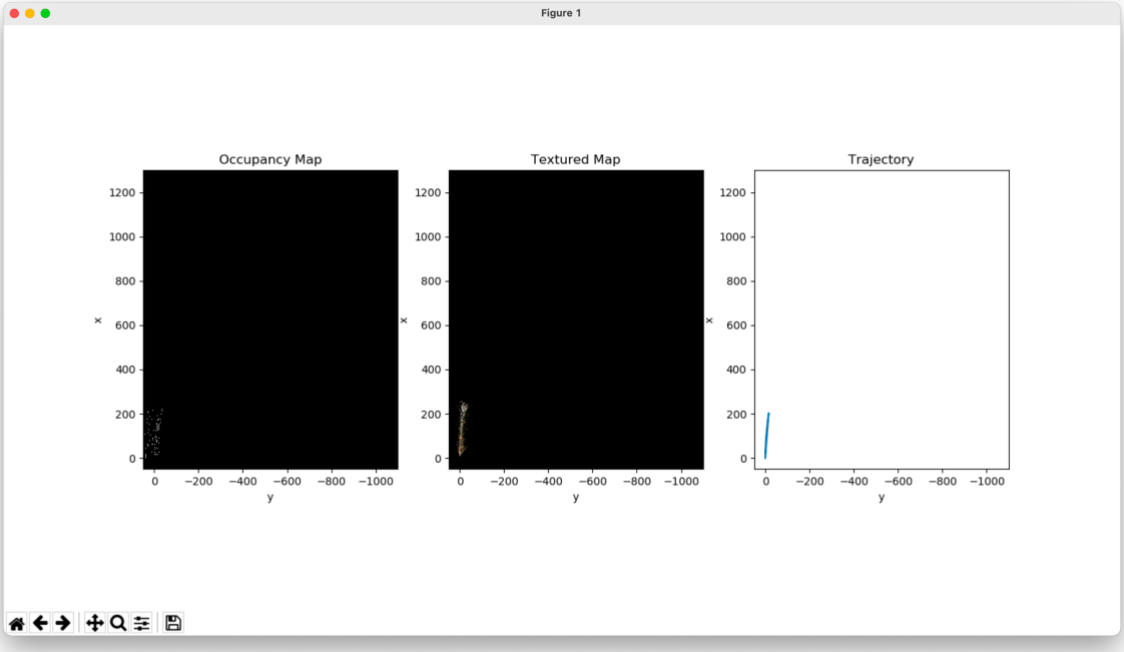


Figure 1
Occupancy Map
Textured Map
Trajectory
x
y
1200
1000
800
600
400
200
0
−200
−400
−600
−800
−1000

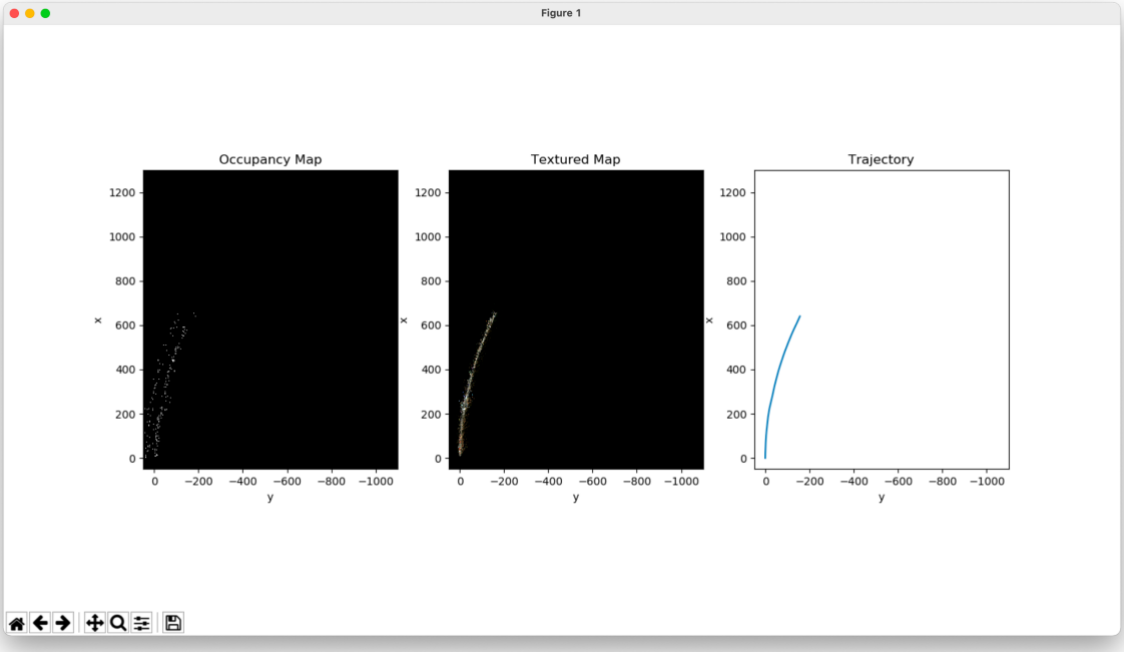


Figure 1
Occupancy Map
Textured Map
Trajectory
x
y
1200
1000
800
600
400
200
0
−200
−400
−600
−800
−1000

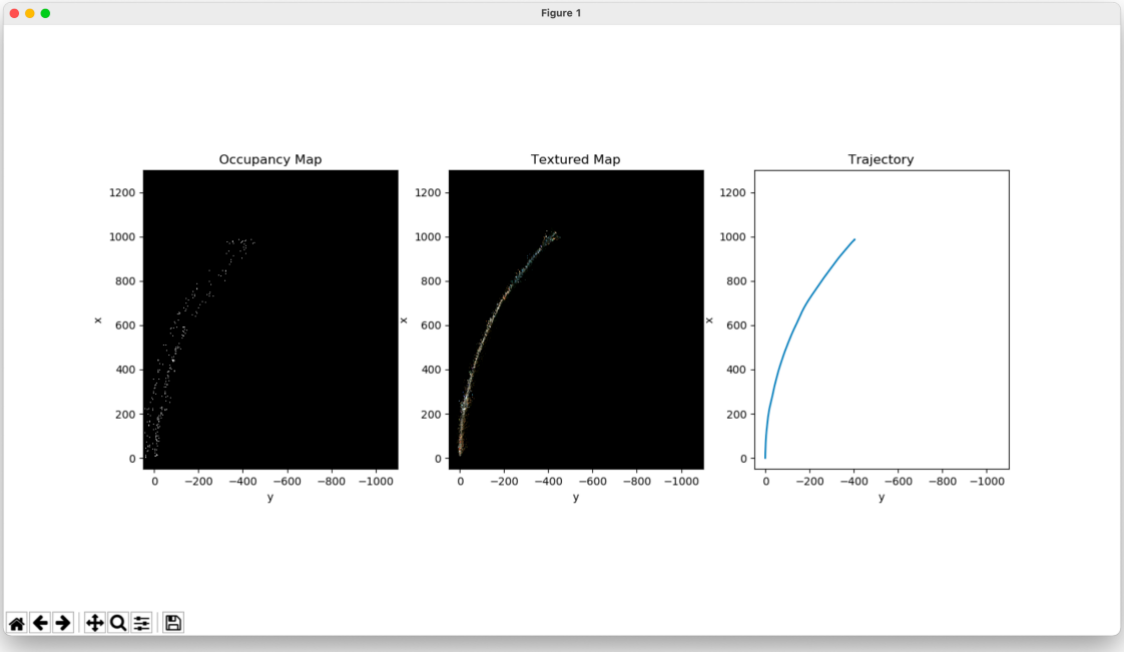


Figure 1
Occupancy Map
Textured Map
Trajectory
x
y
1200
1000
800
600
400
200
0
−200
−400
−600
−800
−1000

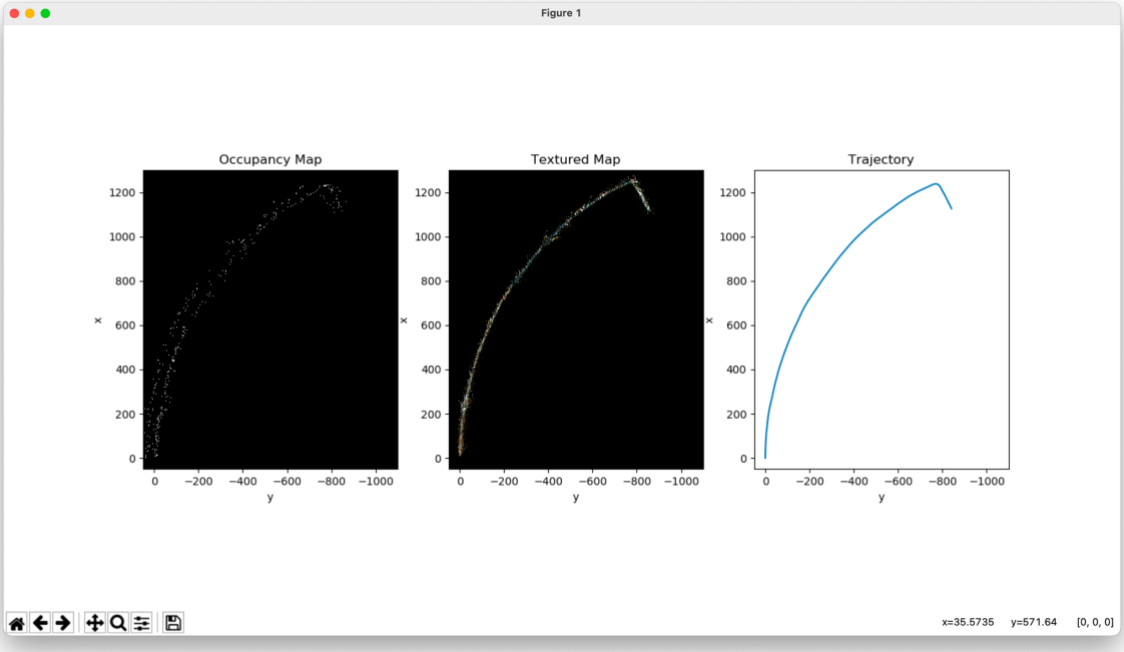


Figure 1
Occupancy Map
Textured Map
Trajectory
x
y
1200
1000
800
600
400
200
0
−200
−400
−600
−800
−1000
x=35.5735 y=571.64 [0, 0, 0]

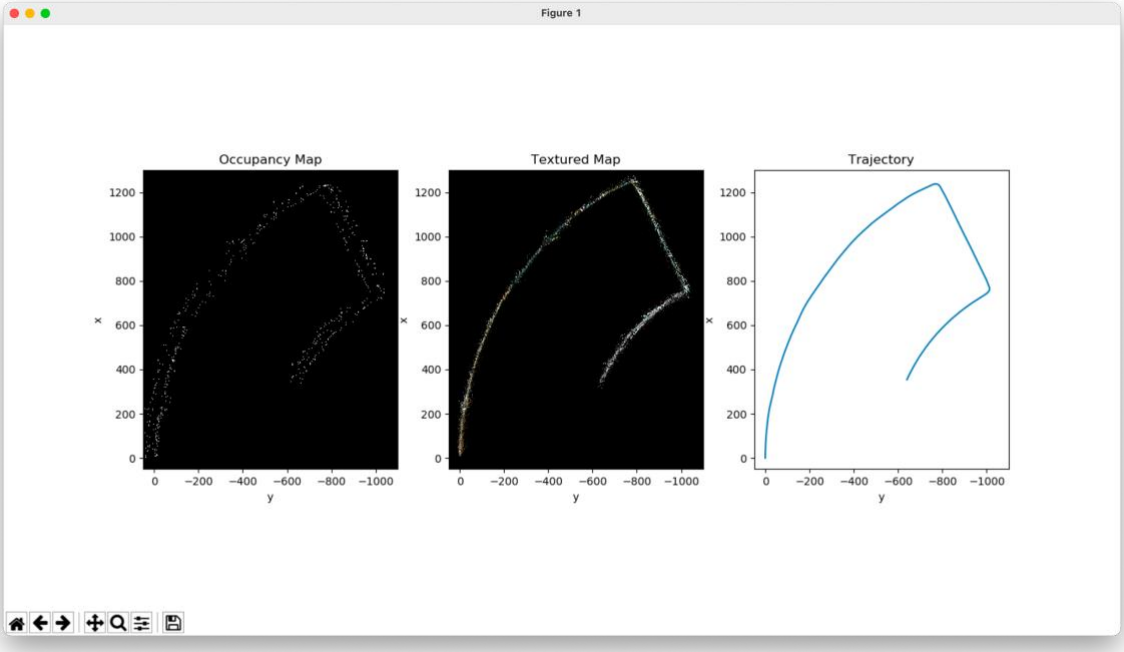


Figure 1
Occupancy Map
Textured Map
Trajectory
x
y
0
200
400
600
800
1000
1200
−200
−400
−600
−800
−1000

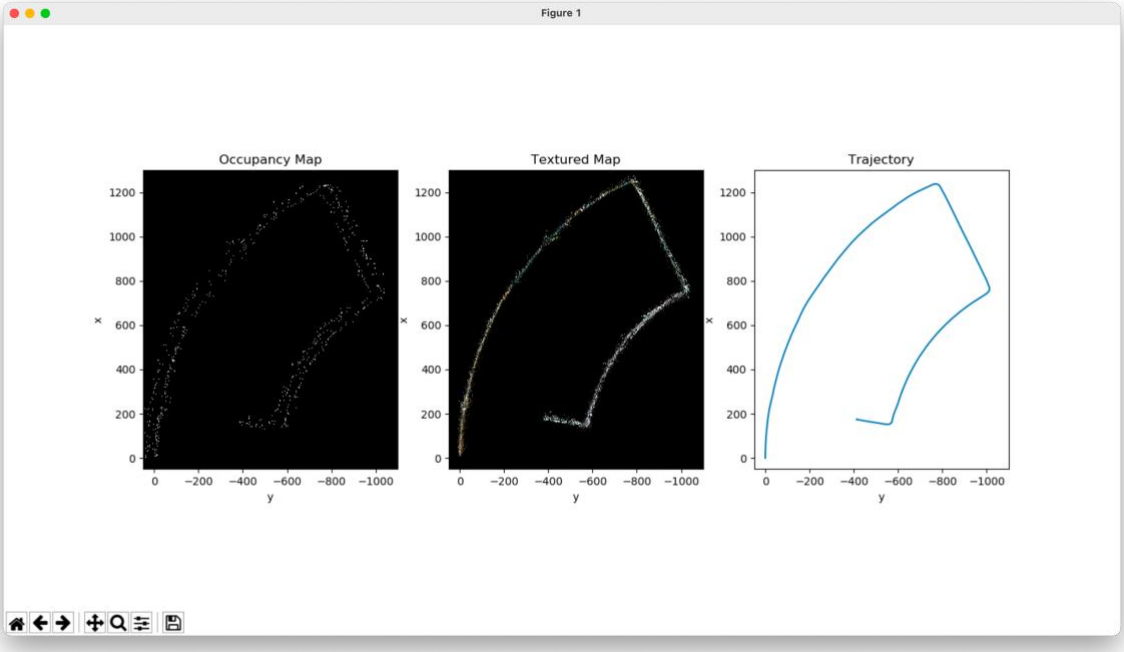


Figure 1
Occupancy Map
Textured Map
Trajectory
x
y
0
200
400
600
800
1000
1200
−200
−400
−600
−800
−1000

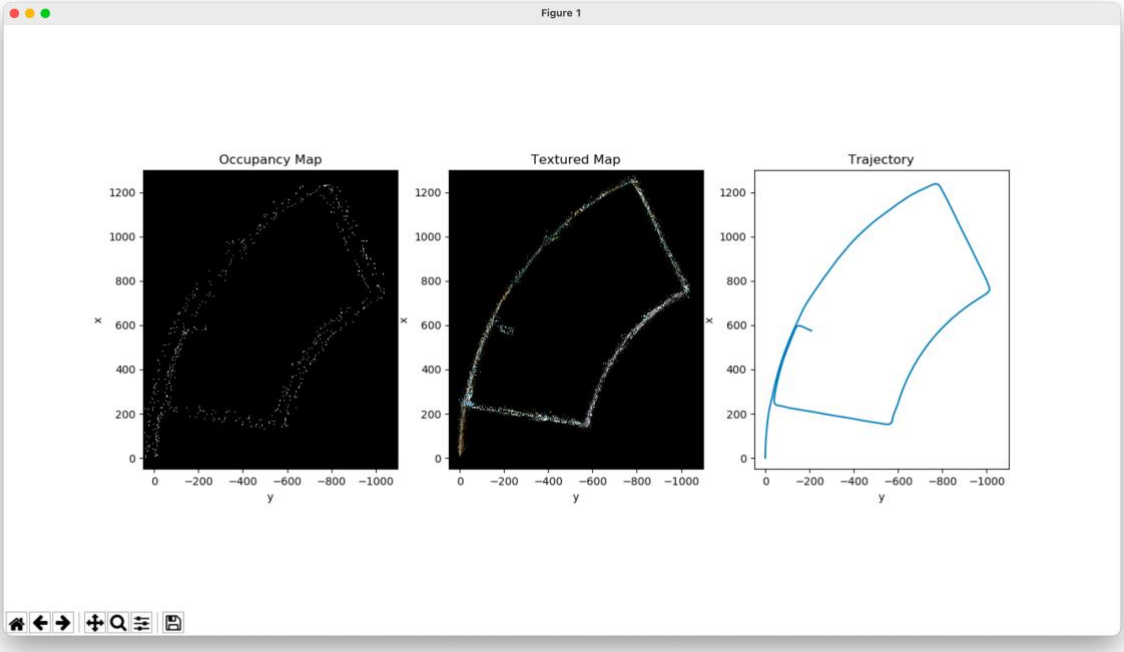


Figure 1
Occupancy Map
Textured Map
Trajectory
x
y
0
200
400
600
800
1000
1200
−200
−400
−600
−800
−1000

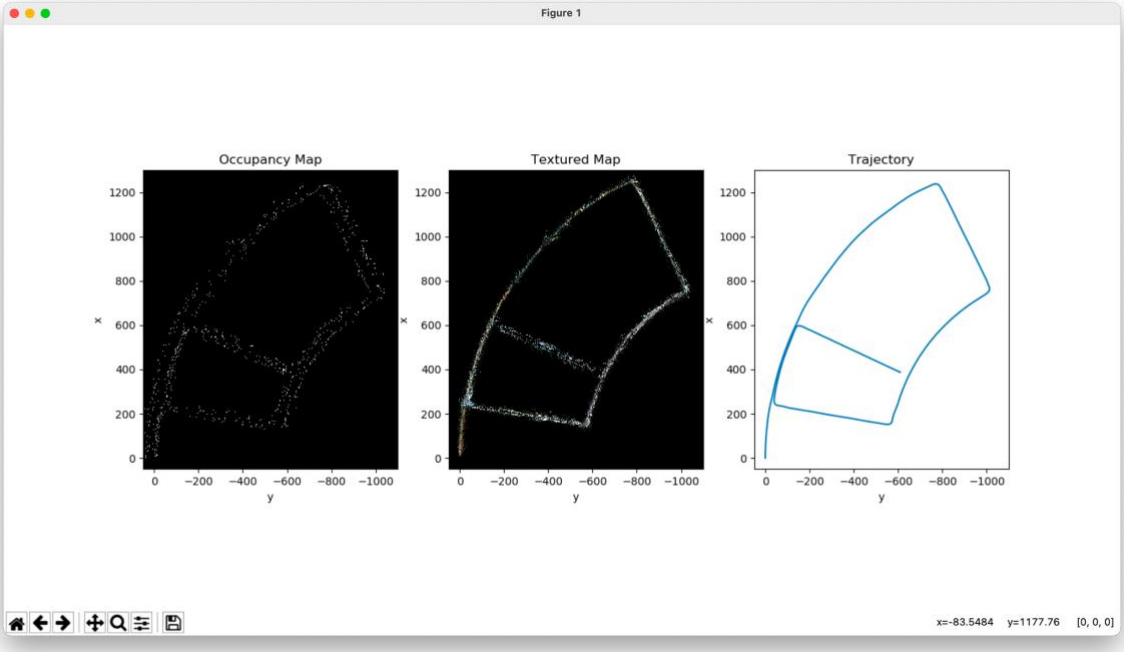


Figure 1
Occupancy Map
Textured Map
Trajectory
x
y
0
200
400
600
800
1000
1200
−200
−400
−600
−800
−1000
x=-83.5484 y=1177.76 [0, 0, 0]

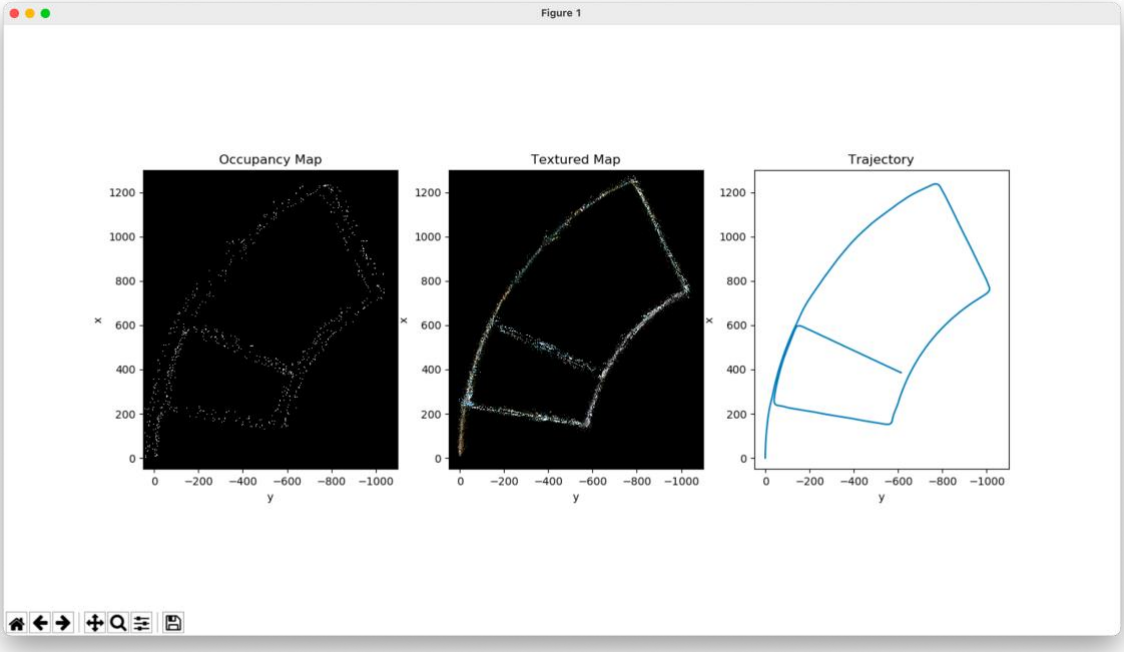


Figure 1
Occupancy Map
Textured Map
Trajectory
x
y
0
200
400
600
800
1000
1200
−200
−400
−600
−800
−1000

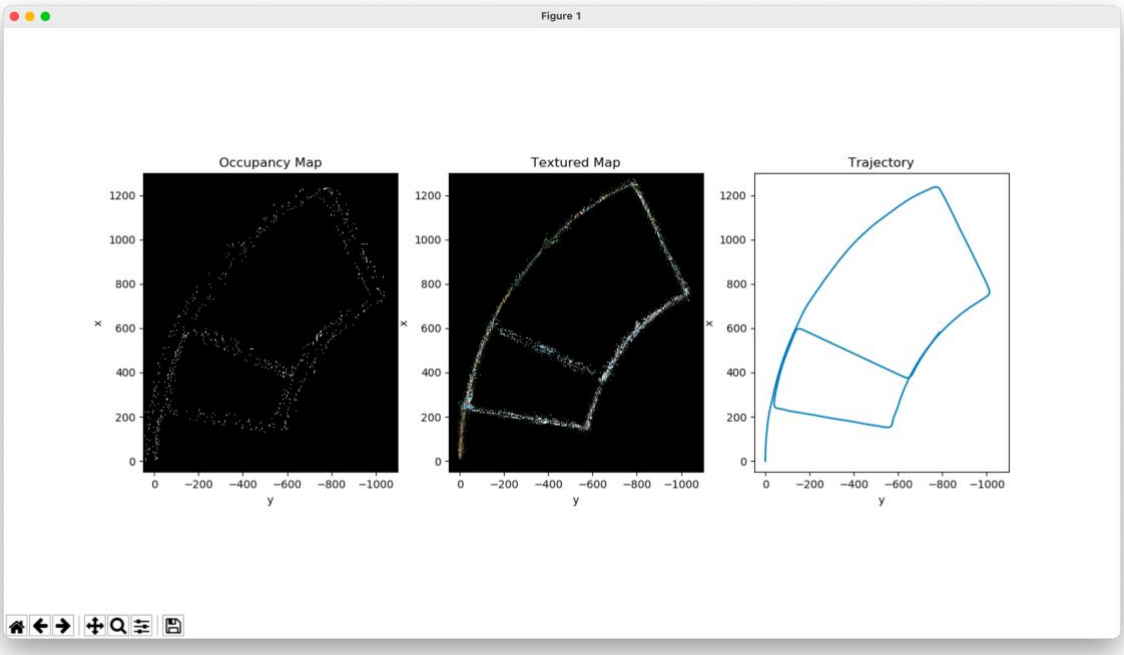
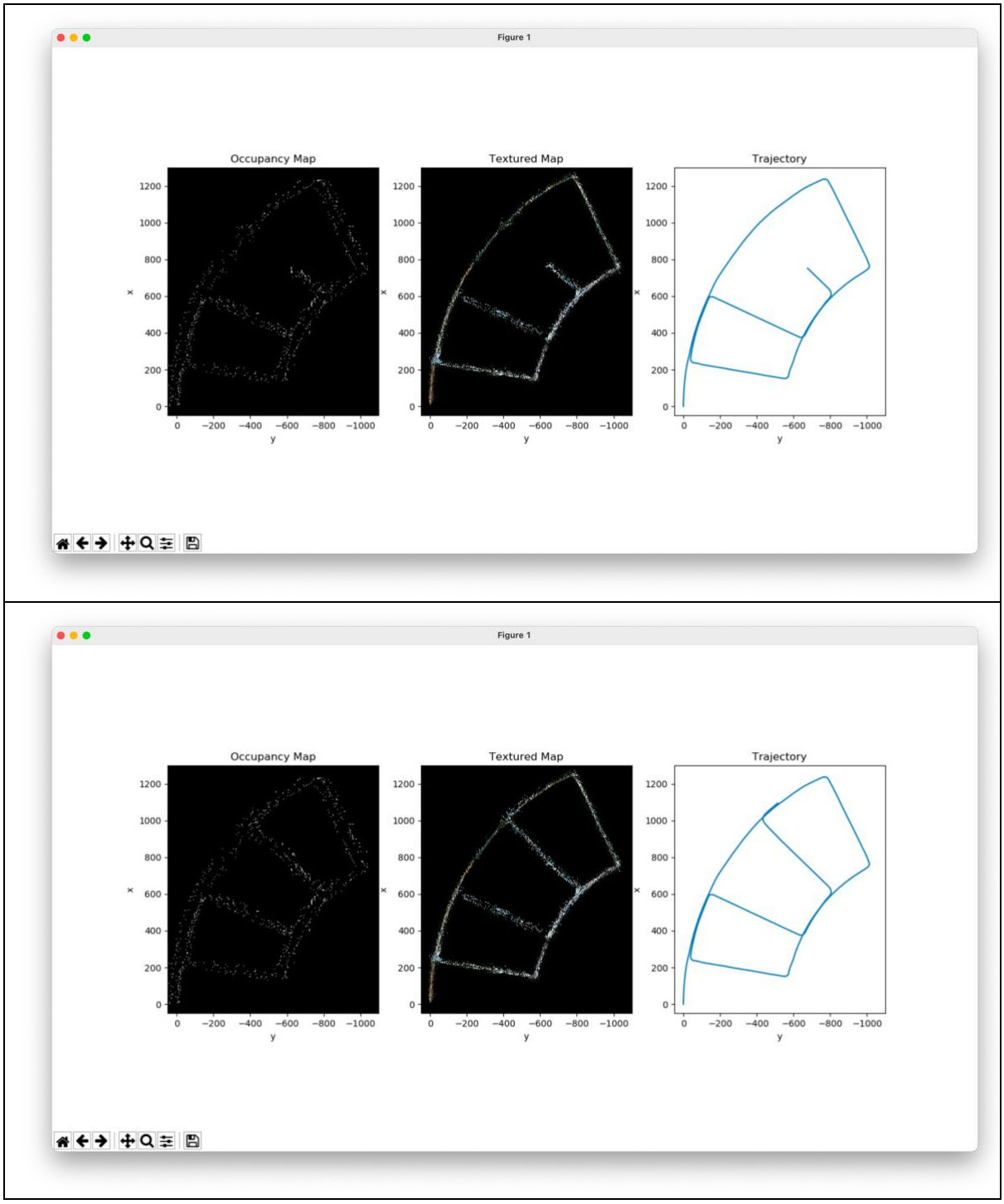


Figure 1
Occupancy Map
Textured Map
Trajectory
x
y
0
200
400
600
800
1000
1200
−200
−400
−600
−800
−1000



We can see that the maps and trajectory is gradually updated as more steps are taken.