PARTICLE FILTER

Introduction:

Simultaneous Localization and Mapping (SLAM) is an important topic within the mobile robotics community. The solutions to the SLAM problem has brought possibilities for a mobile robot to be placed at an unknown location in an unknown environment and simultaneously building a map and locating itself. As it means to make robots truly autonomous, the published SLAM algorithm are widely employed in self-driving cars, autonomous underwater vehicles and unmanned aerial vehicles. In this project, within sensing data provided by encoders, an inertial measurement unit and a horizontal LIDAR scanner in hands, we design a particle filter to perform the localization methods. Furthermore, we texture the map by using the provide data from a RGBD camera.

Problem Formulation:

I. Sensing data interpretation

a. Encoders

We use the formula below to calculate the meters per tick the encoder travels:

$$\text{meters per tick} = \frac{\pi \times (\text{wheel diameter})}{\text{ticks per revolution}}$$

b. LIDAR

The hit points provided by LIDAR in (x,y) coordinates can be described as below :

$$\begin{bmatrix} x_i \\ y_i \end{bmatrix} = (\text{current position}) + \begin{bmatrix} r_i \cos\theta(i) \\ r_i \sin\theta(i) \end{bmatrix}$$

where r_i is the measured range, $i = 1, 2, \ldots, N_r$, N_r is the number of the rays, and θ is a $1 \times N_r$ array constructed by an arithmetic series with a constant difference equal to the angle incrementation.

c. Kinect

To project the pixel values in 2D image coordinate frame to 3D world coordinate frame, the following translation and rotation is applied:

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = KR \begin{bmatrix} I & | -C \end{bmatrix} \begin{bmatrix} X_W \\ Y_W \\ Z_W \\ 1 \end{bmatrix}$$

where $[u\ v\ 1]^T$ is the homogeneous 2D image point, $[X_W\ Y_W\ Z_W\ 1]^T$ is the homogeneous 3D image point, K is the calibration matrix, R is the rotation matrix from camera coordinate to world coordinate frame, and C is the camera center in the world coordinate frame.

II. Motion model

The discrete-time model with time discretization τ can be described as:

$$p_f(s_{t+1} \mid s_t, \theta_t, \omega_t, v_t) = s_t + \tau \times \begin{bmatrix} v_t \text{sinc}(\frac{\omega_t \tau}{2}) \cos(\theta_t + \frac{\omega_t \tau}{2}) \\ v_t \text{sinc}(\frac{\omega_t \tau}{2}) \sin(\theta_t + \frac{\omega_t \tau}{2}) \end{bmatrix}$$

where ω_t is the rotational velocity given by IMU, v_t is the linear velocity obtained by encoders, θ_t is the current orientation, and s_t is the current position of the robot.

III. Observation model

Laser correlation model:

With the LIDAR hit points in hand, the observation probability can be written as choosing the maximum value of the sum of the correlation between the the hit points and a given range of the grid map:

$$p_h(z_{t+1} \mid s_t) = \arg\max_{i \in \mathrm{I}, j \in \mathrm{J}} \sum_{k=1}^{N_r} \text{corr}(\mathrm{y}_k, \mathrm{m}_{ij})$$

where I and J are the range of the the dimensional map that want to be evaluated, $\mathbf{y}_k$ is the LIDAR hit points, and $\mathbf{m}_{ij}$ is the grid map point.

IV. Particle filter

A mixture of delta functions (particles) and weights $\alpha^{(k)}$ are used to represent $p_{t|t}$ and $p_{t|t+1}$:

$$\delta(s;\mu^{(k)}) = \begin{cases} 1 & s = \mu^{(k)} \\ 0 & \text{else} \end{cases}$$

where $k = 1, 2, \ldots, \mathcal{N}$, and $\mathcal{N}$ is the number of the particles. Thus, the prior distribution $p_{t|t}$ can be written as:

$$p_{t|t}(s_t) = \sum_{k=1}^{N_{t|t}} \alpha_{t|t}^{(k)} \delta(s_t; \mu_{t|t}^{(k)})$$

a. Prediction step

$$p_{t+1|t}(s) = \sum_{k=1}^{N_{t+1|t}} \alpha_{t+1|t}^{(k)} \delta(s; \mu_{t+1|t}^{(k)})$$

In this step, bootstrap approximation is used by repeating the following procedure $\mathcal{N}_{t+1|t}$ times to obtain $\mu_{t+1|t}{}^{(k)}$

- randomly draw $k \in \{1, 2, \ldots, \mathcal{N}_{t|t}\}$ with probability $\alpha_{t|t}{}^{(k)}$
- the predicted position for each particle is updated by the motion model f_f with the noisily linear velocity v_t and the noisily rotational velocity ω_t

b. Update step

In this step, particle wights $\alpha_{t+1|t+1}{}^{(k)}$ are updated by using the information we have in the prediction step and the measurement probability. The equation is shown below:

$$\begin{aligned} p_{t+1|t+1}(s) &= \sum_{k=1}^{N_{t+1|t+1}} \alpha_{t+1|t+1}^{(k)} \delta(s; \mu_{t+1|t+1}^{(k)}) \\ &= \sum_{k=1}^{N_{t+1|t}} \frac{\alpha_{t+1|t}^{(k)} p_h(z_{t+1} \mid \mu_{t+1|t}^{(k)})}{\sum_{j=1}^{N_{t+1|t}} \alpha_{t+1|t}^{(j)} p_h(z_{t+1} \mid \mu_{t+1|t}^{(j)})} \delta(s; \mu_{t+1|t}^{(k)}) \end{aligned}$$

V. Occupancy grid mapping

A grid of the map log-odd $\lambda_{i,t+1}$ is used to maintain the occupancy of the grid map, where i is the observed cell in the map. The updated equation of $\lambda_{i,t+1}$ can be written as:

$$\lambda_{i,t+1} = \lambda_{i,t} + log\ g_h(z_{t+1} | m_i, s_{t+1})$$

where

$$g_h(z_t | m_i, s_t) = \frac{p_h(z_t | m_i = 1, s_t)}{p_h(z_t | m_i = 0, s_t)}$$

Technical Approach:

I. Signal processing

a. IMU denoising

We only use the yaw rate data from the IMU. Considering the data was collected from a moving robot with lots of high frequency vibrations, we apply a butterworth filter and a forward-backward filter to reject noise.

b. Encoders

The data sheet has indicated the following specification:

- wheel diameter: 0.254m
- ticks per revolution: 360
- encoder frequency: 40Hz
- encoder counts: [FR, FL, RR, RL] corresponding to the four wheels

Thus, by the formula below,

$$\text{meters per tick} = \frac{\pi \times (\text{wheel diameter})}{\text{ticks per revolution}}$$

, we know that the wheel travels 0.0022 meter per tic. Then the velocity of the differential-drive model can be derived as

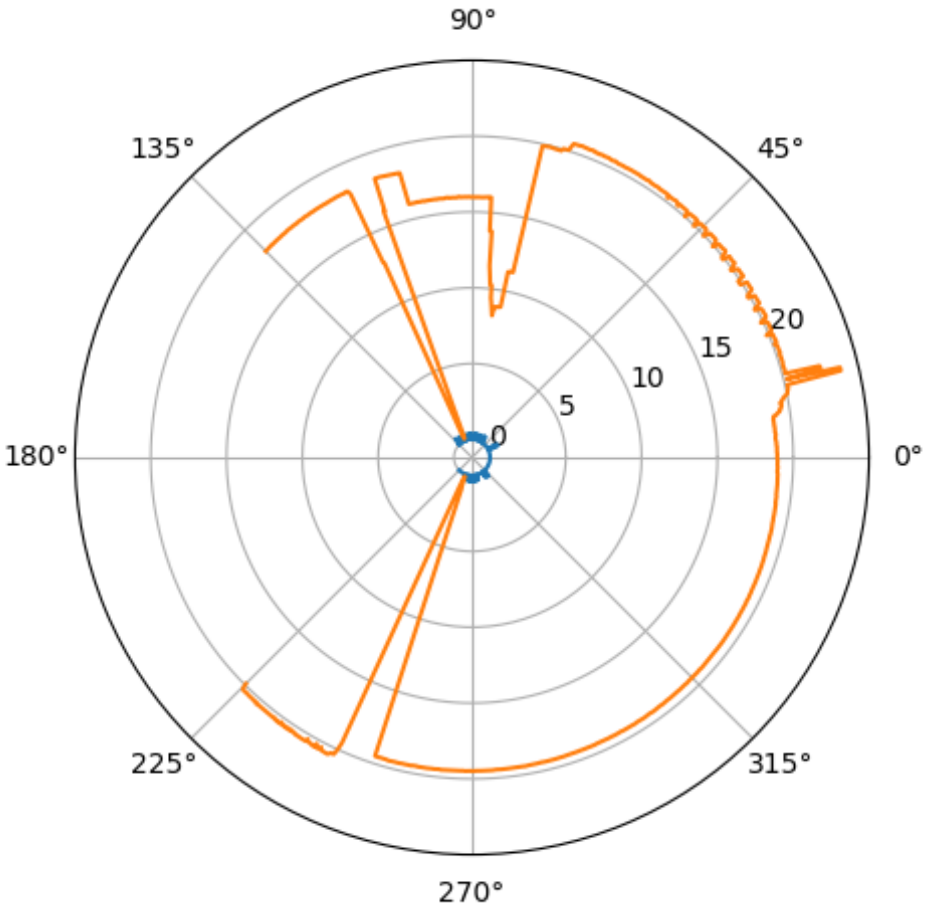
$$v = \frac{(FR + FL + RR + RL)}{4} \times 0.0022 \times \frac{1}{40}\ (\text{meters per second})$$

c. LIDAR transformation

The data sheet has indicated the following specification:

- field of view: 270 degree (counterclockwise from -135 to 135 degrees)
- maximum range value: 30m
- minimum range value: 0.1m
- angle incrementation between 1081 measured rays: 0.00436332

First, we filter out the sensing data for which ranges are out of the legal interval [0.1,30] provided by data sheet.
Second, we have observed that some of the rays are always blocked out by obstacles. The phenomenon can be obtained by the figure below:



In the figure above, blue line shows the minimum of the range of the rays and line in orange shows the maximum of the range of the rays. It is obvious that some of the rays' maximum value and minimum value are pretty close. Thus, we do not take those information into account so filter those ranges out.

Thus, with the provided orientation information and measured range from the rays (may less than 1081 since we reject the ray information falls out of the legal interval), we can construct the legal hit points in (x,y) coordinates by the formula below :

$$\begin{bmatrix} x_i \\ y_i \end{bmatrix} = s_t + \begin{bmatrix} r_i \cos(\omega_t + \theta(i)) \\ r_i \sin(\omega_t + \theta(i)) \end{bmatrix}$$

where r_i is the measured range, $i = 1, 2, \ldots, 1081$ and θ is a 1081 long array constructed by an arithmetic series with a constant difference equal to the angle incrementation.

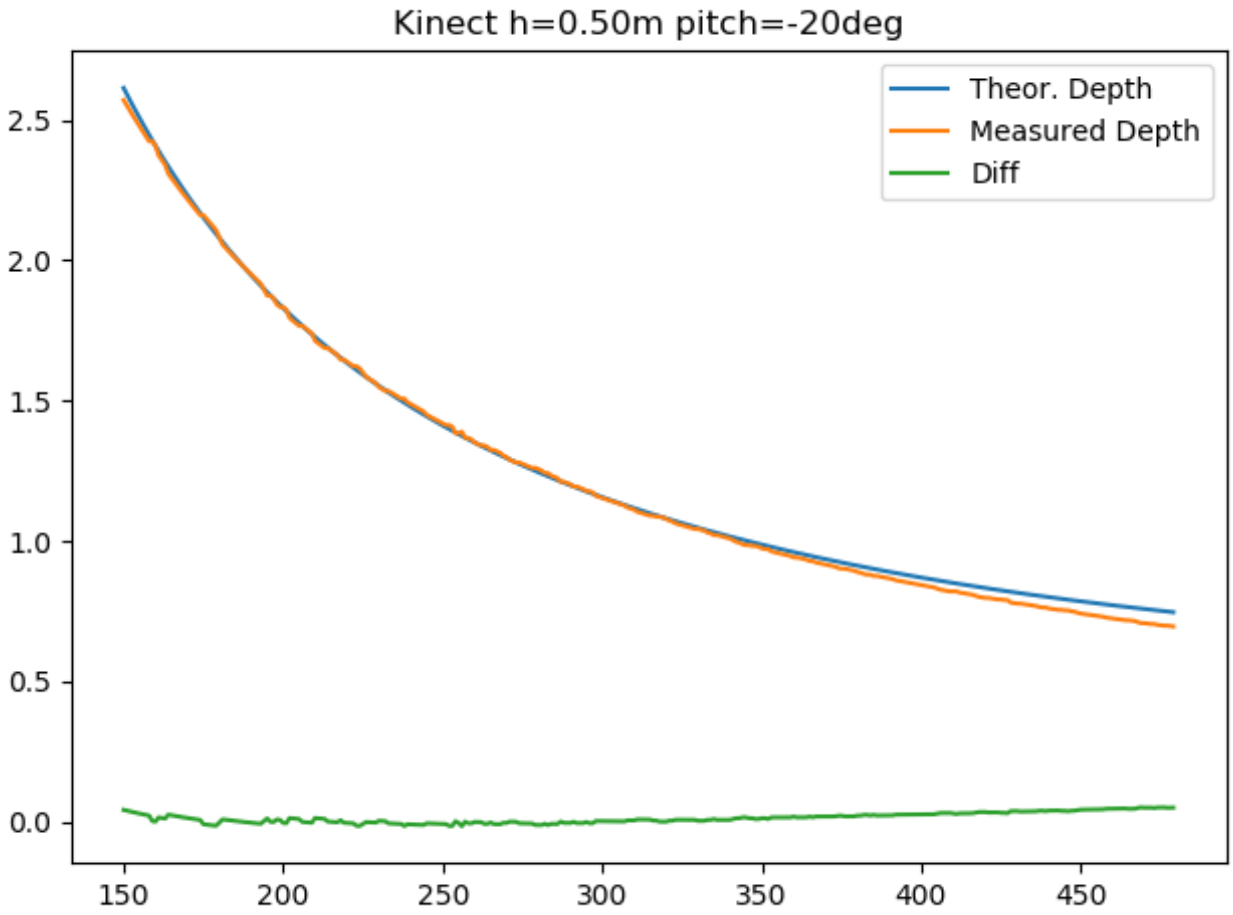
$$\begin{aligned} \theta &= [a_1 \quad a_2 \quad \ldots \quad a_{1081}]_{1\times 1081} \\ a_1 &= -135 \\ a_{1081} &= 135 \\ a_i &= a_1 + (i-1) \times 0.00436332 \\ i &= 1,2, ..., 1081 \end{aligned}$$

Last but not the least, since the LIAR in mounted on the top of the robot, a translation from LIDAR frame to the robot body frame is needed. It is shown in the data sheet that LIDAR is located 0.13673m in front of the geometric center of the robot, therefore, the legal hit points in the robot frame can be written as:

$$\begin{bmatrix} x_i \\ y_i \end{bmatrix} = s_t + \begin{bmatrix} r_i \cos(\omega_t + \theta(i)) \\ r_i \sin(\omega_t + \theta(i)) \end{bmatrix} + 0.13673 \times \begin{bmatrix} r_i \cos(\omega_t) \\ r_i \sin(\omega_t) \end{bmatrix}$$

d. Kinect

To ensure that the calibration matrix and camera rotation matrix is correct, we choose a certain area with only ground in the image and calculate the theoretical depth obtained by the calibration matrix and the rotation matrix, then compare the difference between the theoretical depth and the measured depth. The result is shown as the figure below:



e. Time step

Given that we have the encoders, LIDAR sampling rate as 40Hz, and IMU sampling rate around100Hz. We tend to choose 40Hz as our marching time step. Since the acquisition times of each measurement is different, we use **numpy.interp** to evaluate the measurement at each of out defined time step with given discrete data points (acquisition times, measurement value).

II. Localization prediction

We implement a particle filter with 100 particles. Each particle starts with an uniform weight as

$$\frac{1}{100}.$$

As time marching, to perform the differences between each particle, the given yaw rate from IMU and the given linear velocity from the encoders are added by additive Gaussian noise $n \sim G(0,0.2)$. Then the next position and orientation can be further predicted by the motion model. Moreover, the laser correlation model is used to adjust the location of the predicted position by choosing the grid with the maximum correlation value in the 9×9 map around the predicted position.

III. Localization update

As the measurement probability is required in the update step, we use the laser correlation model to calculate the correlation in a 9×9 map around the current position, then choose the one with the largest correlation value as the measurement probability.

IV. Mapping

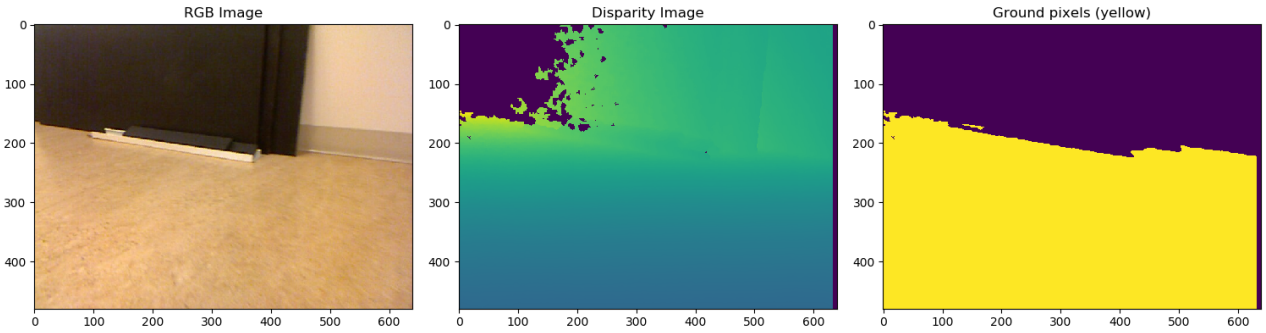
We use a 50×50 grid map with grid length equal to 0.1 for occupancy grid mapping, and a simple model for p_h is applied to determine g_h. In this case, ph is specified as how much we trust the occupancy measurement of cell i:

$$g_h(1 \mid m_i, s_t) = \frac{p_h(z_t = 1 \mid m_i = 1, s_t)}{p_h(z_t = 1 \mid m_i = 0, s_t)} = \frac{80\,\%}{20\,\%} = 4$$
$$g_h(0 \mid m_i, s_t) = \frac{1}{4}$$

Since $\lambda_{i,t+1}$ is updated while time marching, we need to take the constraints of $\lambda_{i,t+1}$ into account to avoid overconfident estimation. In this case, we clip $\lambda_{i,t+1}$ by the range (-10, 25) to maintain the quality of the grid map.

V. Texture mapping

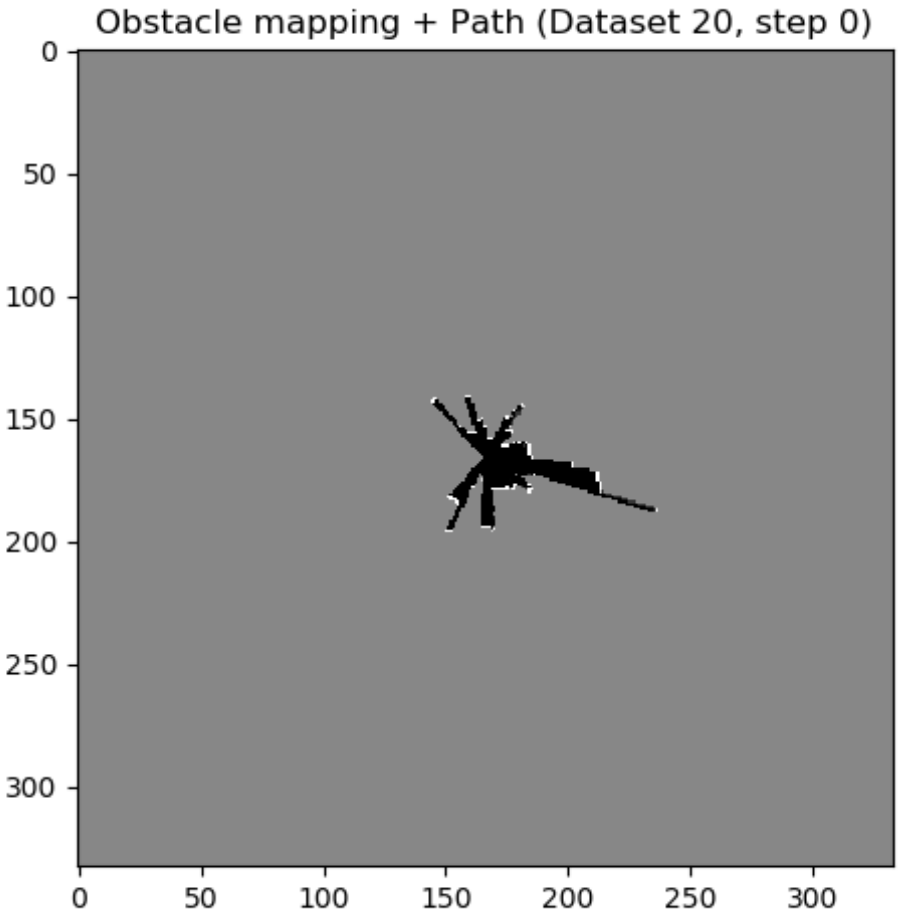
The figure below shows the ground pixels we obtained map to the RGB image.

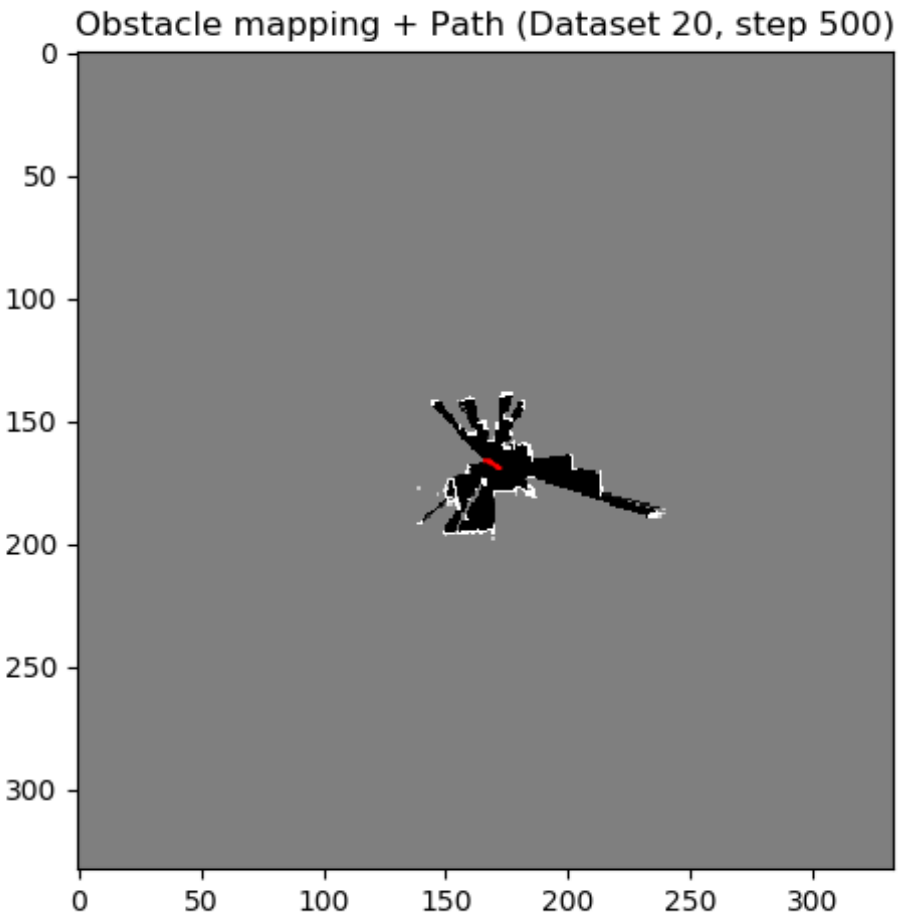


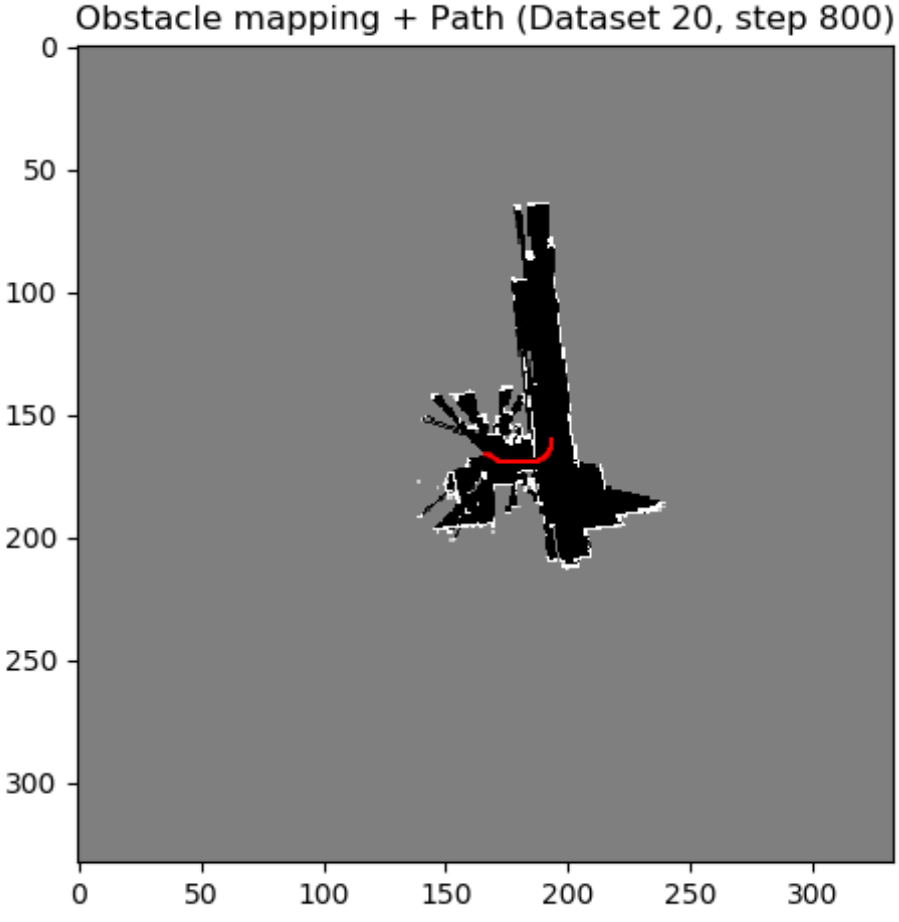
Results:

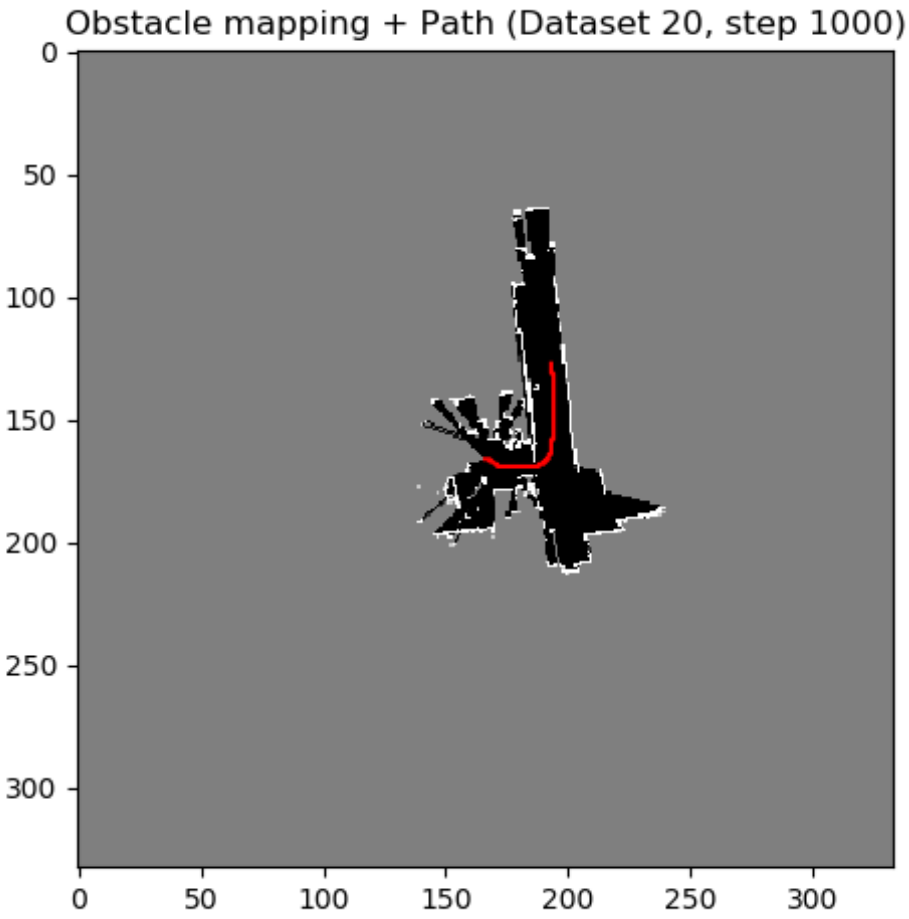
I. Robot trajectory

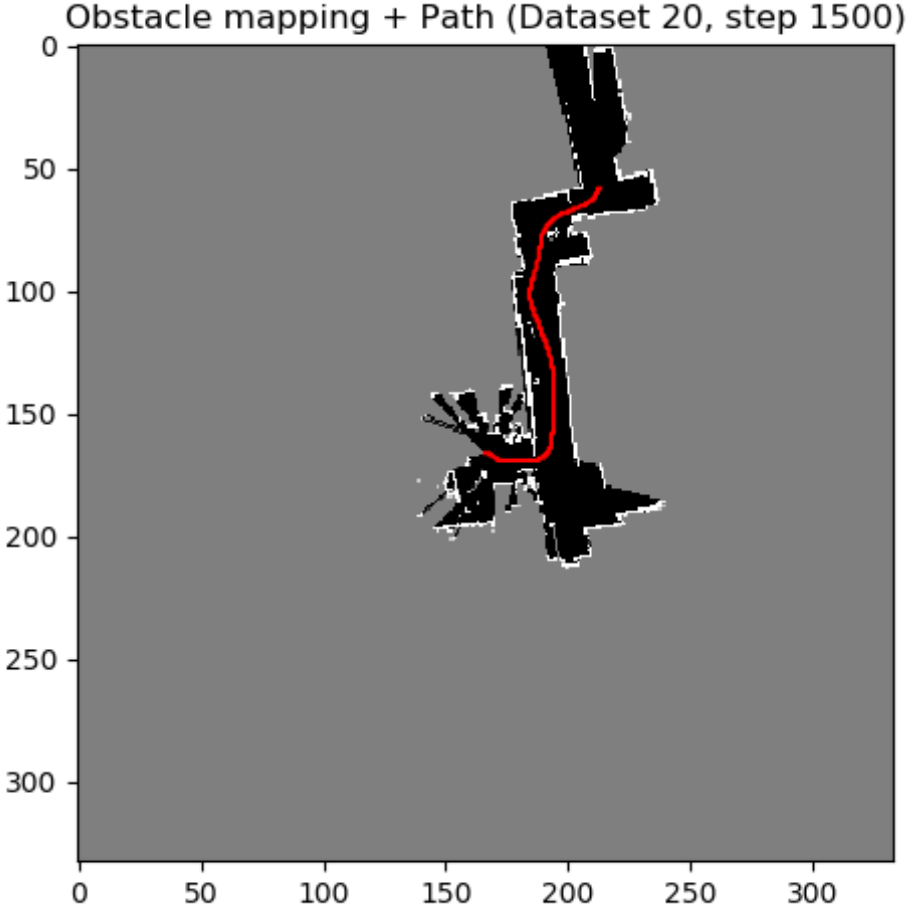
a. Test set 20 over time

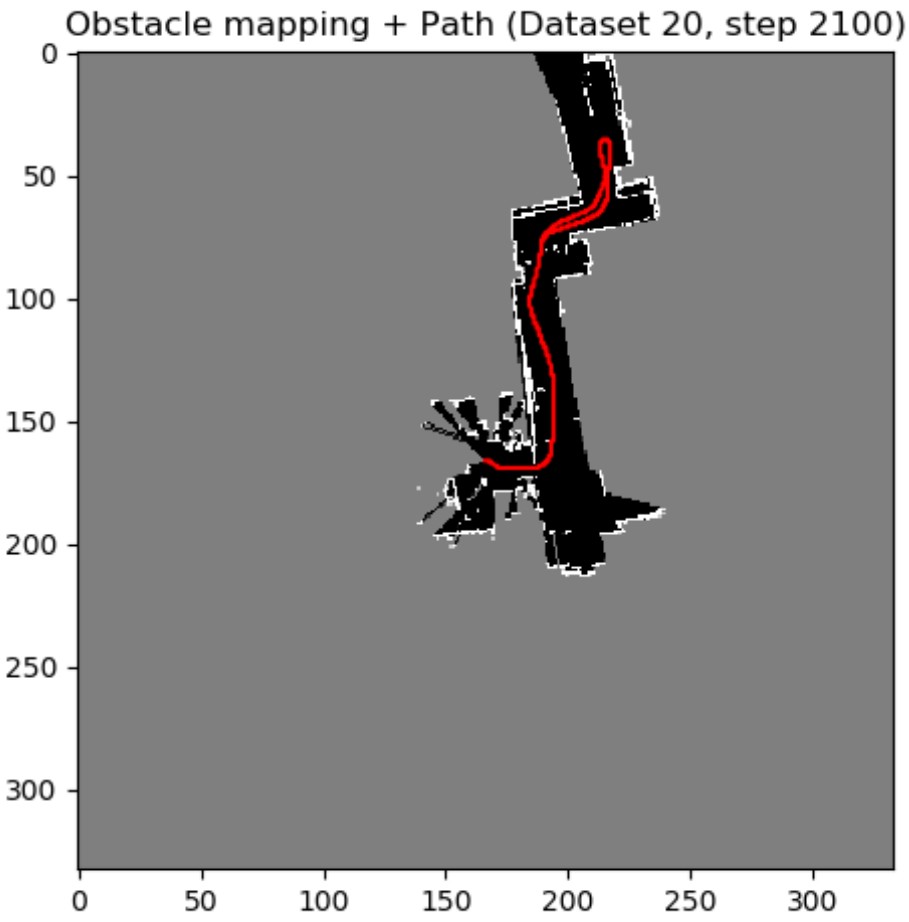


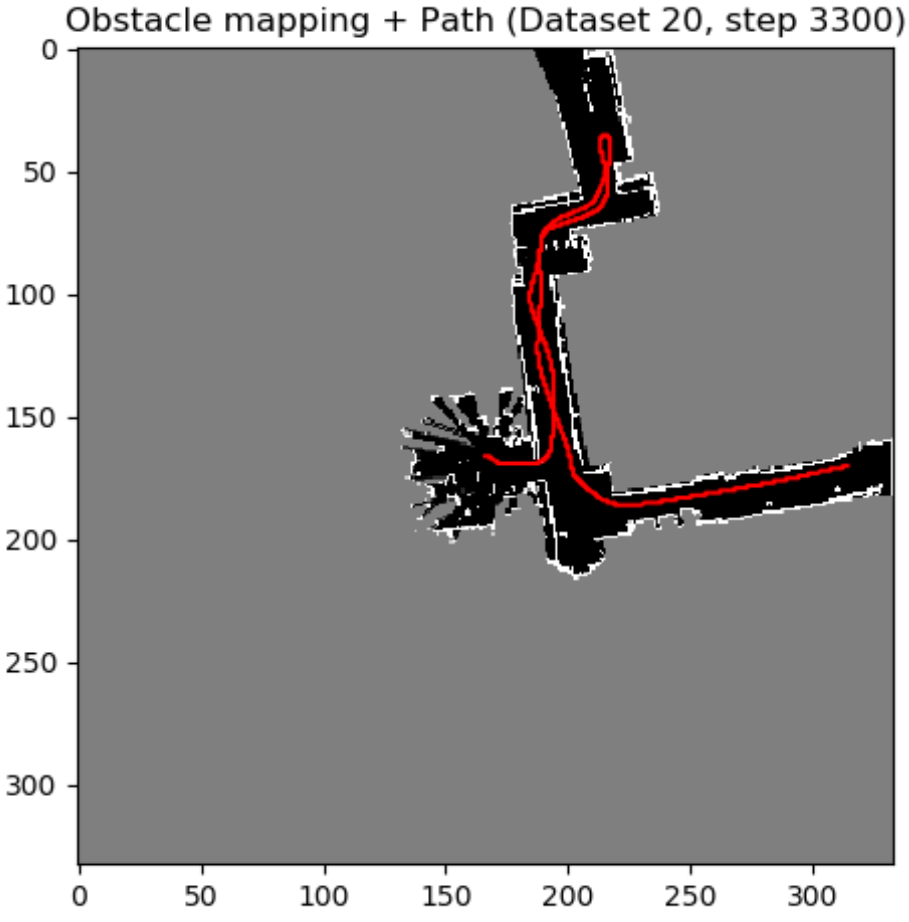

Obstacle mapping + Path (Dataset 20, step 500)
0
50
100
150
200
250
300
0
50
100
150
200
250
300

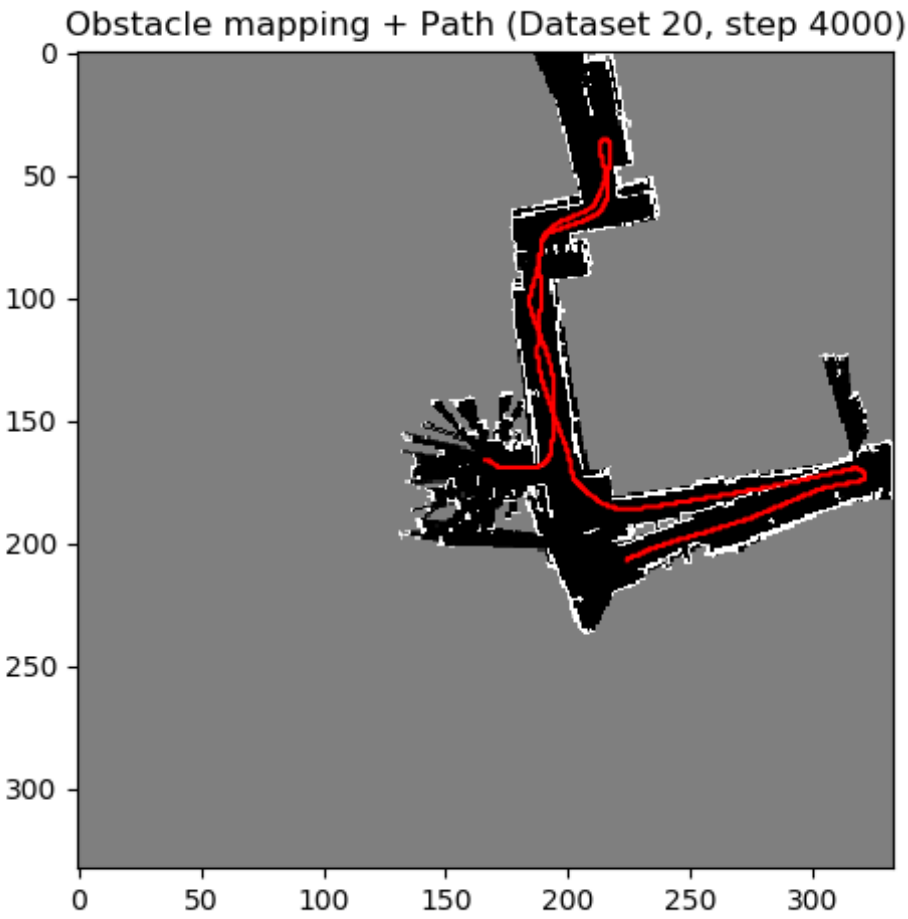

Obstacle mapping + Path (Dataset 20, step 800)
0
50
100
150
200
250
300
0
50
100
150
200
250
300

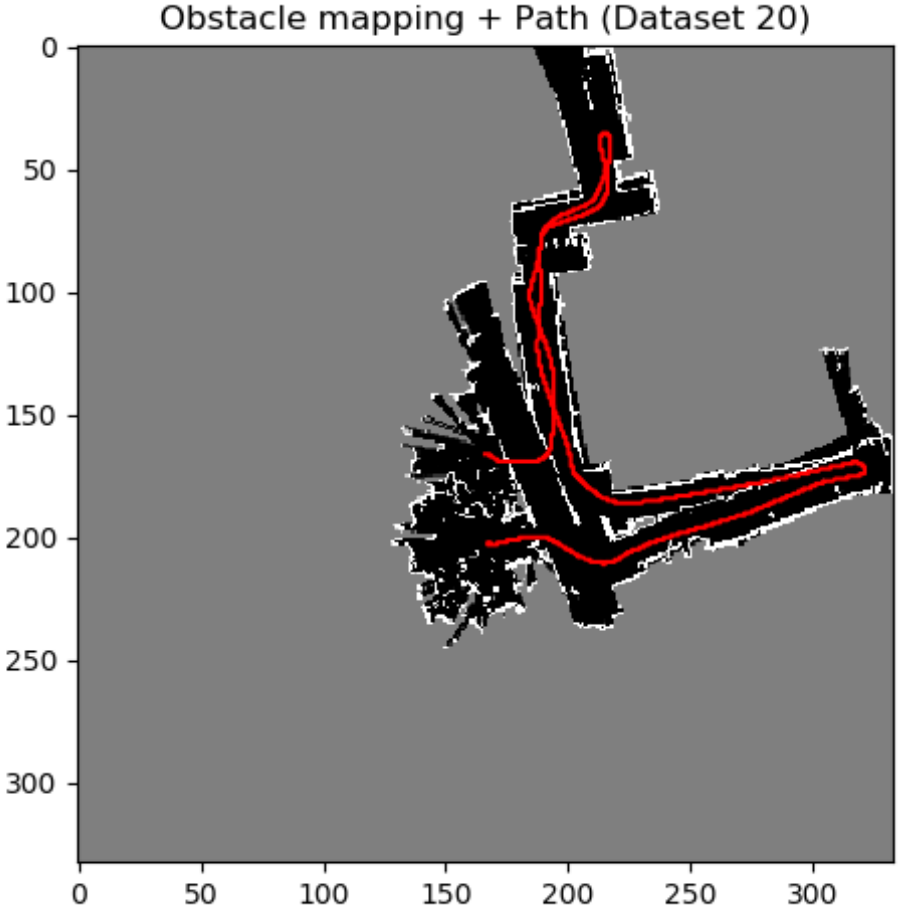

Obstacle mapping + Path (Dataset 20, step 1000)
0
50
100
150
200
250
300
0
50
100
150
200
250
300


Obstacle mapping + Path (Dataset 20, step 1500)
0
50
100
150
200
250
300
0
50
100
150
200
250
300

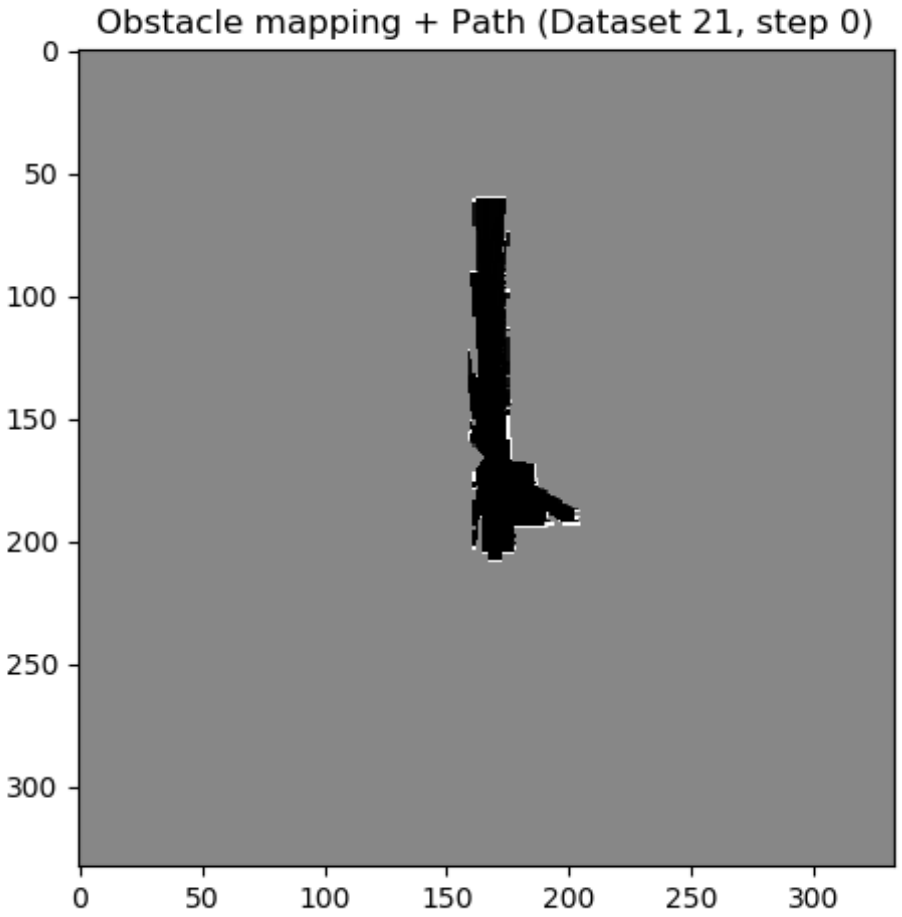

Obstacle mapping + Path (Dataset 20, step 2100)
0
50
100
150
200
250
300
0
50
100
150
200
250
300

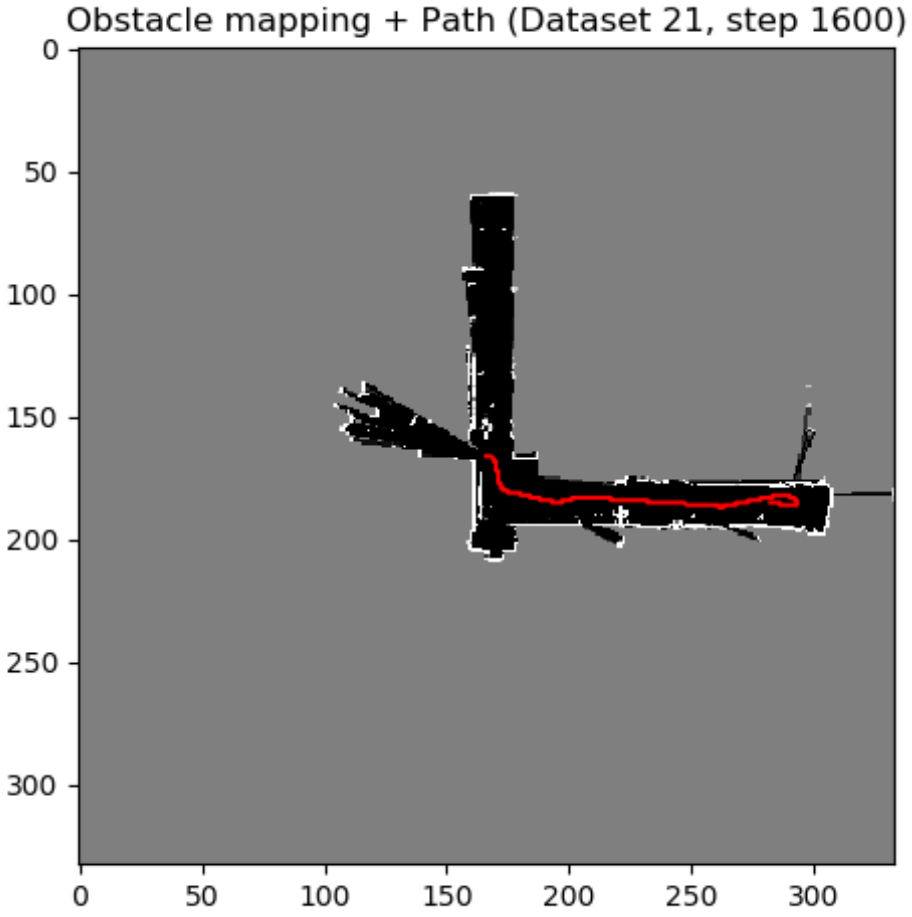

Obstacle mapping + Path (Dataset 20, step 3300)
0
50
100
150
200
250
300
0
50
100
150
200
250
300

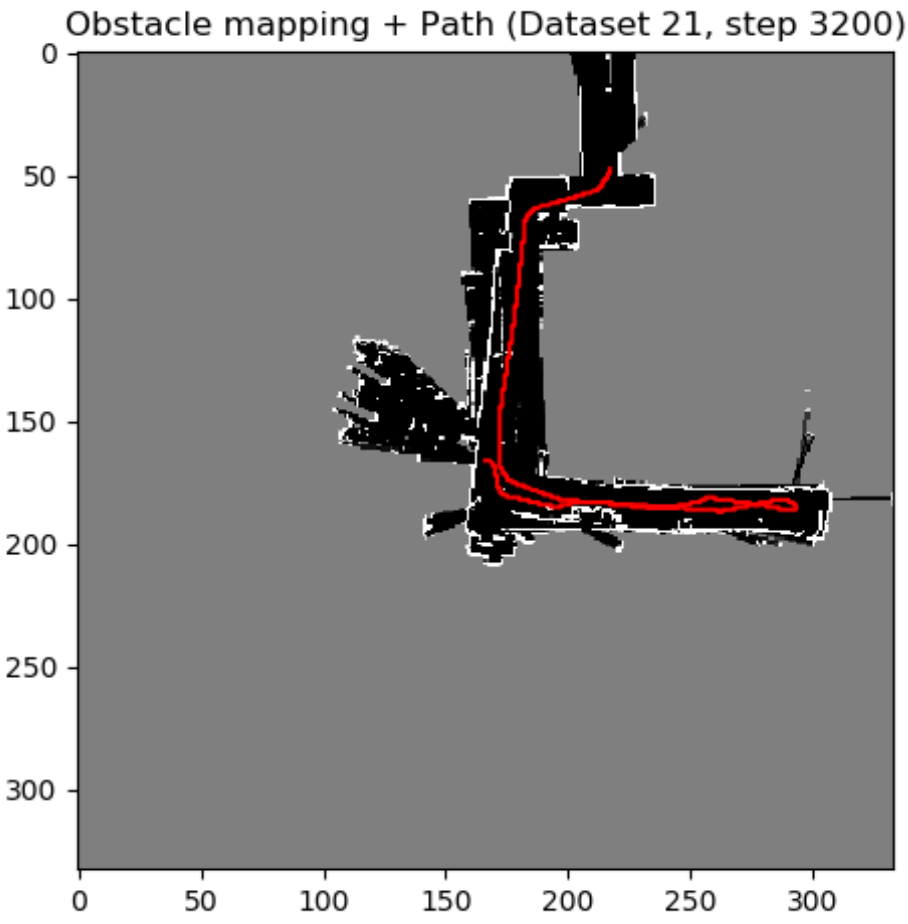

Obstacle mapping + Path (Dataset 20, step 4000)
0
50
100
150
200
250
300
0
50
100
150
200
250
300

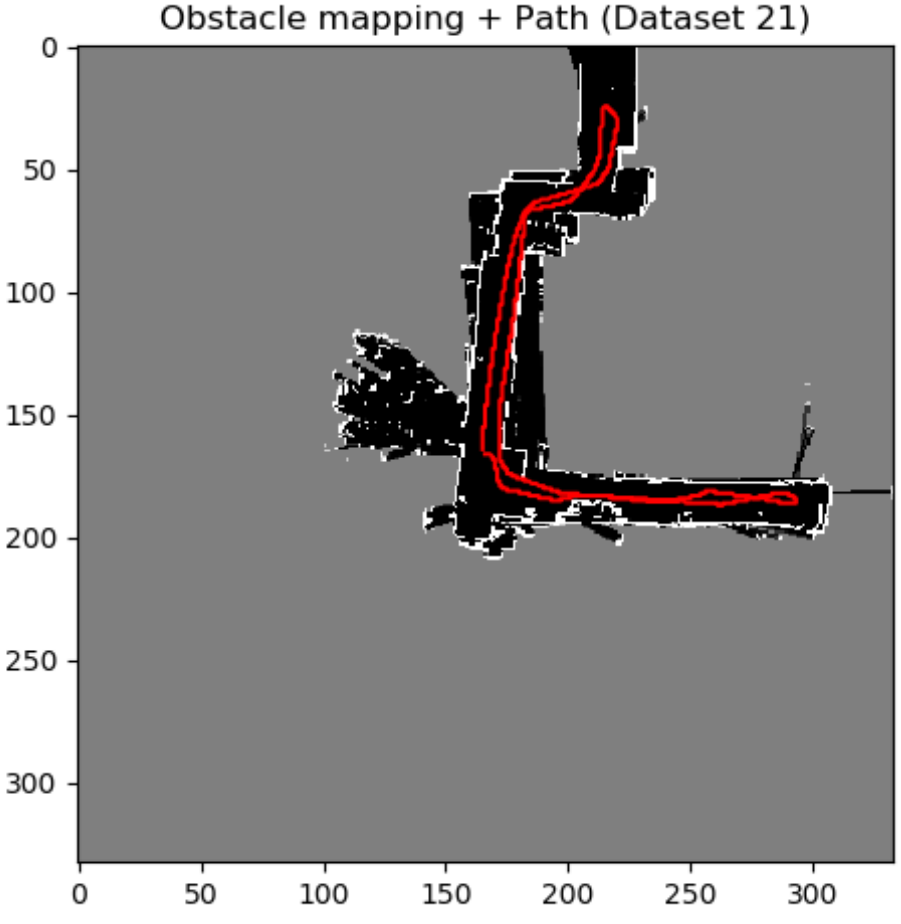

Obstacle mapping + Path (Dataset 20)
0
50
100
150
200
250
300
0
50
100
150
200
250
300

b. Test set 21 over time

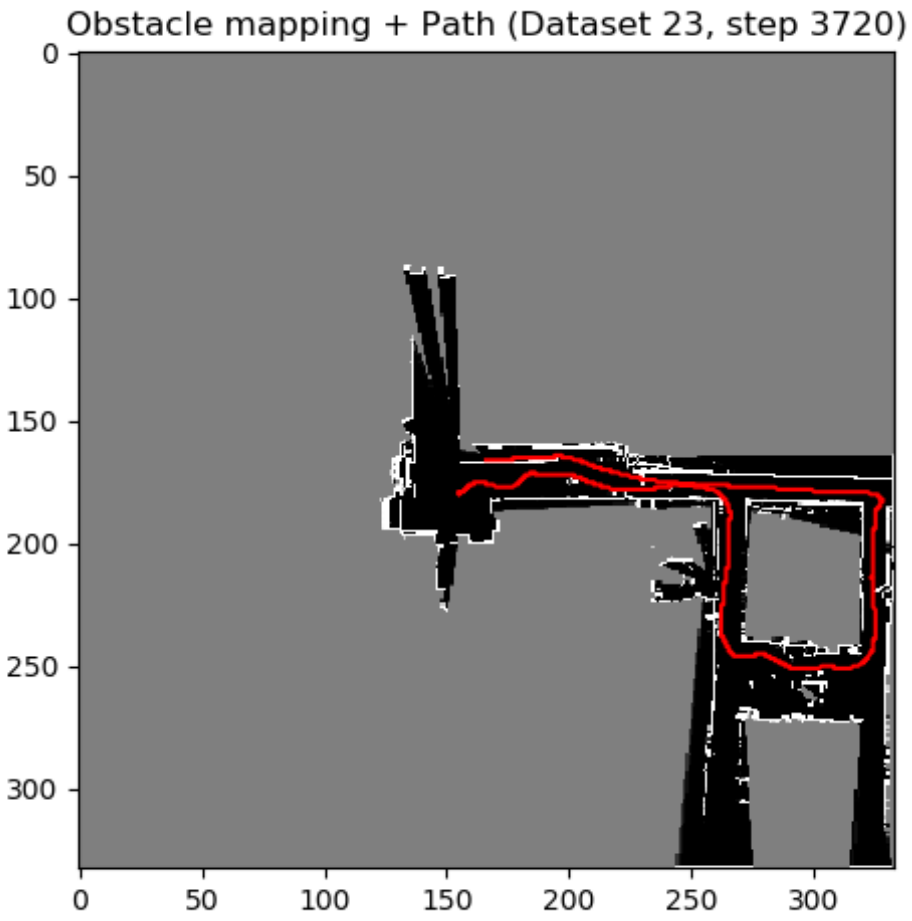





Obstacle mapping + Path (Dataset 21, step 3200)
0
50
100
150
200
250
300
0
50
100
150
200
250
300

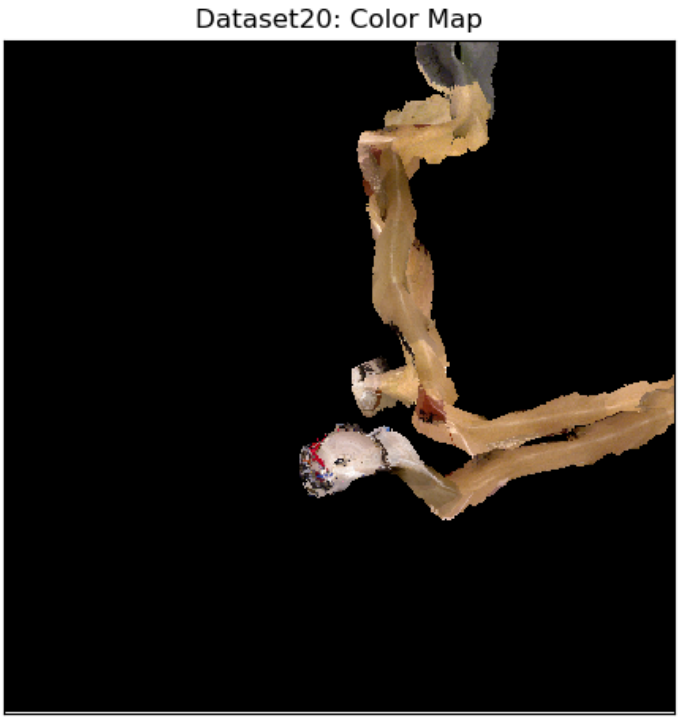

Obstacle mapping + Path (Dataset 21)
0
50
100
150
200
250
300
0
50
100
150
200
250
300

c. Test set 23


Obstacle mapping + Path (Dataset 23, step 3720)
0
50
100
150
200
250
300
0
50
100
150
200
250
300

II. Texture maps

a. Test set 20



b. Test set 21