Project 2: Particle Filter SLAM

Prathamesh Saraf (A59015739)
Department of Electrical and Computer Engineering
University of California, San Diego
psaraf@ucsd.edu

***Abstract*—This project implements SLAM on a differential-drive robot using encoder and IMU odometry, 2-D LiDAR scans, and RGBD measurements to construct a 2-D occupancy grid map of the environment. The mapping, prediction, update, and texture map phases are involved. Dead-reckoning is used to verify the prediction step, and the particle filter prediction step is implemented. The texture map is created by projecting colored points from the depth camera frame to the world frame and finding the plane that corresponds to the occupancy grid.**

***Index Terms*—Particle Filter - Simultaneous Localization and Mapping (SLAM), Differential-Drive Robot, IMU, 2-D LiDAR, RGBD Camera**

I. Introduction

Simultaneous Localization and Mapping (SLAM) is a crucial problem in robotics that allows robots to navigate and map their environment simultaneously. The goal is for the robot to determine its location and orientation while building a map of the environment using various sensors. This project aims to implement SLAM using the encoder and IMU odometry, 2-D LiDAR scans, and RGBD measurements on a differential-drive robot. The primary objective is to create a 2-D occupancy grid map of the environment while localizing the robot. Additionally, the RGBD images are used to assign colors to the map's floor surface.

The project is divided into four main phases: mapping, prediction, update, and texture mapping. In the mapping phase, the robot's identity pose is assumed, and the first LiDAR scan is used to create an occupancy grid map. The map is created by processing the LiDAR scan, removing scan points that are too close or too far from the robot, and transforming the LiDAR points from the LiDAR frame to the world frame. The occupied and free cells in the occupancy-grid map are then obtained using Bresenham2D.

In the prediction phase, the robot's motion is predicted using linear velocity and yaw rate from encoders and IMU. To verify the prediction step, dead-reckoning is implemented, and the robot's trajectory is plotted. A complete 2-D occupancy-grid map is then created by combining dead-reckoning and mapping before implementing the particle filter. The update phase uses the scan grid correlation to correct the robot's pose, and the final step involves using RGBD images to produce a 2D color map of the floor surface.

II. Problem Formulation

A. Particle Filter

The particle filter PDF is given below. It uses Bayes filtering for estimating the dynamical system by using the inputs and observations. Alpha is the confidence value of each particle:

$$p_{t|t}\left(x_t \mid z_{0:t}, u_{0:t-1}, m\right) = \sum_{k=1}^{N} \alpha_{t|t}^{k} * \delta\left(x_t; \mu_{t|t}^{k}\right) \quad (1)$$

B. Prediction and Update

Given the predicted density p over x and the measurement z, we use the observation model ph to incorporate the measurement information and obtain the posterior p over x. The prediction and update equations are given below:

$$p_{t+1|t}(x) = \int p_f\left(x \mid s, u_t\right) * p_{t|t}(s) ds \quad (2)$$

$$p_{t+1|t+1}(\mathbf{x}) = \frac{p_h\left(\mathbf{z}_{t+1} \mid \mathbf{x}\right) p_{t+1|t}(\mathbf{x})}{\int p_h\left(\mathbf{z}_{t+1} \mid \mathbf{s}\right) p_{t+1|t}(\mathbf{s}) d\mathbf{s}} \quad (3)$$

C. Mapping

Mapping refers to the process of creating a map of the environment using sensor observations, which may be noisy and uncertain, along with the robot's trajectory. One commonly used method for mapping is occupancy grid mapping, which involves dividing the environment into a grid of cells and assigning a binary value to each cell to indicate whether it is occupied by an obstacle or not.

The occupancy grid mapping algorithm assumes that the values assigned to each cell are independent of each other, given the robot's trajectory. In other words, the occupancy state of each cell is only influenced by the robot's position at that point in time, and not by the occupancy state of other cells or previous time steps.

While this assumption simplifies the mapping process, it may not always hold true in practice, especially in environments with complex and interdependent structures. Nonetheless, occupancy grid mapping remains a popular and effective approach for mapping in many applications.

$$p\left(m \mid z_{0:t}, x_{0:t}\right) = \prod_{i=1}^{n} p\left(m_i \mid z_{0:t}, x_{0:t}\right) \quad (4)$$

D. Lidar scan

The LiDAR scans data between -135 degrees to 135 degrees. By iterating over this scan and using the lidar ranges data provided, we create an occupancy grid map of every timestep. However, the lidar can only detect distances up to 30 meters. Consequently, any range values in L that exceed this limit are deemed inaccurate and should be excluded from the analysis.

The x and y coordinates of the LiDAR scan are calculated as given below:

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} r\cos(\theta) \\ r\sin(\theta) \\ 0 \end{bmatrix} \tag{5}$$

These coordinates are however in LiDAR frame. They can be converted into the robot body frame by translating using the robot measurements. Thus the LiDAR scan data is converted into the body frame using:

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} r\cos(\theta) + 0.13673 \\ r\sin(\theta) \\ 0 \end{bmatrix} \tag{6}$$

These values are in the spherical frame which is converted into a cartesian frame using the motion model.

E. IMU and Encoder Data

For the project, we utilized only the yaw rate data obtained from the IMU sensor. However, as the data contained noise, it required filtering. To accomplish this, we employed the low pass filter method, using a frequency of 10Hz. The corresponding filtering transfer function is provided below and the equation is mentioned in the code:

$$T(s) = \frac{K}{1 + \left(\frac{s}{\omega_O}\right)} \tag{7}$$

Secondly, it was necessary to synchronize the IMU data frequency with the LiDAR scan frequency. To achieve this, we isolated only those timestamps from the IMU data that closely matched the LiDAR scan data. The filtered and truncated IMU data is plotted and presented in figure (1) Next, we use the encoder data to calculate the velocity of the robot. The formula to do the same is given below:

$$vR = (FR + RR)/2 \times 0.0022m \tag{8}$$

$$vL = (FL + RL)/2 \times 0.0022m \tag{9}$$

The differential drive model is given in figure (2) The left (vL) and right (vR) wheel velocity calculation use respective encoder ticks and using the calibration factor 0.0022m (linear translation of the wheel at each tick) we calculate the left and right velocity of the robot. The final robot velocity is the average of the left and right side velocities.

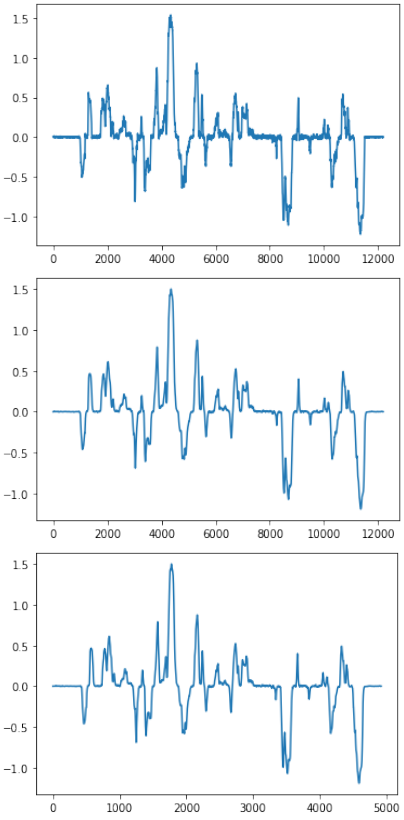

Fig. 1. Yaw rate for dataset 20. (1) Unfiltered, (2) Filtered, (3) Truncated

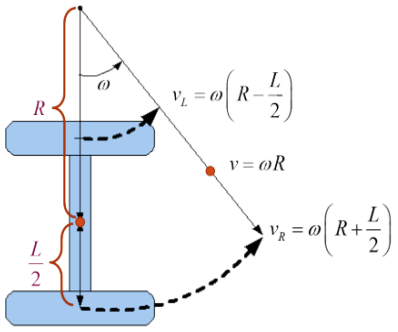

Fig. 2. Differential Drive model of the robot

III. Technical Approach

A. Data Analysis

The measurement datasets used in the project include the IMU, encoders, LiDAR, and Kinect cameras. The IMU provides data on the body's angular velocity (yaw) in the IMU reference frame, with a Unix timestamp recorded for each measurement. The encoders, which count the rotations of the four wheels, are sampled at 40 Hz. The LiDAR system employs a Hokuyo UTM-30LX sensor, which has a 270-degree field of view and can measure distances up to 30 meters. Each LiDAR scan consists of 1081 range values that provide measurements of obstacles in the environment. The Kinect camera captures both RGB and disparity images.

Due to the different sampling rates of the measurement datasets, synchronization was performed on all three datasets - LiDAR, IMU, and encoders - and matched to the Encoder timestamps. The Kinect camera was also synchronized accordingly. The IMU data was then imported and processed using a low pass filter with a cutoff frequency of 10Hz to eliminate any noise present in the yaw measurements. Finally, the linear velocity was calculated using a formula that was determined beforehand.

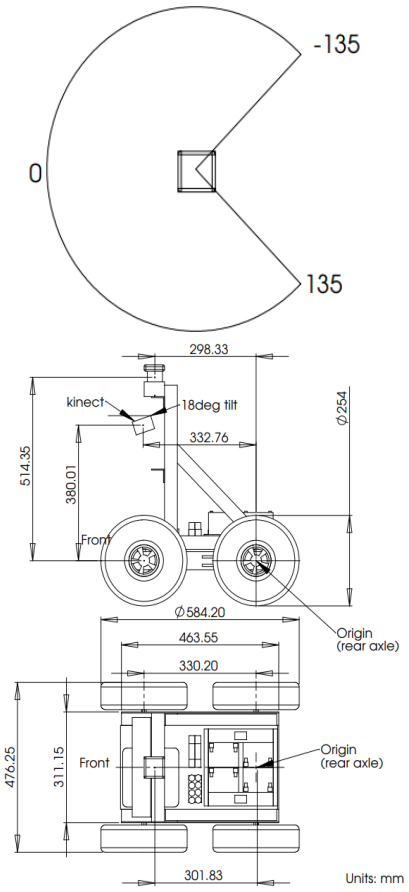

Fig. 3. Robot specifications

B. Motion Model

The differential drive robot motion model uses the steering angle (yaw angle), the yaw rate at each timestep, and the robot velocity to calculate the position and orientation of the robot at each timestep. The linear velocity (vt) data was collected from the encoders, and the yaw rate (t) data was collected from the IMU. The motion model equation is given below:

$$\mathbf{x}_{t+1} = \begin{bmatrix} x_{t+1} \\ y_{t+1} \\ \theta_{t+1} \end{bmatrix} = f_d\left(\mathbf{x}_t, \mathbf{u}_t\right) := \mathbf{x}_t + \tau_t \begin{bmatrix} v_t \cos\left(\theta_t\right) \\ v_t \sin\left(\theta_t\right) \\ \omega_t \end{bmatrix} \tag{10}$$

Since we are doing the SLAM operation over 100 particles, we add noise to the motion model so that each particle is slightly disturbed from the other. We add a gaussian noise with a mean of 0 and a variance of 0.005. I initially tried with variance = 0.05 but got a highly distorted map output. The noise of 0.005 gives a fairly well-constructed graph.

Dead reckoning is a useful step in the particle filter SLAM process, as it provides a rough estimate of the robot's trajectory and environment map. The first step in Dead Reckoning involves using the motion model to derive the robot's trajectory. This trajectory is then used to perform LiDAR data processing, which involves converting the LiDAR radial coordinates into Cartesian coordinates and transforming them from the LiDAR frame to the Robot frame, and then to the World frame. The transformation matrix used to convert body coordinates to world coordinates is given below:

$$T_i = \begin{bmatrix} R_i & X_i \\ 0 & 1 \end{bmatrix} \tag{11}$$

Once the robot's trajectory has been derived, the next step is to build an occupancy map. This is achieved by creating a grid with a specified size and resolution, which is matched to the real world. In this case, a grid size of 50 x 50 with a resolution of 0.01 m was chosen for better precision in the occupancy grid map. The cells which have been filled or identified as an obstacle from LiDAR ranges are then assigned values of 1, and empty cells are assigned 0 to build an occupancy map.

Although not necessary for particle filter SLAM, Dead Reckoning is a valuable step as it helps build intuition of the data and many of the steps involved in Dead Reckoning can be translated into the prediction step. By providing a rough estimate of the robot's trajectory and environment map, Dead Reckoning can also help improve the accuracy of the final map produced by particle filter SLAM.

C. Occupancy Grid

Mapping involves creating a map of the environment using sensor observations while accounting for uncertainty. One common approach is occupancy grid mapping, where the environment is discretized into cells and each cell is assigned a binary value indicating if it's occupied or free. The map is represented as a two-dimensional vector, with each entry indicating the occupancy status of the corresponding cell.

To model the occupancy status of each cell, we assume that the cell values are independent given the robot's trajectory. We use Bernoulli random variables with a uniform prior since we have no prior information about the occupancy status of the cells.

To update the occupancy grid map, we use lidar scans and Bresenham2D algorithm to convert the scan into map cells. The log-odds ratio is used to update the probability of each cell, which is based on the likelihood of the lidar scan given the occupancy status of the cells. The resulting probability in each cell indicates its occupancy status, and a threshold value can be set to determine whether a cell is considered occupied or free. An occupancy map is created by utilizing the previous map generated through the dead reckoning step. The maximum and minimum values of the robot's trajectory are analyzed to establish a new occupancy grid. This grid has a size of 20 * 20 m and a resolution of 0.5 m, resulting in a final grid map of 1200 * 1200.

To project LiDAR coordinates onto the grid, the coordinates in the world frame are first transformed into map coordinates. This allows for precise localization of the LiDAR data within the occupancy map.

In this approach, the map cells mi is regarded as independent Bernoulli random variables. When mi is occupied with probability it, it is represented as 1, while when it is free with

probability 1-it, it is represented as -1. To update the Odds ratio of this variable, the Bayes rule is applied, and the observation model odds ratio is computed.

For the log-odds occupancy grid mapping, a trust ratio of true to false positives is set at 4, and the mapping is calculated based on whether zt indicates that mi is occupied or free. For each observed cell I, the log odds decrease if the observation was free and increased if the observation was occupied. To avoid overconfident estimation, an upper and a lower bound is placed on i.

After the log-odds of cells are estimated, the occupancy probability (i,t) can be calculated from the log-odds value. Given below are the series of equations used to compute the log odds:

$$\begin{aligned} i &= \operatorname{ceil}\left(\left(x_w - x_{\min}\right) / \text{ resolution }\right) \\ j &= \operatorname{ceil}\left(\left(y_w - y_{\min}\right) / \text{ resolution }\right) \end{aligned} \tag{12}$$

$$\gamma_{\mathrm{i,t}} = \mathrm{p}\left(\mathrm{m_i} = 1 \mid \mathrm{z_{0:t}}, \mathrm{x_{0:t}}\right) \tag{13}$$

$$g_h\left(z_t \mid m_i, x_t\right) = \frac{p\left(m_i = 1 \mid z_t, x_t\right)}{p\left(m_i = -1 \mid z_t, x_t\right)} * \frac{p\left(m_i = 1\right)}{p\left(m_i = -1\right)} \tag{14}$$

$$\lambda_{i,t} = \log\left(\frac{p\left(m_i = 1 \mid z_t, x_t\right)}{p\left(m_i = -1 \mid z_t, x_t\right)}\right) - \lambda_{i,0} + \lambda_{i,t-1} \tag{15}$$

$$\Delta\lambda_{i,t} = \log\left(\frac{p\left(m_i = 1 \mid z_t, x_t\right)}{p\left(m_i = -1 \mid z_t, x_t\right)}\right) \pm \log(4) \tag{16}$$

$$\lambda_{i,t+1} = \lambda_{i,t} \pm \log(4) \tag{17}$$

$$\gamma_{i,t} = 1 - \frac{1}{1 + \exp\left(\lambda_{i,t}\right)} \tag{18}$$

D. Particle Filter

Particle Filter Localization is the process of estimating the position and orientation of a robot in an environment. The first step in the Particle Filter is initializing a set of particles. In this case, the number of particles is set to 100, and all particles are initialized to [0, 0, 0], which is the same as the initial location of the robot. Each particle is assigned an initial weight of 1/N, where N is the number of particles.

The next step involves updating the particles using LiDAR data. The LiDAR sensor provides measurements of distances to obstacles in the environment, and these measurements are used to update the particles. The LiDAR scan at time 0 is converted to world coordinates and projected onto an occupancy grid map, which is initially set to all zeros.

The prediction step is then performed, where the motion model of the robot is used to predict the next location of each particle. In this case, noise is added to the particles at each iteration of the motion model, which helps disperse the particles from each other and capture all possible states in the close vicinity of a probable solution. The standard deviation of the noise is set to 0.005 with a mean of 0.

Next, the Particle Update step is performed. The particle weights are updated based on the output of the map correlation function. This step involves extracting a correlation of the LiDAR scans in the world by transforming the position of each particle and then comparing it for overlap with the grid constructed in the previous iteration. The weights of the particles are stored as the correlation scores of the particles.

The weights of the particles are then normalized so that the sum of all weights is equal to 1. The particle with the highest weight is chosen, and its state becomes the estimated location of the robot. This is called the Robot State Estimation.

In the next step, the occupancy grid map is updated based on the LiDAR data. The log-odds technique is used to update the map. In this code, a log4 value is added to the cell for every obstacle detection observed and log2 is added for the ray connecting the robot location (particle location) to the obstacle. The grid locations where no obstacle tracing ray is encountered are kept at a value of 0.

These steps are iterated for all timestamps, and if the number of effective particles decreases below a threshold, particle resampling is carried out. Particle resampling involves selecting new particles based on the weights of the existing particles, with particles with higher weights being more likely to be selected.

Overall, the Particle Filter algorithm is a powerful method for localizing a robot in an environment. By using a set of particles to represent possible positions of the robot, the Particle Filter is able to handle uncertainties in the robot's motion and measurement data, and provide accurate estimates of the robot's location and orientation. The SLAM pseudo-code is given below:

Initialize:

- Occupancy log-odds
- Particles

for t in $0...T$ **do**

1) Transform lidar scan points z_t to body frame
2) **Particle filter: Update Step:**
   a) Transform lidar scan points to world frame with respect to the state (pose) of each particle to get $y_t^{(k)}$
   b) Calculate map correlation $corr^{(k)}(y_t^{(k)}, m_t)$
   c) Update particles $\mu_t|t$ to their local maximum map correlation
   d) Update particle weights $\alpha_t^{(k)}|t$
3) **if** $N_{eff} \leq N_{threshold}$ **then**
   a) **Particle Filter: Resampling**
4) **Occupancy Map: Update log-odds**
   a) Choose particle with maximum weight $\alpha_t|t$
   b) Transform lidar scan points to its world frame y_t
   c) Update log-odds
5) **Particle filter: Prediction Step**

end

E. Resampling

In order to prevent weight collapse due to weight disparity, resampling is employed. This process involves replacing particles with negligible weights with new particles in close proximity to those with higher weights. Resampling results in the creation of a new particle set with equal weights by adding more particles to high-weight locations and fewer particles to low-weight locations. This technique concentrates the representation power of the particles in regions with a high likelihood while leaving regions with low likelihood with fewer particles. Resampling is performed at time t when the effective number of particles falls below a certain threshold, represented by the following inequality:

$$N_{eff} = \frac{1}{\sum_{k=1}^{N} \left(\alpha_{t|t}^{(k)}\right)^2} \leq N_{\text{threshold}} \tag{19}$$

F. Texture Mapping

To begin the texture mapping process, it is necessary to calibrate and undistort the color and depth images obtained from the Kinect. This is important because the RGBD data provided by the Kinect is not accurately calibrated. The intrinsic matrix and distortion coefficients for both the color and depth images are provided in the dataset. Our implementation involves using the cv2.undistort function and the corresponding distortion coefficients to undistort the images.

Next, we align the color and depth images and generate a point cloud in the Kinect frame. This process requires the intrinsic matrix and extrinsic matrix provided in the dataset. The resulting output is a matrix P R N6, where each row contains the x, y, and z coordinates of a pixel in the aligned image, along with its corresponding RGB color information.

Once the point cloud in the Kinect frame is retrieved, we transform the coordinates of the points to the world frame and map them onto the occupancy grid map.

IV. Results

This section presents the plots for:

1) The Occupancy grid for dataset 20 and 21
2) The texture map for datasets 20 and 21

The results of dead reckoning involve the trajectory of the robot obtained from a motion model. To predict the robot's location, the motion model was used and LiDAR coordinates were transformed into a grid frame using a series of transformations. The results are presented in the form of plots.

During experimentation, it was observed that when the Bresenhem2D model was run without log odd updates, the resulting plot was very noisy, as it included all the rays observed by the LiDAR. However, this noise can be filtered out by applying log-odd updates, which would eliminate low-probability areas of obstacles.

The particle filter was then applied using a particle size of 100 and random Gaussian noise with mean 0 and variance 0.005 was added to the motion model updates for each particle. The particle filter was run on both datasets, and the results were displayed in plots.

However, due to the code's lack of vectorization, the program takes a high computation time and uses a lot of memory. This time constraint has limited the number of times the particle filter can be run to test multiple occupancy grid construction approaches. I have not used log odds in building the occupancy grid which has led to a distorted map of the environment. I have only used binary mapping. The use of log odds was taking high computation time and exceeding the RAM of Google Colab. I tried vectorizing the code but was not able to improve the efficiency after one point. Although, I have included the logodds implementation in my final code. Hence, I have done only 0-1 mapping.

V. Acknowledgement

I collaborated with Mihir Kulkarni (A59018127) and Yeshwant Matey (A59015107) for the assignment where we helped in debugging each other's code.

VI. References

[1] https://natanaso.github.io/

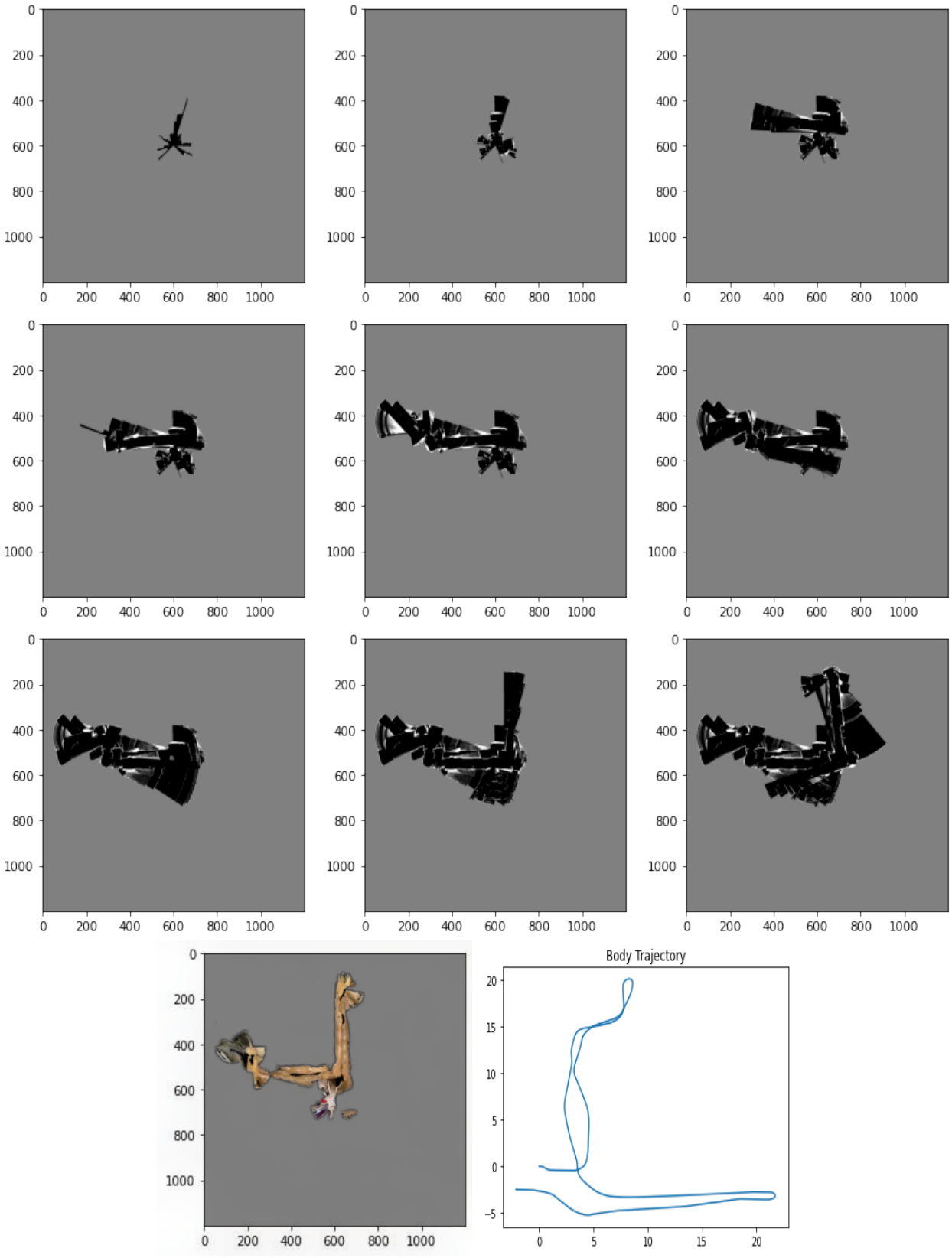

Fig. 4. Final Occupancy grid for dataset 20 after every 500 timesteps (img 1-9), texture map for dataset 20 (img 10), the trajectory (90 degrees clockwise rotated) for dataset 20 (img 11)

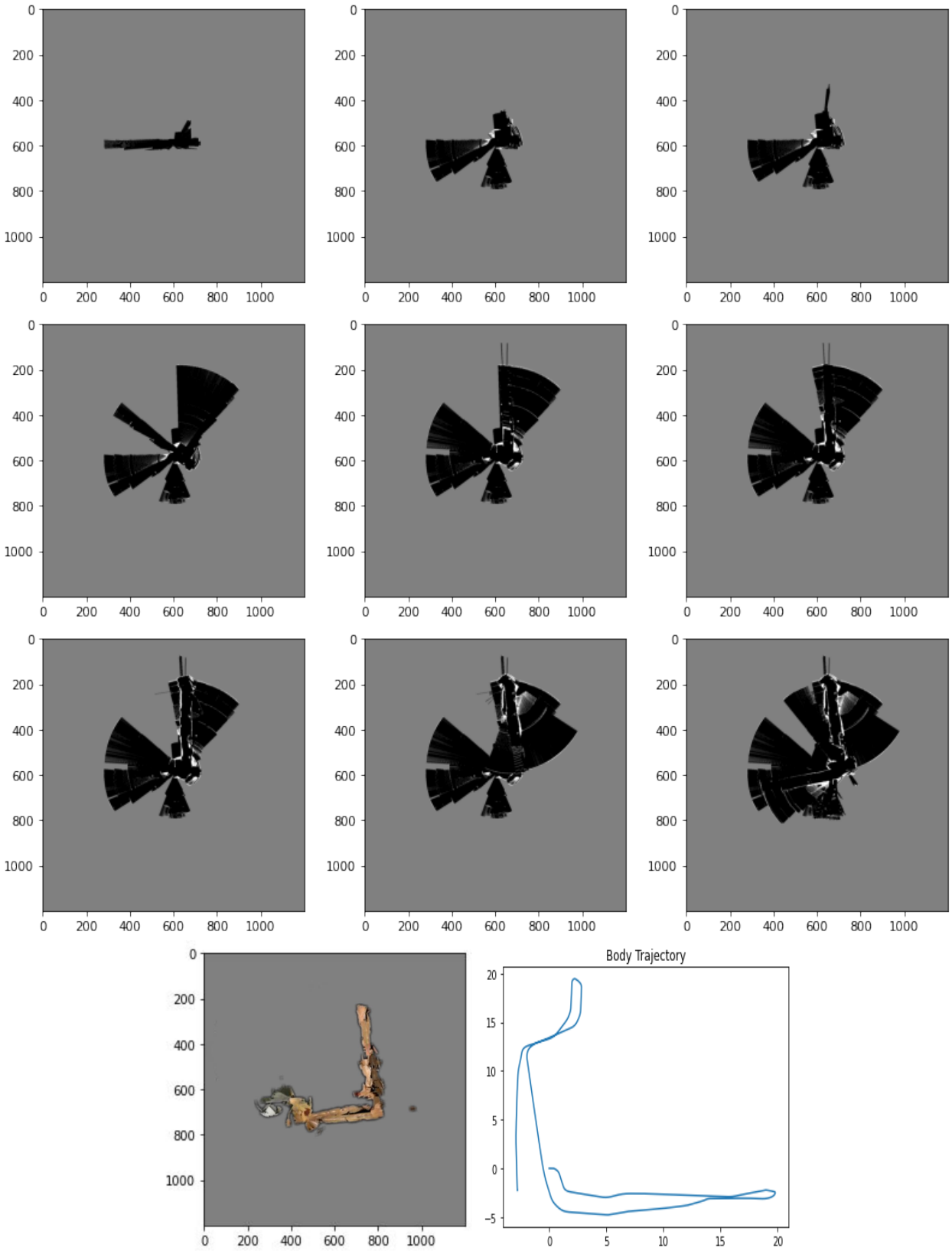

Fig. 5. Final Occupancy grid for dataset 21 after every 500 timesteps (img 1-9), texture map for dataset 21 (img 10), the trajectory (90 degrees clockwise rotated) for dataset 21 (img 11)