ECE276A Project 2: Particle Filter SLAM

Mustafa Shaikh
Department of Electrical and Computer Engineering
University of California, San Diego
La Jolla, U.S.A

I. Introduction

As the scope of applications for autonomous mobile robots increases, it is crucial that these systems are able to operate in new environments with little to no prior knowledge of their surroundings. The problem of *simultaneous localization and mapping*, herein referred to as SLAM, is therefore an important problem to solve in order to successfully deploy mobile autonomous robots. SLAM refers to the problem of determining a robot's location in an environment, while at the same time creating a map of the environment [1]. Since there is error in the motion of the robot [1], as well as noise in the sensor measurements used to create the map, a two stage iterative approach to the problem naturally follows. In this project we present the results of a particle filter based approach to the SLAM problem, in which we generate a map of an unknown environment using various sensor measurements from on-board the mobile robot. Comparing with images captured by camera systems on the robot as it moves around in the environment, we are able to confirm that the map generated by this approach provides an accurate representation of the environment. We then create a texture map using images captured from on-board cameras and overlay these onto the map obtained from the SLAM portion of the problem.

II. Problem Formulation

Consider a mobile robot moving in the xy-plane. The state of the robot at time t can be described by its pose x_t, a combination of its position - an (x,y) pair - and orientation, given by an angle θ in the world frame. The current estimate of the environment that the robot operates in can be represented by a map, denoted by m_t. The motion of the robot can be thought of as caused by a set of control inputs u_t - in this case, linear velocity v_t and angular velocity ω_t - applied to the robot at every time step. The motion of the robot is governed by a discrete-time probabilistic kinematic model, herein referred to as the *motion model* [3]:

$$x_{t+1} = f(x_t, u_t, w_t) \sim p_f(\circ|x_t, u_t),\ w_t = \text{motion noise} \quad (1)$$

The presence of error in the motion of the robot naturally leads to the probabilistic model seen above. In the current context, the robot is a differential drive robot, whose motion model is given by:

$$T_{k+1} = T_k \exp\left(\tau_k \hat{\zeta}\right), \quad (2)$$

where T_k represents the pose at time t, $\hat{\zeta}$ is the hat map of the twist vector $\zeta(t) = \begin{bmatrix} v(t) & \omega(t) \end{bmatrix}$. At each time t, the robot captures information about its environment using on-board sensor arrays. These measurements z_t can be modelled by a probabilistic *observation model*, given by:

$$z_t = h(x_t, v_t) \sim p_h(\circ|x_t),\ v_t = \text{observation noise} \quad (3)$$

We now formulate the SLAM problem. SLAM can be thought of as a single optimization problem with two components; localization, and mapping. The aim of the mapping problem is to generate a map m of the environment given the robot's state trajectory $\mathbf{x_{0:T}}$ and sensor measurements $\mathbf{z_{0:T}}$ by minimizing the following cost function:

$$\min_{\mathbf{m}} \sum_{t=0}^{T} \|\mathbf{z_t} - h(x_t, m)\|_2^2, \quad (4)$$

where $h(x_t, m)$ is the observation model described above. The localization problem takes as an input the map m of the environment, the sensor measurements and control inputs in order to estimate the robot's state x. This is achieved by minimizing the following cost function:

$$\min_{\mathbf{x_{0:T}}} \sum_{t=0}^{T} \|\mathbf{z_t} - h(x_t, m)\|_2^2 + \sum_{t=0}^{T-1} \|\mathbf{x_{t+1}} - f(x_t, u_t)\|_2^2, \quad (5)$$

where $f(x_t, u_t)$ is the motion model described above. Therefore the complete SLAM problem consists of minimizing the above cost function with respect to both the robot state x, and the map m. Alternatively, the probabilistic interpretation of the above problem is to compute the joint posterior distribution of the robot's state and the map conditioned on the control inputs and sensor measurements: $p(x_t, m|u_t, z_t)$ [1].

We now formulate the texture mapping problem. Similarly to the mapping portion of the SLAM problem, we seek to generate the optimal map given sensor measurements - in this case the camera system - and the camera observation model:

$$\min_{\mathbf{m'}} \sum_{t=0}^{T} \|\mathbf{z_t} - h_{cam}(x_t, m)\|_2^2, \quad (6)$$

where m' refers to a map populated by RGB values corresponding to the images captured by the camera system. $h_{cam}(x_t, m)$ refers to the camera observation model.

III. Technical Approach

We approach the problem in two parts. The first consists of the SLAM problem in which we generate the optimal map using the robot's trajectory through an unknown environment. The second part of the problem consists of texture mapping. We first present our approach to the SLAM problem.

A. Particle Filter

We use a particle filter approach to solve the SLAM problem. The particle filter is a special case of the Bayes filter in which the complexity of the problem is reduced by considering a finite set of possible prior locations, called particles, for the robot's pose at a given time [1]. This is in contrast to the Bayes filter in which the distribution of robot's state at time t must be integrated, which poses a challenge for implementation on a computer.

The particle filter uses N particles with locations $\mu[k]$ and weights $\alpha[k]$ to represent the probability distribution functions $p_{t|t}$ and $p_{t+1|t}$:

$$p_{t|t}(x_t) = \sum_{k=1}^{N} \alpha_{t|t}[k]\delta(x_t - \mu_{t|t}[k]) \tag{7}$$

$$p_{t+1|t}(x_{t+1}) = \sum_{k=1}^{N} \alpha_{t+1|t}[k]\delta(x_{t+1} - \mu_{t+1|t}[k]) \tag{8}$$

These are then plugged in to the Bayes filter prediction and update steps to give the particle filter equations.

1) Prediction step: In the prediction step, the differential drive motion model is applied to each of N particles. This has the effect of providing a prediction of the pose of the particle at the next time step [3].

$$\mu_{t+1|t}[k] = f(\mu_{t|t}[k], u_t + \epsilon_t), \alpha_{t+1|t}[k] = \alpha_{t|t}[k], \tag{9}$$

where f(x,u) is the differential drive motion model described above, $u_t = (v_t, \omega_t)$ is the linear and angular velocity input obtained from encoder and IMU readings, and ϵ_t is Gaussian zero mean noise with variance proportional to the incremental linear and angular velocity at each time step. We decided to apply the noise directly to the control inputs rather than applying the noise to the state as we assume that the source of the noise comes from the control inputs and motors that propel the robot. Mathematically the linear velocity noise can be represented as $\epsilon_{t_v} \sim \mathcal{N}(0, \sigma_v)$, where $\sigma_v = 0.02 * (v_{t+1} - v_t)$. The noise for angular velocity is similarly defined with respect to incremental angular velocity at each time step. Note that only the particle poses change during the prediction step; the particle weights remain unchanged.

2) Update step: In the update step, the weight distribution of each particle is updated. The aim of this step is to determine which of the N particles is the most likely true position of the robot given the current sensor measurements and the latest map. We plug the expression for the distribution given by (8) into the Bayes filter update step, which is an application of Bayes rule to obtain the posterior density of the latest observation given the latest position obtained from the prediction step. For this, we first define a LiDAR observation model $p_h(z|x, m)$ given the current pose and map estimate. We define this likelihood to be proportional to the correlation between the current scan's world frame projection y and the latest estimate of the occupancy grid m:

$$p_h(z|x, m) \propto corr(y, m) \tag{10}$$

The actual update is performed on the weight vector and is achieved by normalizing the product of the weights at time t and the correlation obtained using the equation above [3]:

$$\alpha_{t+1|t+1}[k] \propto corr(y_{t+1}[k], m)\alpha_{t+1|t}[k], \tag{11}$$

where $corr(y, m)_t$ is given by:

$$corr(y, m) = \sum_i 1(y_i = m_i), \tag{12}$$

which is the sum of all matches between map cells m_i at time t and LiDAR scan endpoints y_i in the world frame at time t.

3) Data Pre-processing: We begin the implementation by processing the encoder data, which provides linear velocity measurements for the robot. Encoders count the rotations of each wheel at 40Hz, where the raw reading represents the number of ticks that have passed a fixed point in the given time frame. Given the wheel diameter and 360 ticks per revolution, the wheel travels 0.0022 meters per tic. Given encoder counts [FR,FL,RR,RL] corresponding to the front right, front left, rear right, rear left wheels, the right wheels travel a distance of $(FR + RR)/2 * 0.0022m$ while the left wheels travel a distance of $(FL + RL)/2 * 0.0022m$.

The IMU data requires minimal pre-processing, as we only use the yaw rate measurement since the robot is moving in the xy-plane and therefore any turns are conducted about the z-axis. We passed the noisy IMU data through a low pass filter with bandwidth of 10Hz to filter out the high frequency components caused by sensor noise.

Next, we process the 2-D LiDAR data. The LiDAR data stream consists of arrays of range values taken at 1080 regular intervals from -135° to 135°. These can be considered as spherical coordinates which can be converted to cartesian (x,y) pairs using standard conversions from spherical coordinates. This provides the (x,y) coordinates of obstacles in the LiDAR frame.

4) Occupancy Grid: In order to discuss the map update, we first develop some concepts needed to understand the procedure. We use an occupancy based surface model in which we split the environment up into cells m_i. The naive assignment method is to simply +1 to occupied cells and 0 to free cells, and plot the resulting map. However, given sensor noise, we establish a sensor observation model which reflects our trust in the sensor measurements. We wish to reflect our confidence in the true state of a cell m_i by using a probabilistic approach to map assignment. We model each cell as a Bernoulli random variable, and assume each cell m_i in the map is independent given the robot trajectory and so we can track the distribution $p(m_i|z_{0:T}, x_{0:T})$ for each cell separately. By applying Bayes rule and computing log odds of a cell being occupied or free, and using an observation

model that the sensor is correct 80% of the time, we arrive at the following map update steps [3]:

$$\log \frac{p(m_i = 1|z_t, x_t)}{p(m_i = -1|z_t, x_t)} = +/ - log4, \tag{13}$$

where we apply $+\log 4$ if z_t indicates that cell m_i is occupied, and $-\log 4$ if z_t indicates that cell m_i is free. To update the map, we first apply the prediction and update steps at each time t. After updating the weight vector in the update step, we then update the occupancy grid map estimate. This is done by selecting the particle with the highest weight from the update step, assuming this is the true robot pose, and projecting the LiDAR scan to world using this particle's pose. Mathematically we represent the optimal particle as follows:

$$\mu[k]^* = \arg\max_{\alpha} \mu_{t+1|t}[k] \tag{14}$$

Once the LiDAR scan has been projected to world using the optimal particle's pose, we apply the Bresenham ray tracing algorithm to determine the cells m_i that the LiDAR beam passes through - these cells are considered free. The endpoint of the LiDAR scan is assumed to be the cell m_i that is occupied. Then the map cells are updated according to the update equations above.

5) Resampling: In order to prevent our belief over the true lcoatino of the robot to become too confident, i.e. to weight a subset of the particles much higher than others, we implement resampling if the weight vector skews too much towards certain particles. This is done by using the following measure of particle 'spread' [3]:

$$N_{eff} = \frac{1}{(\sum_k (\alpha_{t+1|t+1}[k])^2)} \tag{15}$$

If N_{eff} falls below $N/10$, we uniformly sample from the particle poses at the previous time step with probability according to the previous weight vecotr, and set all the weights equal. This has the effect of sampling more particles from the higher weight particles in the previous time step and superimposing them on top of each other. It also ensures that lower weight particles are less likely to be randomly sampled and so the particle distribution should cluster around the more likely particle positions.

B. Texture Mapping

We work with 480x640 RGB and disparity images captured by a camera array on-board the robot. The aim is to project the images captured by the robot onto the floor of the map to obtain a coloured, visual representation of the actual environment that the robot moves in.

First, since the RGB camera and disparity camera are not located in the same position on the robot, the RGB pixel locations need to be mapped to the disparity pixel array. Given the values d at pixel (i, j) of the disparity image, we can find the pixel location of the corresponding RGB colour using the following conversions:

$$\begin{aligned} dd &= (0.00304d + 3.31) \\ depth &= 1.03/dd \\ rgbi &= (526.37i + (4.51750.46)dd + 19276.0)/585.051 \\ rgbj &= (526.37j + 16662)/585.051 \end{aligned} \tag{16}$$

These conversions provide pixel locations in the optical frame. We then scale these with the depth information from the disparity images, and convert to optical frame coordinates via the following map:

$$\begin{bmatrix} X_o & Y_o & Z_o \end{bmatrix} = K^{-1} \begin{bmatrix} rgbi & rgbj & 1 \end{bmatrix} * depth \tag{17}$$

We then convert these to the regular camera frame using the optical to camera frame rotation matrix. From the camera frame we convert to the body frame using the known position of the camera with respect to the origin of the body frame. The rotation matrix is constructed using the x,y offset as well as roll, pitch and yaw information provided in the robot configuration document. Finally, for each time step t, the pixel coordinates in the body frame are converted to the world frame using the pose of the robot at time t obtained from the particle filter:

$$\begin{bmatrix} X_w & Y_w & Z_w \end{bmatrix} = R_{bodytoworld} \begin{bmatrix} X_b & Y_b & Z_b \end{bmatrix} \tag{18}$$

To convert these (x,y) values in the world frame into a colour map requires projecting the RGB values corresponding to the world frame coordinates obtained using the transformations above into an occupancy grid of the same size as that used in the particle filter. Note that values outside the range of the occupancy grid size are possible once the coordinates have been transformed to world, and so must be clipped. Since the camera is located at an elevation when compared to the origin of the body frame, the converted world frame pixel coordinates will have elevation values associated with them. Since we only aim to project the map onto the floor, we threshold all indices with z-value greater than the height of the camera above the robot body frame origin.

C. Technical Details

We use a MacBook Pro with 1.4 GHz Quad-Core Intel Core i5, 8 GB 2133 MHz LPDDR3, Intel Iris Plus Graphics 645 1536 MB. We make use of the well known numpy and opencv libraries in Python for reading in images. We also make extensive use of the numpy einsum matrix multiplication function that allows arrays to be multiplied along user-provided axes to avoid using extra loops.

IV. Results

We first present the results from the SLAM problem, and then move on to texture mapping.

A. SLAM

Before implementing the particle filter, we first confirm that the prediction step (i.e. the motion model) is providing the expected results, and that the initial LiDAR scan (before the robot moves), generates a map that is in accordance with the room that the robot is in. Fig. 1 shows the initial LiDAR scan map which appears to provide a reasonable map of the first view that the robot has of the environment.

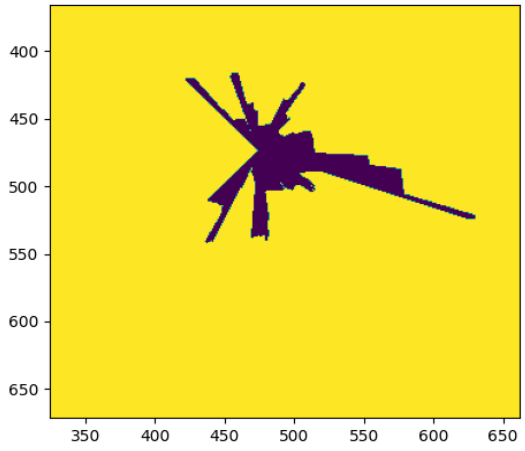

Fig. 1. Initial LiDAR scan map dataset 20

1) Dead Reckoning: We first show the results from dead reckoning, which is a successive application of the motion model to a single particle without introducing nose into the control input (i.e. single hypothesis on robot pose at each time t). We then plot the trajectory to ensure the motion model is implemented correctly. The figure above shows that the robot motion predicted by the motion model matches up with the actual path the robot takes, when checked against the image set provided.

We also show the occupancy map achieved from dead reckoning, i.e. with a single particle with no noise. We will see that the difference in the map between dead reckoning and particle filter is subtle, and mostly in how sharp the final map appears to be.

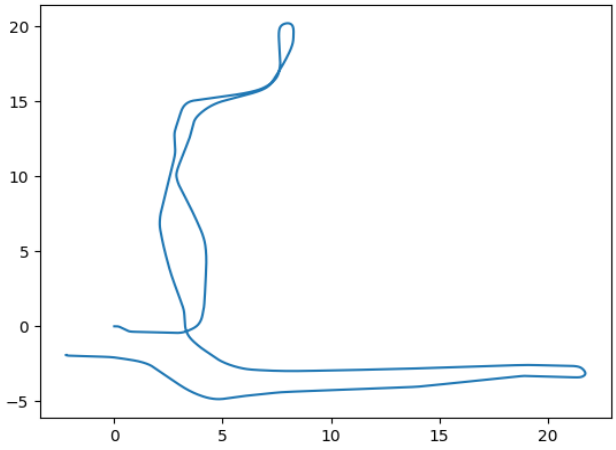

Fig. 2. Dataset 20 dead reckoning trajectory

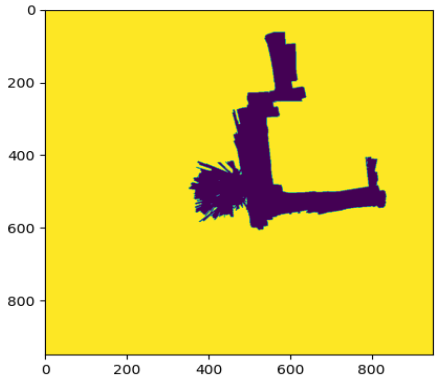

Fig. 3. Dataset 20 dead reckoning occupancy map

2) Particle Filter Occupancy Map: In this section we present the results of the occupancy map generated using the particle filter. We used a particle filter with 20 particles, and were limited to lower values due to code efficiency. However, as we will see the results are quite satisfactory even with a relatively low number of particles. The map generated at 3 different time steps for both datasets is shown. As the robot passes through the main corridor and returns in the second half of the dataset, we see that the detail in the corridor becomes sharper as it has another set of LiDAR measurements with which to fine tune the occupancy map. The overall results are fairly similar to dead reckoning, as the particle filter essentially provides small corrections to the robot pose at every time step t which provides a slightly more accurate LiDAR projection to world and hence map estimate. However, these updates are minor since particles are clustered close together. We also show the trajectory of the 20 particles over time with the noise added. Since the noise is a small percentage of the control

inputs at each time step, the trajectory of each particle is quite similar. Future experiments could include increasing the noise ratio.

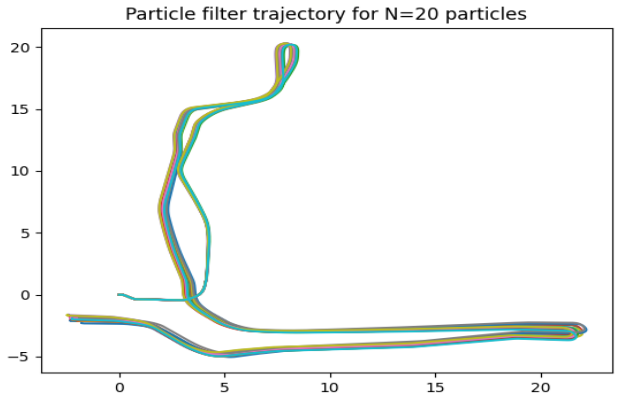

Fig. 4. Dataset 20 particle filter trajectory plot, N=20

Fig. 5. Dataset 21 particle filter occupancy map at t=1500, N=20

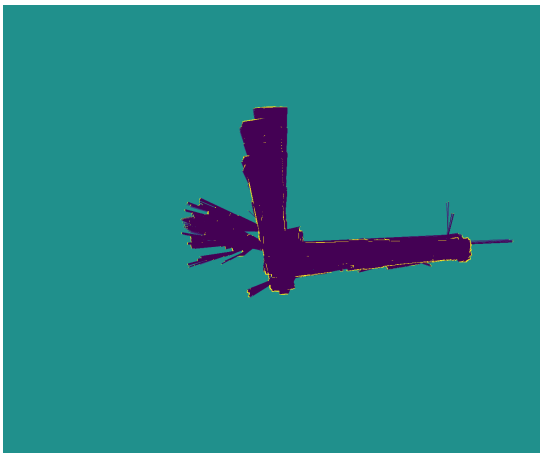

Fig. 6. Dataset 21 particle filter occupancy map at t=3000, N=20

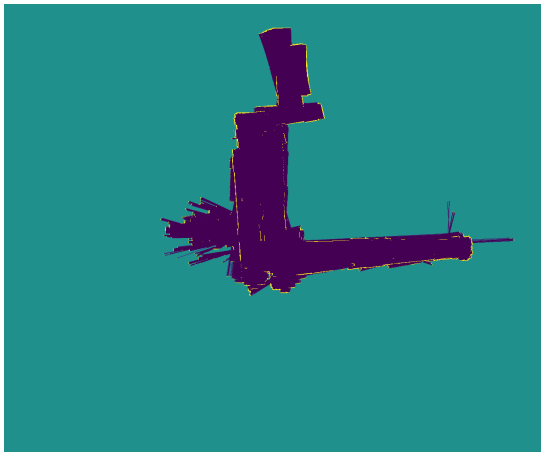

Fig. 7. Dataset 21 particle filter occupancy map by end of dataset, N=20

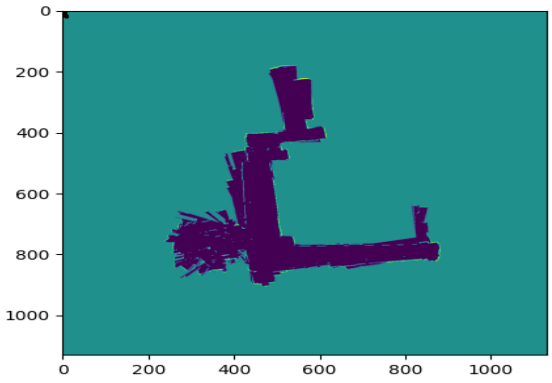

Fig. 8. Dataset 20 particle filter occupancy map by end of dataset, N=20

During execution of the code with perturbations as described in the technical details section led to the weight vector rapidly converging to values in which a single particle dominated the others. This caused resampling to occur frequently, and provided sub-optimal results. This was likely caused by a bug in the implementation of the perturbation logic. The figures below demonstrate some of the issues encountered during the mapping exercise with perturbations. Consequently, we implemented the particle filter with N=20 particles without the 9x9 grid of perturbations around the particle. We see that the results are fairly good even without the perturbations. We noted that resampling did not occur without the perturbations implementation. In future work we would experiment with applying yaw angle perturbations to each particle instead of just x,y offsets.

Fig. 9 below shows that the initial LiDAR scan matches closely with the occupancy map generated by the first LiDAR scan. This is expected as the robot has not yet moved. By the time we get to t=200, we see in Fig. 10 that the occupancy map and the LiDAR scan become displaced, despite the fact that the robot has not yet moved. This is likely due to errors

in the body to world conversion of the LiDAR scan. In future work, such bugs would be addressed first.

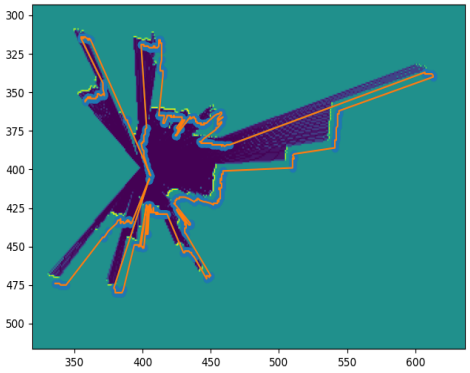

Fig. 9. Initial LiDAR scan outline vs. occupancy map at first time step

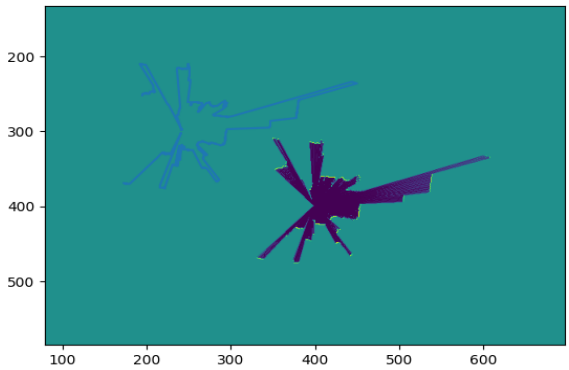

Fig. 10. LiDAR scan outline vs. occupancy map at time t=200

B. Texture Mapping

In the figures below we see that the texture mapping has provided fairly satisfactory results. We show the texture maps as they evolve for times t = 800 and t = 1500, and we can see that more detail is added through time, and also the map is expanded as the robot explores new territory. Although the orientation of the map is rotated, the shape with respect to the robot is the same as the shape of the map as seen in the particle filter section. Dataset 21 seems to have more noise in the texture mapping, and this could be because of a misalignment of time stamps with the image timestamps, or of large overlap between the x,y coordinates obtained after the transformations applied to the original pixels.

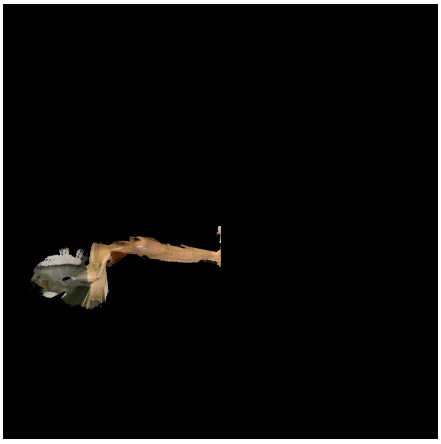

Fig. 11. Texture map at time 800 dataset 20

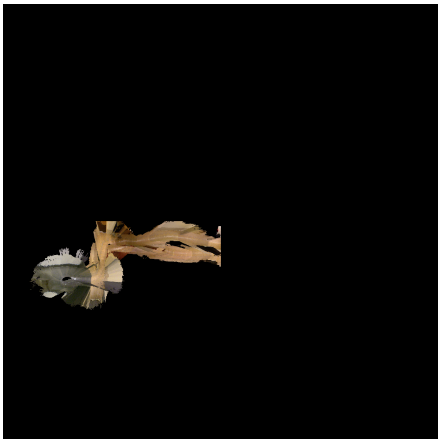

Fig. 12. Texture map at time 1500 dataset 20

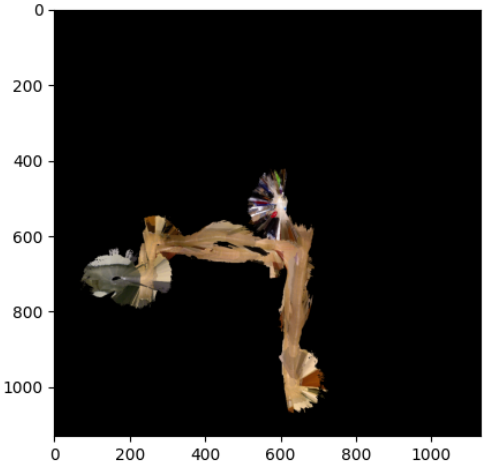

Fig. 13. Texture map at end of dataset 20

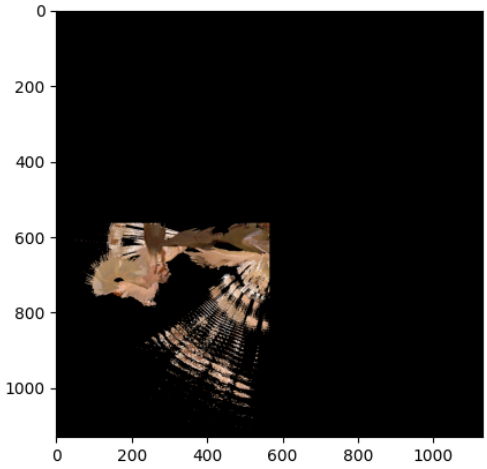

Fig. 14. Texture map at end of dataset 21

REFERENCES

[1] M. Montemerlo, S. Thrun, D. Koller, B. Wegbreit, ”FastSLAM: A Factored Solution to the Simultaneous Localization and Mapping Problem”
[2] ”https://natanaso.github.io/ece276a/ref/ECE276A_7_BayesFilter.pdf”
[3] ”https://natanaso.github.io/ece276a/ref/ECE276A_8_ParticleFilter.pdf”