Particle Filter SLAM

Dhruv Talwar
Electrical and Computer Engineering
ECE 276A

I. Introduction

Simultaneous Localization and Mapping (SLAM) is a fundamental problem in robotics that has been researched for several decades. It involves building a map of the environment from the observations of a robot, while also estimating the robot's pose relative to that map. A reliable estimate of the robot's position is necessary for constructing a consistent and accurate map.

The problem of SLAM has attracted considerable attention in the field of robotics. Implementation of SLAM can be seen everywhere from indoor warehouse robots, to self driving cars to autonomous underwater vehicles as well. In these applications, robots need to operate in unknown environments, which makes SLAM a crucial technology for enabling them to navigate and make decisions autonomously.

This paper presents an implementation of the particle filter based SLAM for determining the position and orientation of the robot. The position and orientation estimation is done leveraging the IMU sensor and the rotary encoders. A 2D lidar is used to create a 2D occupancy map of the environment of where the robot traversed. As seen in project 1 IMU is susceptible to high noise, the occupancy grid is leveraged to rectify any position noise associated with the sensors. To ensure the accuracy of the predicted position of the robot, a correction process is implemented by cross-checking the newly obtained lidar readings with the existing ones. To further improve the mapping and trajectory we implement a particle filter and calculates several pose states at a time and correcting them according to the correlation with the observed state from the lidar measurements. This ensures that the predicted position aligns with the actual position of the robot and makes logical sense within the context of the environment. Upon determining the robot's trajectory, depth and color data from an RGBD camera, specifically the Microsoft Kinect, are utilized along with homogeneous transformations in 3D space. This process aims to determine and color points in the occupancy grid that correspond to the ground plane.

The results of the construction of the occupancy grid, the particle filter based SLAM and the texture map are presented and evaluated in this paper. The effectiveness of the proposed approach in simultaneous localization and mapping is demonstrated.

II. Problem Formulation

The aim of this project is to implement Simultaneous Localization and Mapping (SLAM) using the particle filter. To fully comprehend the SLAM problem, it is crucial to first understand the fundamental principles of SLAM and Particle Filter.

A. SLAM and Bayes Filter

The problem of SLAM involves, given sensor measurements $z_{0:t}$ and inputs $u_{0:t-1}$ estimating the robot state x_t at time t and map state m, under Markov assumptions. The Markov assumptions state that The state x_{t+1} only depends on the previous input u_t and state x_t, and is independent of the history $x_{0:t1}$, $z_{0:t1}$, $u_{0:t1}$ and the observation z_t only depends on the state x_t The robot state in our case x_t comprises of

$$\begin{pmatrix} X \\ Y \\ \theta \end{pmatrix}$$

Where X and Y are the coordinates in X and Y axis and θ is the yaw. The state x_t and map m can be computing the joint probability distribution of the agent state and map state as described by the following equation:

$$p(x_{0:T}, m | z_{0:T}, u_{0:T-1}) \tag{1}$$

To solve this problem, we leverage the motion model and observation model. The motion model describes the state of the agent given the previous state and control input as in the following equation:

$$x_{t+1} = f(x_t, u_t, w_t) \sim p_f(\cdot | x_t, u_t) \tag{2}$$

where f defines the motion model and w_t is the motion noise. The observation model helps model the surroundings based on the observation z_t conditioned on the observation model h and observation noise v_t as in the following equation:

$$z_t = h(x_t, v_t) \sim p_h(\cdot | x_t) \tag{3}$$

To simplify the equation and make it easier for understanding we aim to club the map m and x_t as one and will be calling it x_t from now on. Using Equations (2) and (3) and applying them to Equation (1) under the Markov assumption, we can break down the problem into the following equation:

$$p(x_{0:T}, z_{0:T}, u_{0:T-1}) = p(x_0) \prod_{t=0}^{T} p_h(z_t|x_t) \cdot \prod_{t=0}^{T-1} p_f(x_{t+1}|x_t, u_t) \cdot \prod_{t=0}^{T-1} p(u_t|x_t) \tag{4}$$

Bayes filtering is a statistical technique used to estimate the state x_t of dynamic systems, such as robots, by incorporating evidence from both control inputs and observations. This is achieved by utilizing the Markov assumptions and Bayes rule. To keep track of the probability density functions $p_{t|t}(x_t)$ and $p_{t+1|t}(x_{t+1})$, the Bayes filter relies on two steps.

Prediction step: given a prior probability density function $p_{t|t}(x_t)$ of x_t and control input u_t, use the motion model p_f to compute the predicted probability density function $p_{t+1|t}(x_{t+1})$ of x_{t+1}:

$$p_{t+1|t}(x_{t+1}) = \int p(x_{t+1}|x_t, u_t) \cdot p_{t|t}(x_t) dx_t \tag{5}$$

where $p(x_{t+1}|x_t, u_t)$ is the motion model which describes the probability of the state x_{t+1} given the current state x_t and control input u_t. The integral is taken over all possible values of x_t.

Update step: given a predicted probability density function $p_{t+1|t}(x_{t+1})$ of x_{t+1} and measurement z_{t+1}, use the observation model p_h to obtain the updated probability density function $p_{t+1|t+1}(x_{t+1})$ of x_{t+1}:

$$p_{t+1|t+1}(x_{t+1}) = \frac{p(z_{t+1}|x_{t+1}) \cdot p_{t+1|t}(x_{t+1})}{\int p(z_{t+1}|x_{t+1}) \cdot p_{t+1|t}(x_{t+1}) dx_{t+1}} \tag{6}$$

where $p(z_{t+1}|x_{t+1})$ is the observation model which describes the probability of the measurement z_{t+1} given the state x_{t+1}. The integral is taken over all possible values of x_{t+1}.

B. Particle Filter

The particle filter is a type of Bayesian filter used to estimate the state of a dynamic system by maintaining a set of weighted samples (particles) representing the posterior probability distribution. In this project we shall see how effective is the particle filter SLAM.

The particle filter is a method for estimating the state of a dynamic system. In this approach, each particle represents a hypothesis about the value of the state variable x at a particular time step. Specifically, particle k represents the hypothesis that $x = \mu_k$ with probability a_k, where μ_k represents the location and a_k represents the weight of the particle.

Based on these definitions, probability density functions can be constructed to represent the distribution of particles over the state space

$$P_{t|t}(x_t) = \sum_{k=1}^{N} \alpha_k^{t|t} \delta(x_t - \mu_k^{t|t}) \tag{7}$$

$$P_{t+1|t}(x_{t+1}) = \sum_{k=1}^{N} \alpha_k^{t+1|t} \delta(x_{t+1} - \mu_k^{t+1|t}) \tag{8}$$

Here, δ represents the Dirac delta function, N is the number of particles at time t, and $\alpha_k^{t|t}$ and $\mu_k^{t|t}$ represent the weight and location of the kth particle at time t. Similarly, $\alpha_k^{t+1|t}$ and $\mu_k^{t+1|t}$ represent the weight and location of the kth particle at time $t+1$ predicted using information up to time t. We will have to develop the predict and update step based on our motion and observation models. For the update step, we also need to check among the N particles which has the highest correlation with the current sensor scan and use that state for particle weight updation and map generation. We would also would have to check for re sampling if in case the effective particle numbers drop significantly.

The particle filter prediction step is as follows

$$\hat{\mathbf{p}}_{t+1|t}(\mathbf{x}_{t+1}) \approx \frac{1}{N} \sum_{k=1}^{N} \alpha_{t+1|t}[k] \delta(\mathbf{x}_{t+1} - \boldsymbol{\mu}_{t+1|t}[k]) \tag{9}$$

The Update step equation to solve for is as follows:

$$\mathbf{p}_{t+1|t+1}(\mathbf{x}_{t+1}) = \sum_{k=1}^{N} \frac{\alpha_{t+1|t}[k] p(\mathbf{z}_{t+1}|\boldsymbol{\mu}_{t+1|t}[k])}{\sum_{j=1}^{N} \alpha_{t+1|t}[j] p(\mathbf{z}_{t+1}|\boldsymbol{\mu}_{t+1|t}[j])} \times \delta(\mathbf{x}_{t+1} - \boldsymbol{\mu}_{t+1|t}[k]) \tag{10}$$

One other challenge would be of particle depletion, Particle depletion occurs when the most updated particle weights become close to zero due to a finite number of particles being insufficient to represent the state pdf. For example, the observation likelihoods $p(\mathbf{z}_{t+1}|\boldsymbol{\mu}_{t+1|t}[k])$ may be small for all $k = 1, ..., N$. We will be implementing resampling in our technical approach.

Given a particle set $\boldsymbol{\mu}_{t|t}[k], \alpha_{t|t}[k] k = 1^N$, resampling is applied if the effective number of particles, defined as $N_{\text{eff}} := 1/\sum_{k=1}^{N}(\alpha_{t|t}[k])^2$, is less than a threshold.

C. Occupancy grid construction

In this part of the report we shall be understanding the problems and challenges that need to be encountered and tackled while building a 2D occupancy grid. The problem of Occupancy grid formations can be divided into the following sections:

1) Data understanding and correction: The dataset provided contains raw IMU data, rotary encoder data, 2D lidar data and RGBD camera data. The encoder data contains rotation count and this is given at 40 Hz. The IMU data contains the linear acceleration and angular velocity measurements. One of the challenge presented in the dataset is that each sensor data records in its own frequency and time, thus the sensor data is not synchronized.

Another challenge in the data is that is The IMU data captured is noisy and is not smooth throughout the time period. Thus the noise from the IMU needs to be removed, implementing a low pass filter for this.

2) Occupancy Grid: To develop the occupancy grid the challenge remains to divide the environment into a regular grid consisting of n cells, each of which represents a pixel in 2D. The occupancy of each cell is then represented by a binary value in the vector m, where m_i = -1 if the cell is free and m_i = 1 if it is occupied. The ultimate goal of occupancy grid mapping is to estimate the posterior probability over time, which is a function of the sensor observations and the known robot poses. Mathematically the goal of the occupancy grid is to estimate the posterior probability over time.

$$p(m_j|z_{0:t}, x_{0:t}) \tag{11}$$

We use the independence assumption that the occupancy grid cells are independently conditioned on the robot and will solve for this equation for each cell:

$$p(m|z_{0:t}, x_{0:t}) = \prod_{i=1}^{n} p(m_i|z_{0:t}, x_{0:t}) \tag{12}$$

We also have to convert the lidar range values from the Cartesian to the XYZ coordinates and from there to the world frame. We calculate this using the state calculated from the motion model, we will have to transform it to the world frame. The observation model calculates the probability of observing these ranges at a given location, given the map and the robot's pose. Let $z_{k,i}$ be the range measurement obtained by the i^{th} laser beam at time step k. The expected range $\hat{z}_{k,i}$ at the i^{th} laser beam can be calculated as the distance between the robot's position (x_k, y_k) and the nearest obstacle along the beam's direction α_i:

$$\hat{z}_{k,i} = h(x_k, y_k, \alpha_i) \tag{13}$$

where h is a function that returns the distance to the nearest obstacle.

Finally we will have to convert the world coordinates obtained from the lidar and have to be transformed to the map pixel frame.

Instead of populating the occupancy grid cell with positive binary values, we take a probabilistic approach and increase the probability according to the sensor observations. The occupancy grid cells can be understood by the following equation:

$$p(m_{ij}|z_{0:t}) = \frac{p(z_t|m_{ij}) \cdot p(m_{ij}|z_{0:t-1})}{p(z_t|z_{0:t-1})} \tag{14}$$

where $p(m_{ij}|z_{0:t})$ is the probability of cell m_{ij} being occupied or free given all lidar observations up to time t, $p(z_t|m_{ij})$ is the probability of observing the lidar scan z_t given the state of cell m_{ij}, $p(m_{ij}|z_{0:t-1})$ is the probability of cell m_{ij} being occupied or free given all previous observations up to time $t-1$, and $p(z_t|z_{0:t-1})$ is the probability of observing all previous lidar scans up to time $t-1$ and the current scan z_t.

In our project we will use a similar approach and use log odds for the map grid representation. To update the map based on the lidar scan, we will use the following equation:

$$\lambda_{i,t} = \lambda_{i,t-1} + \Delta\lambda_{i,t} \tag{15}$$

where $\Delta\lambda_{i,t}$ is given by equation :

$$\Delta\lambda_{i,t} = \begin{cases} +\log(k) & \text{if } z_t \text{ is occupied} \\ +\log(1/k) & \text{if } z_t \text{ is empty} \end{cases} \tag{16}$$

The probability can be recovered from the log odds value by applying the sigmoid function.

3) Motion Model: In this work, we aim to formulate a motion model for a differential drive robot. We will use the discrete-time differential-drive kinematic model. The goal here is to use a model to calculate the state of the robot at time t+1, from the state and control inputs at time t. The robot used in this project is a 4 wheel robot, and to develop the kinematic model we have to assume the following before we show the technical approach in the next section.

- The wheels on each side of the robot move at the same speed and do not slip.
- The motion of the robot is only in two dimensions, so that it can be represented using a planar coordinate system.
- The robot does not experience significant external forces that would affect its motion.
- The acceleration of the robot is constant and small enough such that the robot can be considered to be moving at a constant speed during a short period of time.

Using these conditions we can formulate the motion model as shown below.

$$\begin{bmatrix} x_{t+1} \\ y_{t+1} \\ \theta_{t+1} \end{bmatrix} = f_d(x_t, u_t) := \begin{bmatrix} x_t \\ y_t \\ \theta_t \end{bmatrix} + \tau_t \begin{bmatrix} v_t \cos(\theta_t + \frac{\omega_t \tau_t}{2}) \\ v_t \sin(\theta_t + \frac{\omega_t \tau_t}{2}) \\ \omega_t \end{bmatrix} \tag{17}$$

Where x_{t+1}, y_{t+1} and θ_{t+1} is the state at time $t+1$. v_t is the velocity of the robot derived from the encoder, τ_t is the time difference between the consecutive sensor reading and ω_t is the yaw rate derived from the IMU sensor. This is the Exact integration over time interval of length τ_t

D. Texture Mapping

We are given the intrinsic parameters and position of an RGBD camera and the disparity images it provides, we have to calculate the closest SLAM pose that matches the time stamp of the current Kinect scan. And using that we have to create an RGB value array for each pose representing the floor in the occupancy grid map. We are given The K matrix as

$$\begin{bmatrix} f_{su} & f_{s\theta} & c_u \\ 0 & f_{sv} & c_v \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 585.05 & 0 & 242.94 \\ 0 & 585.05 & 315.83 \\ 0 & 0 & 1 \\ & & \end{bmatrix}$$

The problem is to transform the RGB color image to match the depth image obtained from an RGBD camera, which are not in the same location due to an x-axis offset. Given the values of the disparity map, d at pixel (i,j), the depth can be calculated using the following equation:

$$dd = (-0.00304d + 3.31) \tag{18}$$

where dd is the disparity distance. Then, the depth can be obtained using the equation:

$$depth = \frac{dd}{1.03} \tag{19}$$

The corresponding pixel location $(rgbi, rgbj)$ of the RGB color image can be calculated using the following equations:

$$rgbi = \frac{585.051}{526.37i - (4.5 \times 1750.46)dd + 19276.0} \tag{20}$$

$$rgbj = \frac{585.051}{526.37j + 16662} \tag{21}$$

where i and j are the row and column indices of the disparity map, respectively.

$$\begin{bmatrix} X & Y & Z \end{bmatrix} = \pi^{-1} Z_o K^{-1} p \tag{22}$$

Here, K is the matrix containing the intrinsic parameters of the disparity camera, Z_o is the depth captured by the disparity camera in the optical frame, and π is a homogeneous transformation matrix from the optical frame, through the robot body frame, and finally to the world frame. p is a pixel in the disparity image. p = $\begin{pmatrix} u \\ v \\ 1 \end{pmatrix}$ pixels = (u,v)

III. Technical Approach

1) Data understanding and correction
: Time Sync To solve the time sync issue, I have taken the time stamps from the encoder as the pivot, as the motion model update is dependant on values from the encoder and then the closest time to this encoder time has been used for other sensors such as IMU and Lidar. This makes sure that for a particular point we are getting the closest sensor readings.

High frequency noise in IMU data: A low pass filter must be applied to it to remove the high frequency noise and smoothen out the noise. For this project we will only utilize the yaw rate from the IMU measurement To solve the high frequency noise in the IMU data, we have implemented a Butter worth low-pass filter. This allows signals below a specified cutoff frequency to pass through the filter, while attenuating signals above the cutoff frequency. The following equation shows the calculation of the filtered output.

$$y(t) = \int_{-\infty}^{\infty} x(\tau)h(t-\tau)d\tau \tag{23}$$

where $x(t)$ is the high frequency input IMU data, $h(t)$ is the impulse response of a low-pass filter, and $y(t)$ is the filtered output. it is necessary to align the different sensor data to a common time frame. This can be achieved by using timestamp information that is recorded by each sensor. For this we will align the timestamps of the encoder and IMU data which requires correction before it can be processed by subsequent algorithms.

2) Motion and Observation Model: Based on the assumptions mentioned in the Problem formulation section, we use the differential drive kinematic model for the robot. The equation for modeling the motion of the robot is given as

$$\begin{bmatrix} x_{t+1} \\ y_{t+1} \\ \theta_{t+1} \end{bmatrix} = f_d(x_t, u_t) := \begin{bmatrix} x_t + \tau_t v_t \cos(\theta_t + \frac{\omega_t \tau_t}{2}) \\ y_t + \tau_t v_t \sin(\theta_t + \frac{\omega_t \tau_t}{2}) \\ \theta_t + \tau_t \omega_t \end{bmatrix} \tag{24}$$

Where x_{t+1}, y_{t+1} and θ_{t+1} is the state at time $t+1$. v_t is the velocity of the robot derived from the encoder, τ_t is the time difference between the consecutive sensor reading and ω_t is the yaw rate derived from the IMU sensor. This is the Exact integration over time interval of length τ_t : We can use the following equation to calculate the instantaneous velocity at a given time.

$$\begin{aligned} \text{Tick per rev} &= 360 \\ \text{Wheel diameter} &= 0.254m \\ \text{Distance per tick} &= \frac{\text{Wheel diameter} \times \pi}{\text{Tick per rev}} \\ \text{distance FR} &= \text{encoder ticks[FR]} \times \text{Distance per tick} \\ \text{distance FL} &= \text{encoder ticksFL} \times \text{Distance per tick} \\ \text{distance RR} &= \text{encoder ticks[RR]} \times \text{Distance per tick} \\ \text{distance RL} &= \text{encoder ticks[RL]} \times \text{Distance per tick} \\ \text{distance right} &= \frac{\text{distance FR} + \text{distance RR}}{2} \\ \text{distance left} &= \frac{\text{distance FL} + \text{distance RL}}{2} \\ \mathrm{v_t} &= \frac{\text{distance right} + \text{distance left}}{2 \times \mathrm{dt}} \end{aligned} \tag{25}$$

Here FR, FL, RR, RL, refers to front right, front left, rear right, rear Left wheel respectively. We use this v_t along with the yaw rate from the IMU to calculate the state at t+1. In this model we assume that there is no slipping and the wheels on the left side rotate together and the wheels on the right rotate together. Thus we are able to take the average of the two to calculate the velocity of the robot. To calculate the θ of the robot from the yaw rate from the IMU we use the following equation:

$$\theta_t = \omega_t * \tau_t \tag{26}$$

Where ω_t is the yaw rate obtained from the IMU sensor and τ_t is the time difference in the sensor reading.

For the observation model, The observation in lidar frame $z_{i,t}$ is obtained by transforming lidar scan data $ranges_i$ with the following equation:

$$z_{i,t} = \begin{bmatrix} ranges_i \cos(\alpha_i) \\ ranges_i \sin(\alpha_i) \\ 0 \end{bmatrix}, \quad \alpha_i = -135^\circ + \frac{i}{1081} 270^\circ \tag{27}$$

Here, α_i is the angle of the i-th lidar measurement in degrees, where the lidar sensor scans 1081 measurements at a resolution of 0.25° per measurement. The z-coordinate is zero because the lidar sensor is mounted on a 2D robot and therefore cannot sense depth. The transformation to the lidar frame is performed by multiplying the polar coordinate $ranges_i$ with the cosine and sine of the angle α_i. The observation model for a 2D lidar is used to estimate the probability of observing a laser scan given the state of the robot. The lidar scans the environment and returns a set of range measurements along different angles. Then from the $z_{i,t}$ computed in the previous step, where $z_{i,t}$ = [x,y,0] we calculate the x_{world} and y_{world}

$$\begin{aligned} x_{world} &= x\cos(\theta) - y\sin(\theta) + T_x \\ y_{world} &= x\sin(\theta) + y\cos(\theta) + T_y \end{aligned} \tag{28}$$

Where θ is the yaw rate and T_x and T_y denote the X and Y translation values all of which are calculated from the motion model state.

Also to convert the world coordinates obtained from the lidar and have to be transformed to the map pixel frame. We shall be using this equation to achieve this.

$$x_{map} = \lceil \frac{x - x_{min}}{resolution} \rceil - 1, \quad y_{map} = \lceil \frac{y - y_{min}}{res} \rceil - 1 \tag{29}$$

Where x and y are the world values to be transformed and res is the resolution of the map and x_{min} and y_{min} are the minimum values of x and y, respectively, in the physical space being mapped to the occupancy grid. They are used as offsets to ensure that the mapped grid indices start at (0, 0).
In this paper, we utilize the laser correlation model as the observation model to tackle the occupancy grid map problem. The pdf $p_h(z_t|x_t, m)$ is obtained by evaluating the correlation between the map m and the lidar scan points in the world frame we just calculated above. The observation model can be obtained by utilizing the softmax function to convert the correlation between the map m and the lidar scan points in world frame z_{world} to probabilities. The equation for the observation model can be written as:

$$p_h(z_t|x_t, m) = \frac{\exp\left(\text{corr}(z_{world}, m)\right)}{z_{world} \exp\left(\text{corr}(z_{world}, m)\right)} \tag{30}$$

where $\text{corr}(z_{world}, m)$ is the correlation function between the lidar scan points z_{world} and the map m given by:

$$\text{corr}(z_{world}, m) = \sum_i f(m_i = y_i) \tag{31}$$

$$f(y_i = m_i) = \begin{cases} 1, & \text{if } y_i = m_i \\ 0, & \text{else} \end{cases} \tag{32}$$

3) Occupancy Map: We will model the map cells m_i as independent Bernoulli random variables.

$$m_i = \begin{cases} +1 \text{ (Occupied) with prob. } \gamma_{i,t} := p(m_i = 1|z_{0:t}, x_{0:t}) \\ -1 \text{ (Free) with prob. } 1 - \gamma_{i,t} \end{cases} \tag{33}$$

We can express the accumulation of log-odds ratio of the binary random variable m_i over time as a means of updating the map, given the occupancy probabilities at time t follow the Bayes rule where $i, t = 1$:

$$\lambda_{i,t} = \log \frac{p(m_i = 1|z_t, x_t)}{p(m_i = -1|z_t, x_t)} = \Delta\lambda_{i,t} - \lambda_{i,0} + \lambda_{i,t-1} \tag{34}$$

In this equation, $\lambda_{i,t}$ represents the log-odds of occupancy for grid cell i at time t, and $\Delta\lambda_{i,t}$ represents the change in log-odds of occupancy due to the latest measurement z_t and control input x_t. $\lambda_{i,0}$ represents the initial log-odds of occupancy for grid cell i, and $\lambda_{i,t-1}$ represents the log-odds of occupancy for grid cell i at the previous time step.

To estimate the probability mass function of the occupancy status of grid cell i at time t, given all previous measurements $z_{0:t}$ and control inputs $x_{0:t}$, we can accumulate the log-odds ratio $\Delta\lambda_{i,t}$ of the inverse measurement model over time, which is equivalent to computing the log-odds of occupancy $\lambda_{i,t}$ as the sum of the log-odds at the previous time step $\lambda_{i,t-1}$ and the change in log-odds $\Delta\lambda_{i,t}$ from the latest measurement and control input, minus the initial log-odds $\lambda_{i,0}$.

If the prior probability distribution of occupancy status for grid cell i is assumed to be uniform, i.e., occupied and free space are equally likely, then the initial log-odds of occupancy $\lambda_{i,0}$ is zero.

Assuming that each measurement z_t indicates whether grid cell i is occupied or not, the log-odds ratio $\Delta\lambda_{i,t}$ of the inverse measurement model specifies the "trust" of the measurement. In our project we are taking for lidar it is 80

$$\Delta\lambda_{i,t} = \log \frac{p(m_i = 1|z_t, x_t)}{p(m_i = -1|z_t, x_t)} = \begin{cases} \log 4 & z_t \text{ indicates } m_i \text{ is occupied} \\ -\log 4 & z_t \text{ indicates } m_i \text{ is free} \end{cases} \tag{35}$$

In the code, I implemented a method to maintain a grid of map log-odds $\lambda_{i,t}$ for each cell i in the map. When a new LiDAR scan z_{t+1} is received, it is transformed to the world frame using the robot pose x_{t+1}. Then, the cells that the LiDAR beams pass through are determined using Bresenham's line rasterization algorithm. All the laser points are transformed to the map coordinates and are given value as occupied, and all the cells in between through which the laser passes are called as free cells. For each observed cell i, the log-odds $\lambda_{i,t}$ are either decreased if the cell is observed free or increased if the cell is observed occupied, the increase or decrease is done by $\log 4$. To avoid overconfident estimation, the log-odds values are constrained within a specified range of λ_{MIN} to λ_{MAX}. The thresh hold for the max and min are set to be 100 and -100 respectively. A decay factor can also be introduced to handle changing maps. Finally, the map probability mass function $\gamma_{i,t}$ is recovered from the log-odds $\lambda_{i,t}$ using the logistic sigmoid function.

$$\gamma_{i,t} = p(m_i = 1|z_{0:t}, x_{0:t}) = \sigma(\lambda_{i,t}) = \frac{\exp(\lambda_{i,t})}{1 + \exp(\lambda_{i,t})} \quad (36)$$

Thus from the output of the motion model and and using the observation model as explained in the previous section we can update the log-odds of the grid map.

4) Particle Filter: The equations used in this project are given in the problem formulation. The steps taken to implement Particle filter in this project are as follows:

- Intialize N particles, each particle N_i is a 3X1 array storing the robot state at time t. Along with each particle we initialize uniform weights. The N_i array looks like $\begin{bmatrix} X_i \\ Y_i \\ \theta_i \end{bmatrix}$. Initially all states are given a 0 value and all the starting weights are $\frac{1}{N}$
- Due to the fact all the particles have the same state, to get diversification and more different samples to try to model the robot state we add normal Gaussian noise to the states at every motion model step. In my application I have added a noise with standard deviation of σ = 0.001. This makes the particles take on different values through the prediction step.
- After the prediction step, only the states in the particle array have changed, as soon as the lidar measurement is taken, the current poses are evaluated against the existing map through the map co relation function. The correlation score in the particle filter is a measure of how well the particles match the measurements obtained from the sensors. The correlation score is calculated by comparing the measurements obtained from the lidar with the predicted measurements generated by each particle. The closer the predicted measurements are to the actual measurements, the higher the correlation score. We calculate the probability of each particle pose in the world frame using the LiDAR scan measurement. This probability is given by the equation:

$$P(z_{t+1}|\mathbf{x}_k) = \frac{\exp(\text{corr})}{\sum_{i=1}^{N} \exp(\text{corr}_i)} \quad (37)$$

where corr represents the correlation score between the predicted scan measurement and the actual measurement, and cor_i is the correlation score of the ith particle. Then, we use this probability to update the weight of each particle by multiplying the weight by the probability:

$$w_{t+1} = w_t P(z_{t+1}|\mathbf{x}_k) \quad (38)$$

To ensure that the weights of all particles sum up to 1 after the update, we normalize the weights by dividing them by the sum of all weights:

$$w_{t+1}^{(k)} = \frac{w_{t+1}^{(k)}}{\sum_{i=1}^{N} w_i} \quad (39)$$

- With each update in the weights we then find out the state from the P_i array with the highest weight. This selected pose is the pose which is the closest to the current observation. Thus after each weight update, we use the particle with the highest weight to estimate the robot's pose. Then, we transform the pose from the LiDAR frame to the world frame, and update the map log-odds using this pose.
- It is pertinent to apply re sampling as well, as Without resampling, the particles with low weights would have a negligible influence on the posterior distribution, while the particles with high weights would dominate the estimate. Re sampling helps to re balance the distribution by discarding low-weight particles and replicating high-weight particles. We implement re sampling when the effective number of particles is less than 0.1.

$$N_{eff} = \frac{1}{\sum_{k=1}^{N} (w_k(t|t))^2} \leq 0.1 \quad (40)$$

- In the code I have used systematic re sampling. The algorithm divides the interval [0, 1] into N equal subintervals, where N is the number of particles. Then, a single random number is drawn from the uniform distribution in the interval [0, 1/N], and this number is added to each of the N subinterval endpoints. The particles that correspond to the subintervals containing the resulting numbers are selected as the new particles. This method ensures that each particle has an equal chance of being selected and avoids the problem of having some particles with very low weights that are unlikely to be selected.
The following equation shows systematic resampling

$$u_i = \frac{1}{N}\left(i - 1 + \frac{r}{N}\right) \qquad i = 1, 2, \ldots, N$$

where N is the number of particles, i is the index of the particle, and r is a random number uniformly distributed between 0 and 1. The particle with index i is then selected by finding the smallest j such that $u_i \leq c_j$, where c_j is the cumulative sum of the particle weights.
Once I get the best state from the particle filter, that state is used for mapping and trajectory generation.

5) Texture Mapping: To utilize texture mapping on the floor surface, it is crucial to determine the points that comprise the surface. To achieve this, a conversion between a pixel p in the disparity image and a point in the 3D world frame is employed. This conversion is described by the following equation:

$$\begin{bmatrix} X \ Y \ Z \end{bmatrix} = \pi^{-1} Z_o K^{-1} p \quad (41)$$

Here, K is the matrix containing the intrinsic parameters of the disparity camera, Z_o is the depth captured by the disparity camera in the optical frame, and π is a homogeneous transformation matrix from the optical frame, through the robot body frame, and finally to the world frame. p is a pixel in the disparity image. p = $\begin{pmatrix} u \\ v \\ 1 \end{pmatrix}$ pixels = (u,v)

By the method of implementation, I first we converted the camera frame to the body frame. The values of roll, pitch, and yaw are set to 0, 0.36, and 0.021, respectively. Then, the intrinsic parameters of the camera are defined using the focal lengths image center and a skew factor. These values are combined into a calibration matrix,

$$K = \begin{bmatrix} f_{su} & f_{s\theta} & c_u \\ 0 & f_{sv} & c_v \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 585.05 & 0 & 242.94 \\ 0 & 585.05 & 315.83 \\ 0 & 0 & 1 \\ & & \end{bmatrix}$$

Next, the inverse of the calibration matrix is calculated and a rotation matrix, is defined to convert coordinates from the camera frame to the orientation frame. The orientation matrix was obtained by calculating the product of three individual rotation matrices, R_x, R_y, and R_z, corresponding to rotations about the x-axis, y-axis, and z-axis, respectively.

After this, the rotation matrix to bring the camera frame back to the body frame is obtained by multiplying the inverse of the orientation matrix with the camera to body transformation vector. Using this matrix, the pose of the camera with respect to the body was computed.

To find the body coordinates in the world frame we require to compute

$$\mathbf{Transformation_{world}} = \begin{bmatrix} \cos(\theta) & -\sin(\theta) & 0 & x \\ \sin(\theta) & \cos(\theta) & 0 & y \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

where x, y, and θ are the X, Y and θ values of the robot $Transformation_{world}$ is the resulting transformation matrix.

IV. Results and Discussions

We were provided with two datasets, dataset 20 and dataset 21 We shall discuss the accuracy of the trajectory, occupancy grid and the particle filter SLAM. The trajectories were very accurate and the effects of noise on the particles could be seen as well. The map using just a single state was also good but the map and trajectory obtained when using a particle filer was much better than the single state. The details about the plots and results are shown below

A. Trajectory and Motion Model

Initially the motion model was set up to get the trajectory and check the time sync if it were done accurately. The motion model yielded satisfactory results, with the noise standard deviation set to 0.001. Figure 1(a) displays the trajectory obtained from the motion model with a singular state for dataset 20. Furthermore, Figure 1(b)-(d) illustrates the trajectories' variances obtained with 5, 10, and 20 particles, respectively, while maintaining the same noise level.

Likewise, Figure 2(a) represents the trajectory obtained from the motion model with a singular state for dataset 21, and Figure 2(b)-(d) depict the trajectories' variances obtained with 5, 10, and 20 particles, respectively, while keeping the same noise level.

Overall, the trajectory results obtained were excellent, and the particles with noise were found to be within an acceptable range.

To investigate the impact of noise and particles, an experiment was conducted using zero particles with low noise and five particles with high noise. Figure 3(a)(b) presents the outcomes of this experiment, where Figure 3(a) depicts the trajectory for dataset 20 with five particles and no noise, while Figure 3(b) shows the trajectory for dataset 21 with five particles and no noise. In the absence of noise, the particles do not exhibit any diversity and produce the same trajectory since they go through the same motion model with the same values. On the other hand, Figure 3(c)(d) illustrates the consequences of high noise, where the standard deviation was set to 0.1. As the initial guess of the particles was already incorrect, the motion model continued to introduce errors to this guess, leading to an erroneous trajectory.

B. Occupancy map

I was able to make the occupancy map from the dataset 20 and 21. The log odds used for making the maps was log(4).Figure 4(a)(c) shows the occupancy grid made from the first Lidar scan. Figure 4(b)(d) show the completed occupancy grid for dataset 20 and 21 respectively. We used only 1 state here and no particles were used in the making of this map. As we can see from both the maps certain places the walls are of high density, have more black color, this is because the log odds for those places have increased after repeated laser scans on those points. Giving high probability that they are a wall or an obstacle. We can also notice that the hallway is at an angle and is not completely straight, one of the reason for this is drift from the motion model. Drift occurs when there are errors in the robot's motion estimation or sensor readings, which accumulate over time and cause the robot to gradually move away from its intended path. When drift occurs, the robot's position estimate may not align with the actual location of the robot, causing the occupancy grid to be incorrect.

To address the issue of drift, a better estimation of the robot pose is necessary. Despite this challenge, both generated occupancy maps were highly effective in comprehending the mapped area.

C. Particle filter

Once the single state mapping and trajectory was done, particle filter was implemented. For this these hyper parameters were tuned according to my work

- Noise Level: The Gaussian noise added to each particle was selected based on performance, if the noise was too low the particles would not have enough variation and if the noise was too high it would give incorrect results. For the results shown I implemented a noise level std of 0.01.

- Number of particles: We also have to decide on the number of particles to be used, the more we use the better distribution we have at having the best particle for state prediction, but this comes at a computational expense. For my results I used particle number as 60.

Figure 5(a)(c) show the map along with the trajectory as obtained from the particle filter for dataset 20 and 21 respectively. A noticeable improvement in the map's quality is observed as compared to the previous map, as it appears smoother and has more clearly defined boundaries. Also the trajectory that is overlay ed on the map is not as smooth as we saw earlier, it has some high frequency fluctuations, this is because now that we are using the particle filter it can happen at the best state which is used for trajectory generation, varies in the fact it can be particle 1 at time 1 can be particle 40 at time 2 and as the particles have noise in them the trajectory is a little less smooth than what we got previously. We can see the map is in a very great form and can clearly see the boundaries and free space.

In Figure 5(b)(d) you can see the map generated for dataset 20 and 21 respectively.

Figure 7 and 8 have been provided to show the evolution of the map and the trajectory as time progresses. I have also created gifs that will be better to show this phenomenon.
For Dataset 20 the mapping and trajectory GIF can be found HERE

For Dataset 21 the mapping and trajectory GIF can be found HERE

Figure 7-10 shows the evolution of the map and the trajectory with time for dataset 20 and 21

D. Texture Map

The texture map generated for data sets 20 and 21 produced visually impressive results. The map was able to accurately capture the 3D structure of the environment, providing a detailed representation of the floor in the scene. Figure 6(a)(c) shows the resulting texture map for dataset 20 and 21. And figure 6(b)(d) shows the texture map and underlying trajectory on the map. The map is made transparent so that the trajectory could be visualized on it. It is observed that when the robot moves with less deviation aka straight and not many turns the texture map obtained is quite good and continuous. Whereas when the robot had much more of an angular trajectory the texture map is not continuous and has breaks, this also has to do with the fact, the time sync is not perfect and when the robot rotates the environment changes at a higher rate than the map what the algorithm is able to make. Additionally, the quality of the estimated distortion coefficients may also be a factor. The distortion coefficients are used to correct for lens distortion, and if these coefficients are not accurately estimated, it can lead to inaccurate texture mapping. Overall the texture map is quite satisfactory and shows the texture of the floor to a high extent.

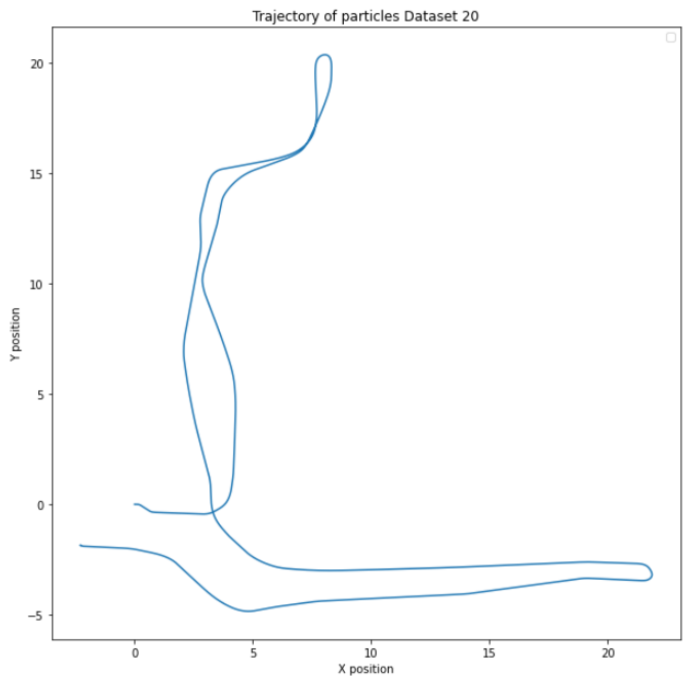
I have also created the texture map progression with time
For Dataset 20 the texture map GIF can be found HERE

For Dataset 21 the texture map GIF can be found HERE

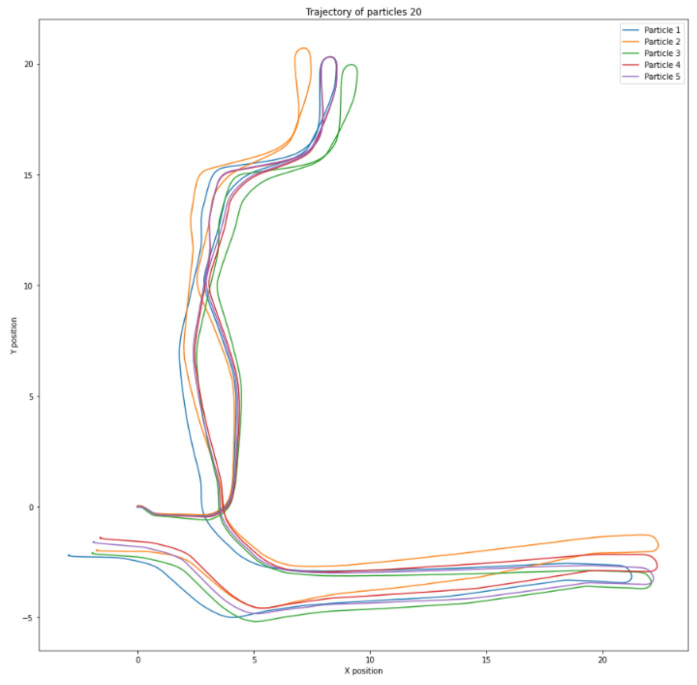
Figure 11-14 shows the evolution of texture map with time for dataset 20 and 21

E. Conclusion

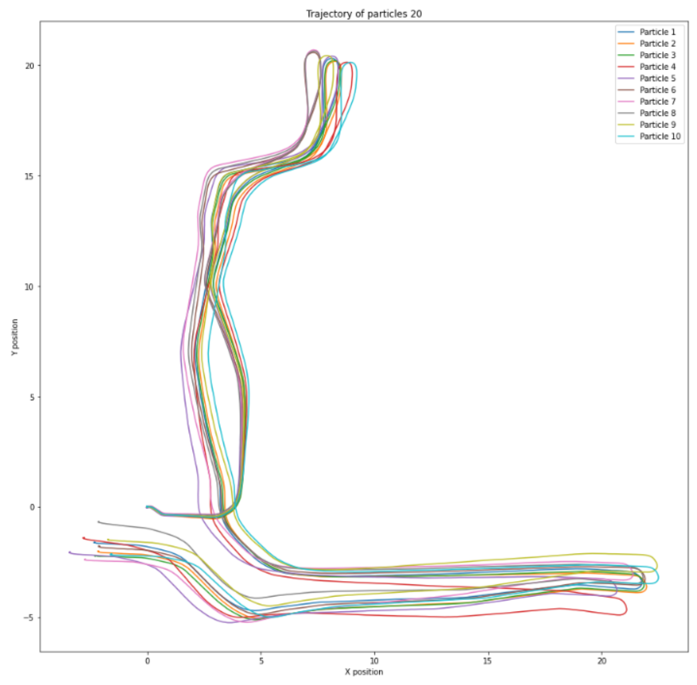
In conclusion, this paper presents a comprehensive approach for simultaneous localization and mapping of a mobile robot. The combination of IMU, rotary encoders, and lidar sensor data provides a robust estimation of the robot's position and orientation. The particle filter algorithm further improves the accuracy of the estimated trajectory and occupancy grid, reducing the effects of noise and drift. The texture mapping with Kinect data provides a visualization of the environment, highlighting the ground plane. Although some defects and limitations were observed, this work demonstrates the potential of the proposed approach in real-world applications.



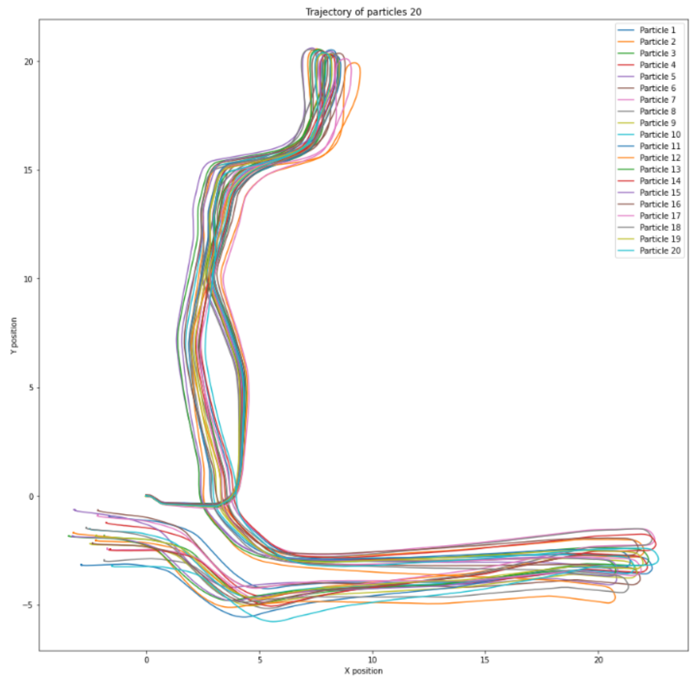
(a) Trajectory with singular state



(b) Trajectory with 5 particles

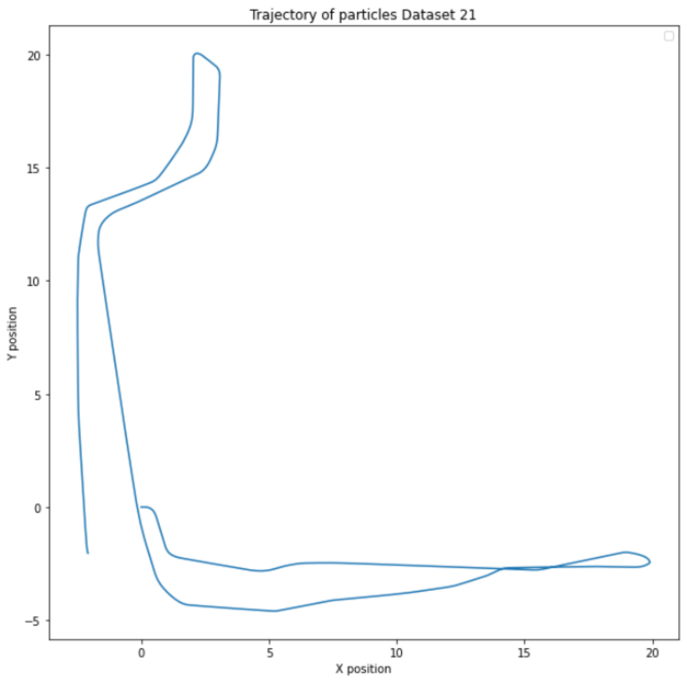


(c) Trajectory with 10 particles

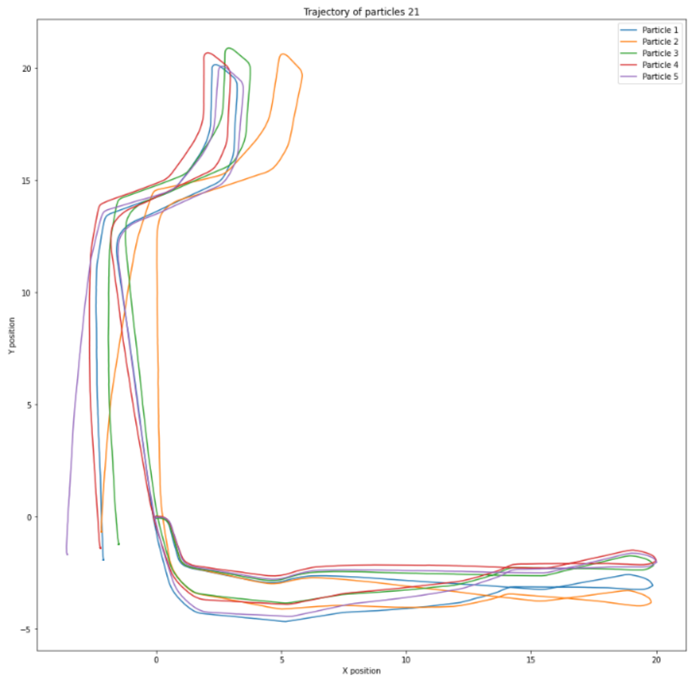


(d) Trajectory with 20 particles

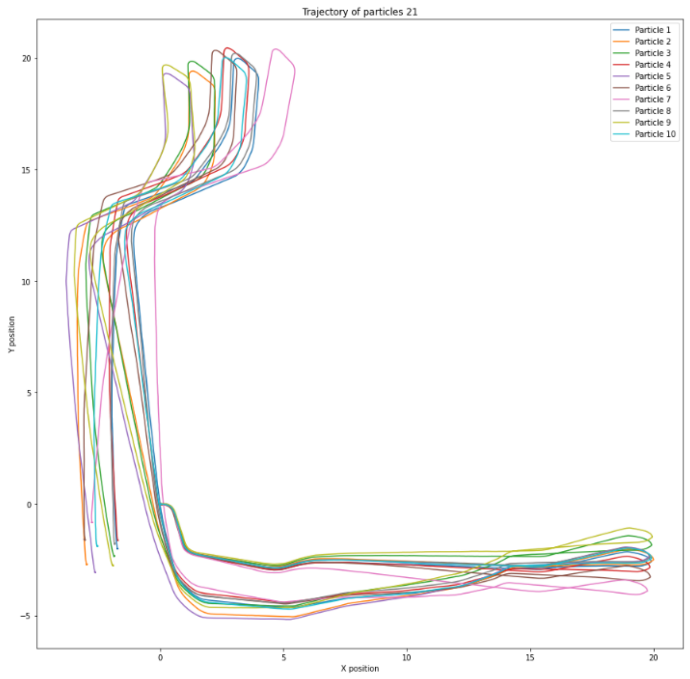
Figure 1: Dataset 20 trajectories with different particle numbers



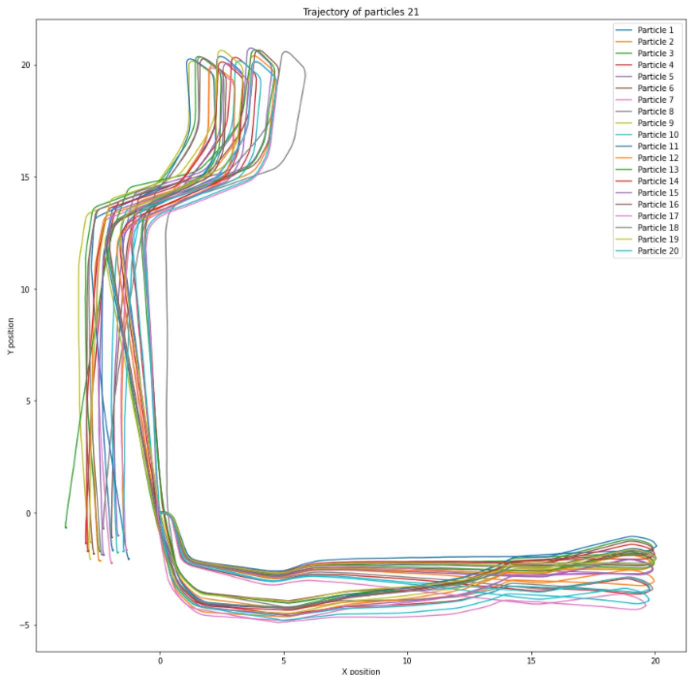
(a) Trajectory with singular state



(b) Trajectory with 5 particles

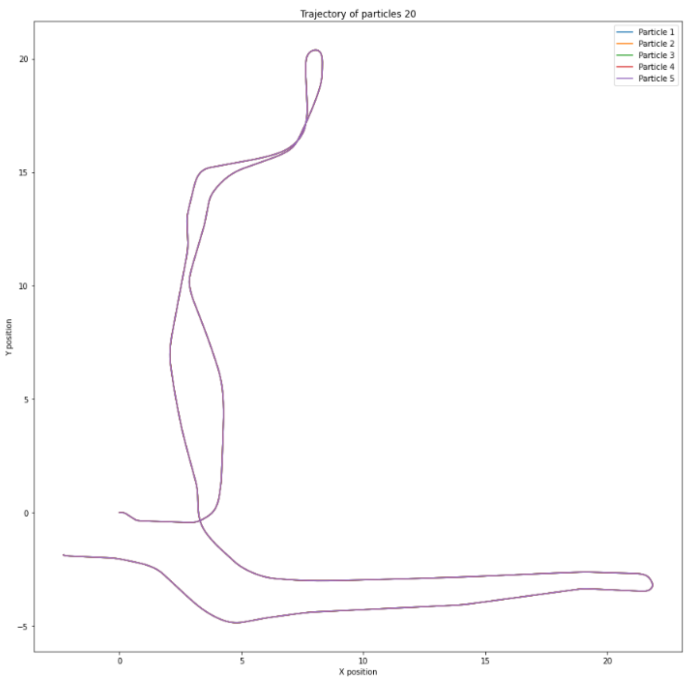


(c) Trajectory with 10 particles

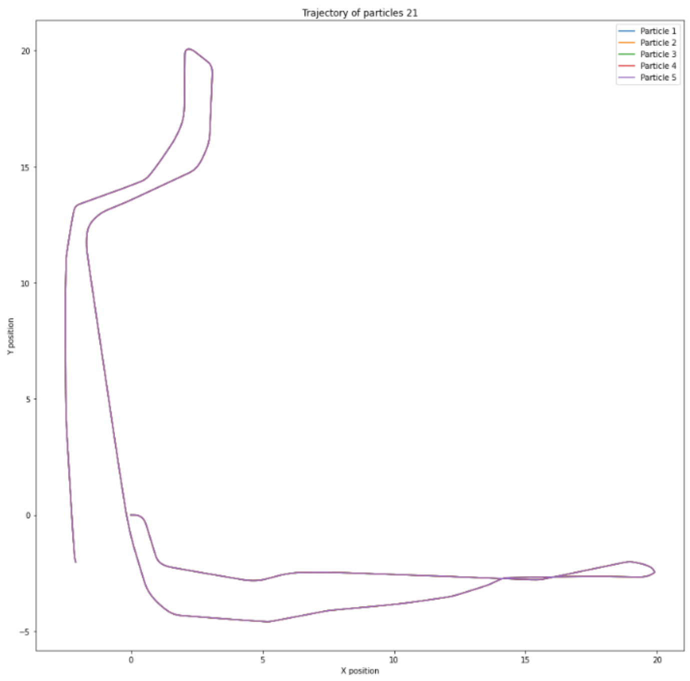


(d) Trajectory with 20 particles

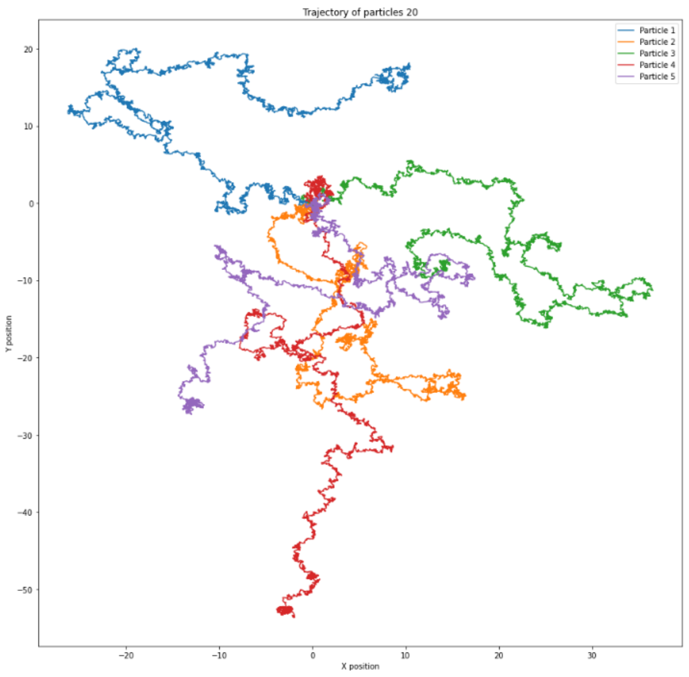
Figure 2: Dataset 21 trajectories with different particle numbers



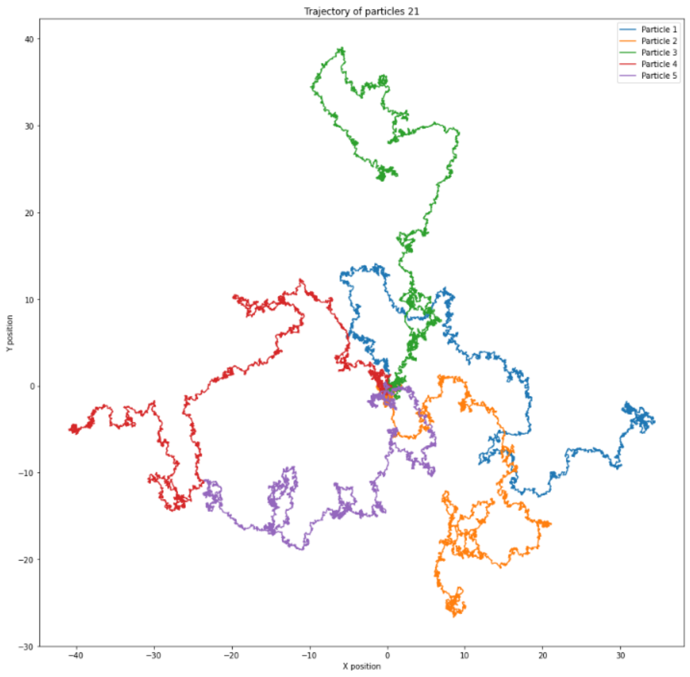
(a) Dataset 20: Trajectory with 5 particles no noise



(b) Dataset 21: Trajectory with 5 particles no noise

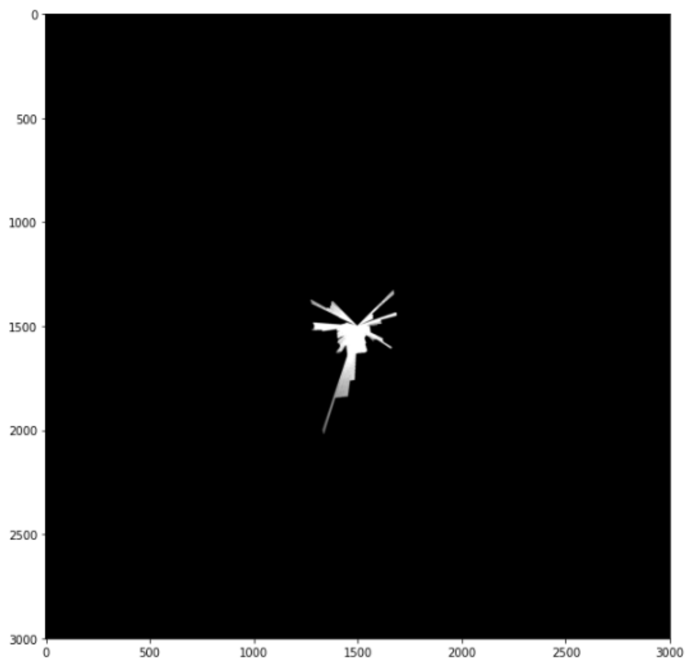


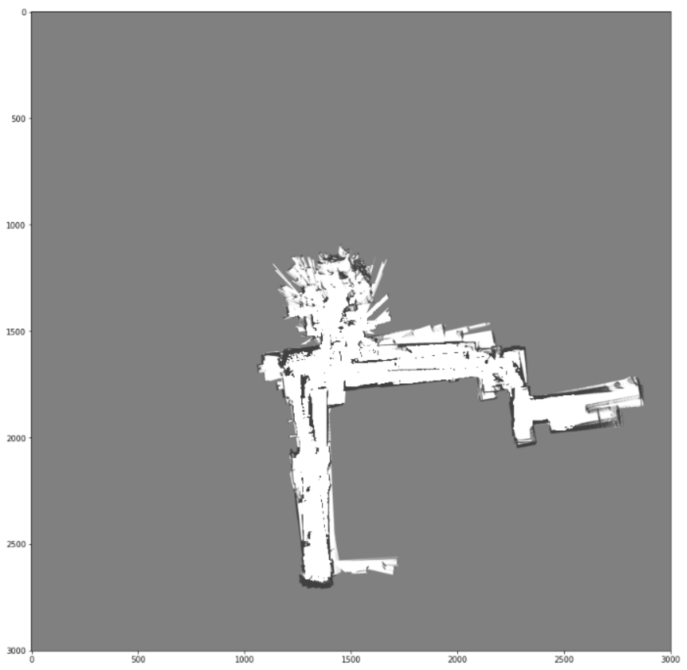
(c) Dataset 20: Trajectory with 5 particles high noise

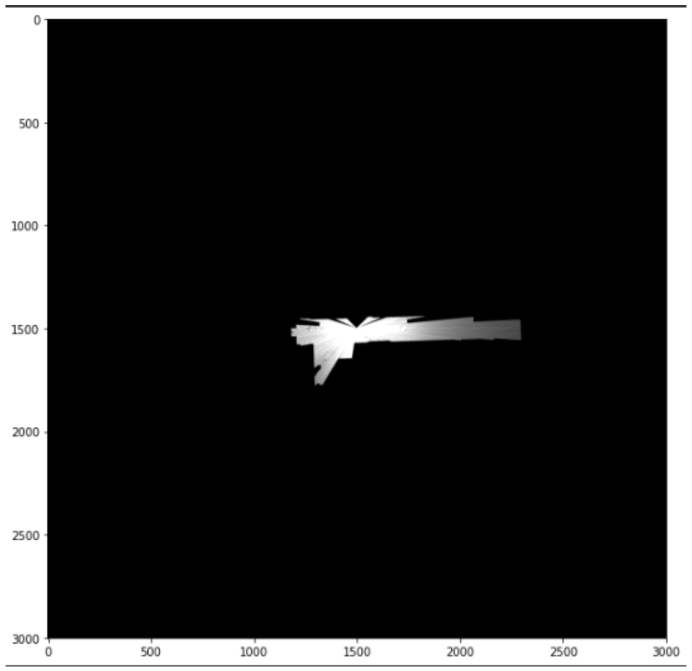


(d) Dataset 21: Trajectory with 5 particles high noise

Figure 3: Effects of noise level on trajectories


(a) Dataset 20: Initial Map Frame


(b) Dataset 20: Occupancy grid


(c) Dataset 21: Initial Map frame

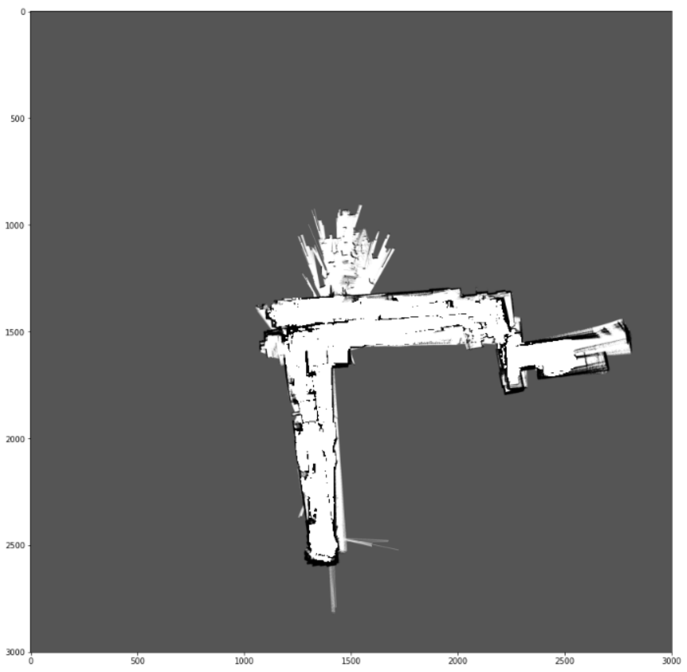
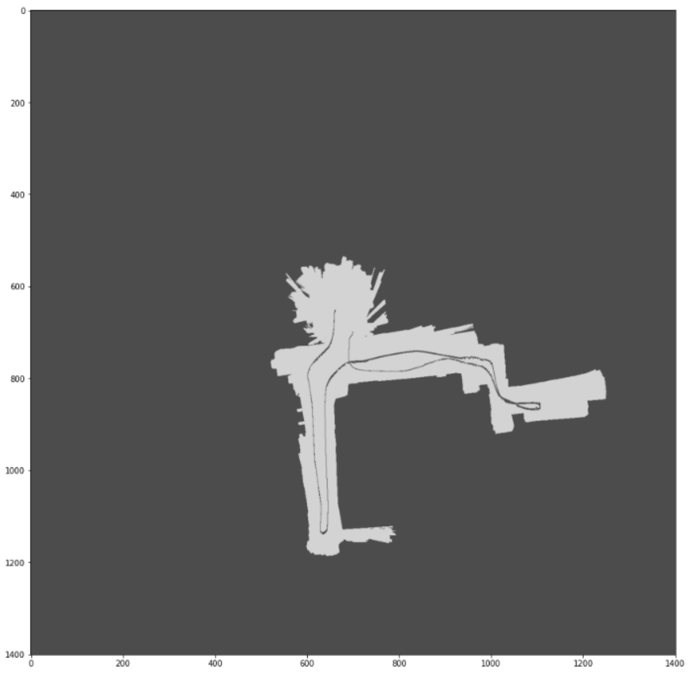
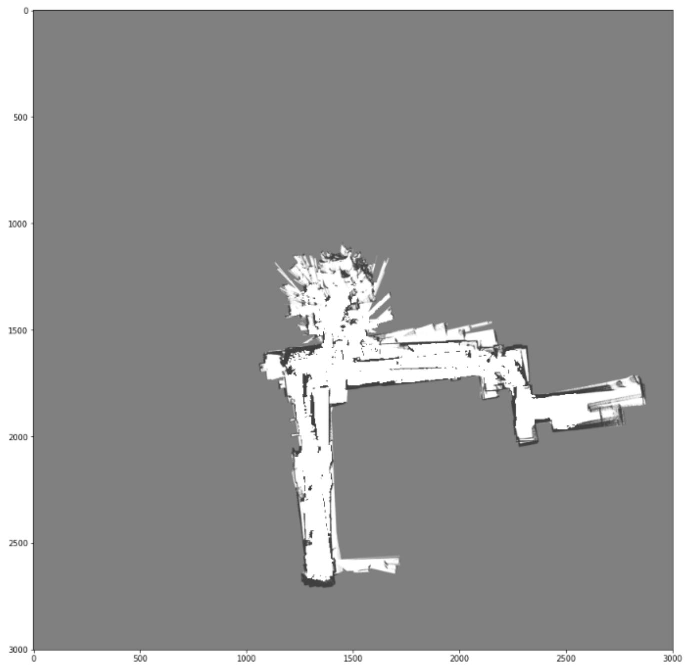
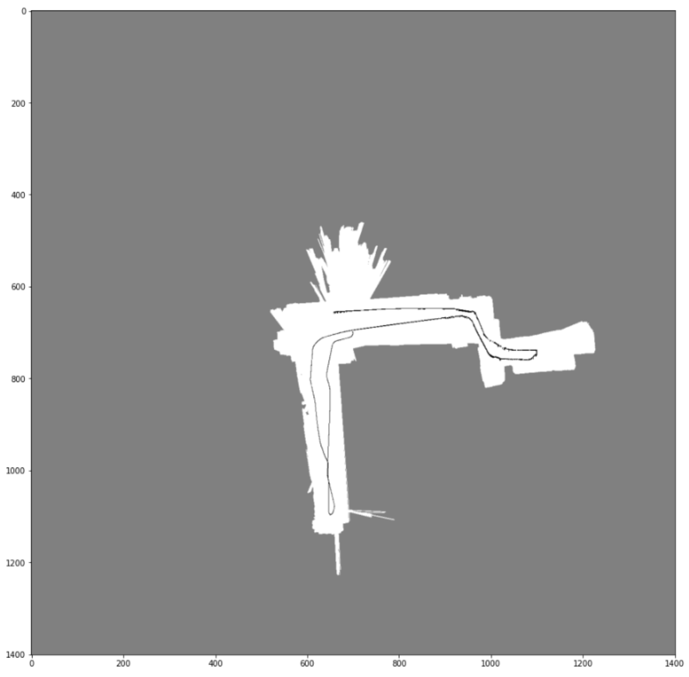

(d) Dataset 21: Occupancy Grid

Figure 4: Initial Maps from laser data


(a) Dataset 20: Map and Trajectory, N=60


(b) Dataset 20: Map with a singular state


(c) Dataset 21: Map and Trajectory N=60

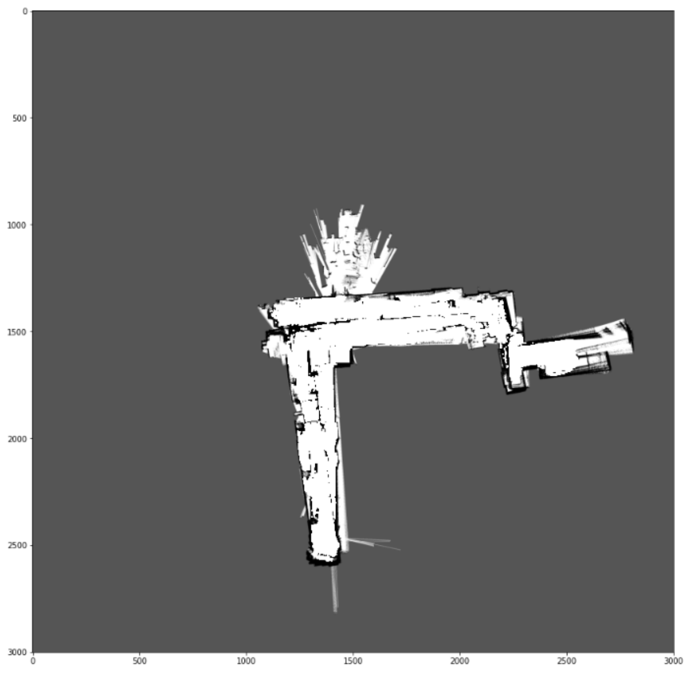
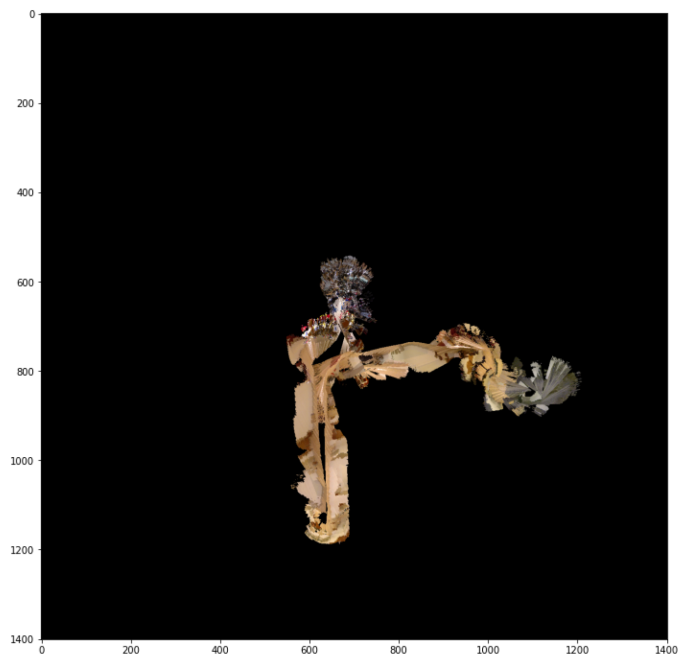
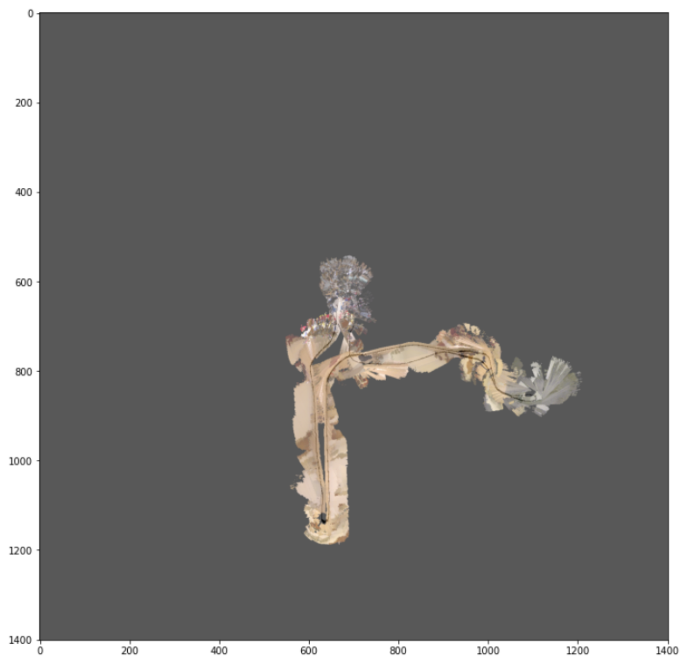
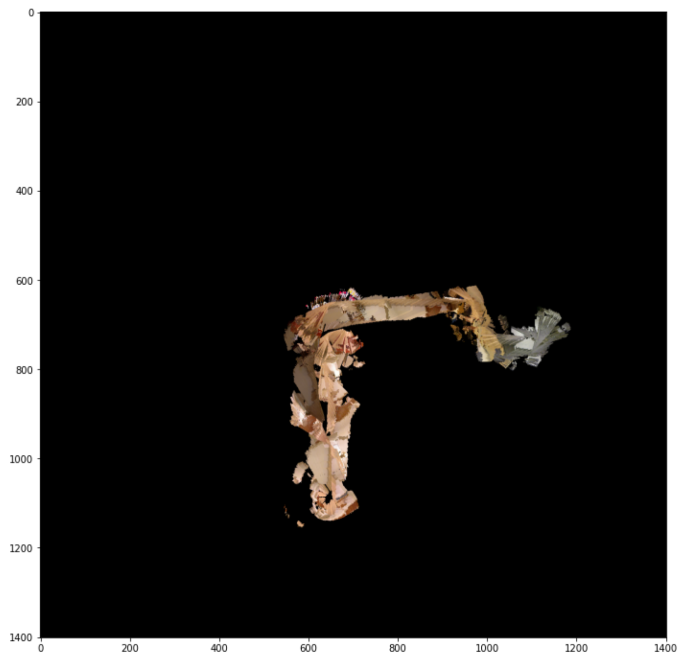

(d) Dataset 21: Map with a singular state

Figure 5: Comparison between maps from particle filter and singular state


(a) Dataset 20: Texture Map


(b) Dataset 20: Texture Map with trajectory


(c) Dataset 21: Texture Map

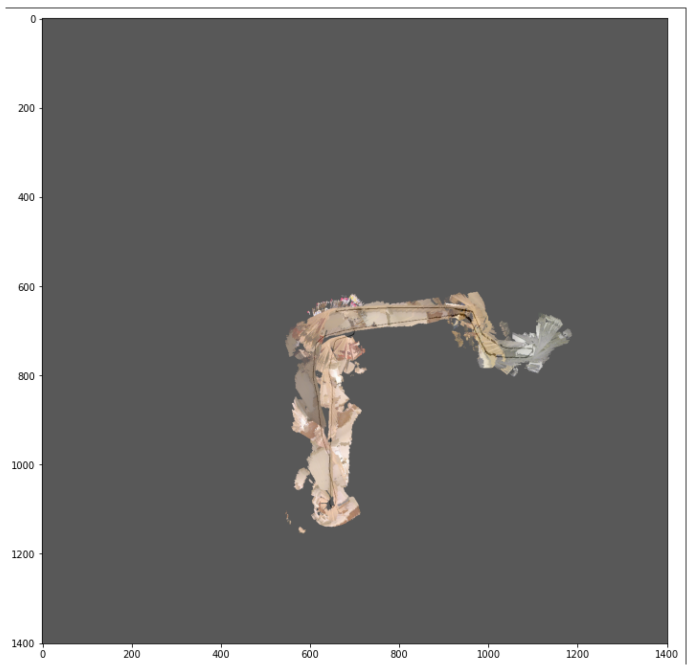
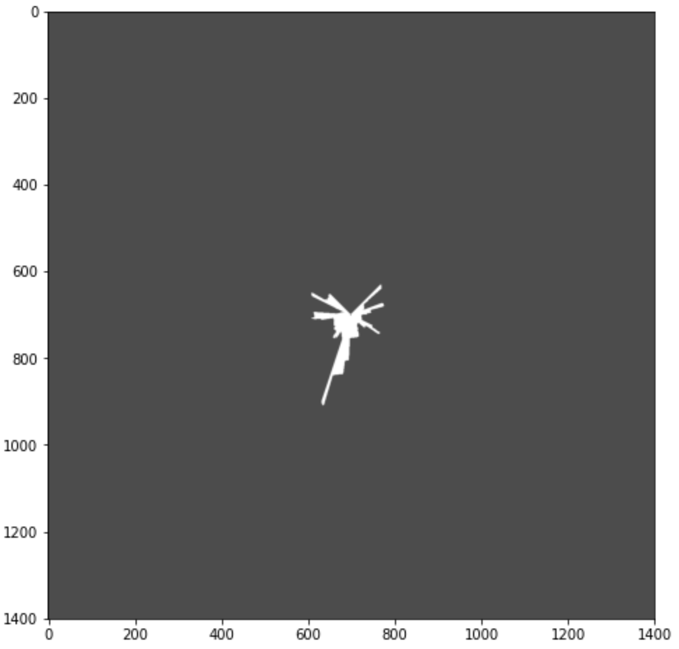
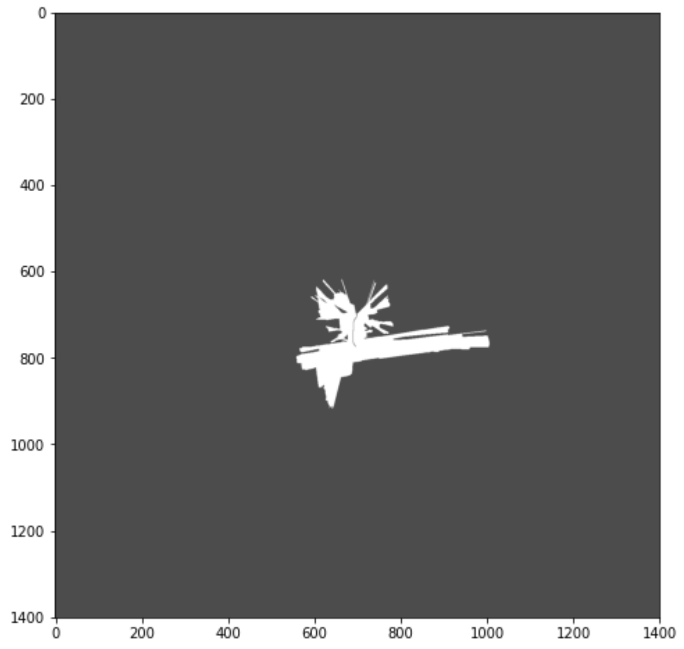

(d) Dataset 21: Texture Map with Trajectory

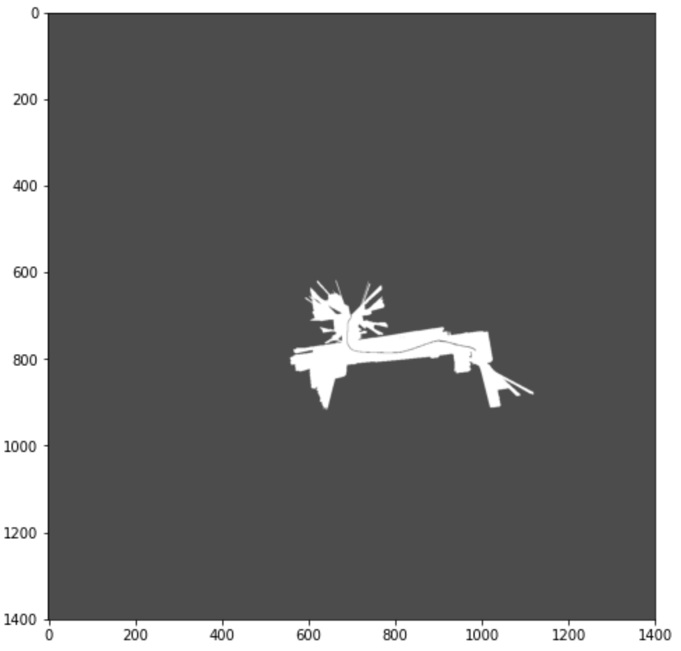
Figure 6: Final Texture map with trajectory



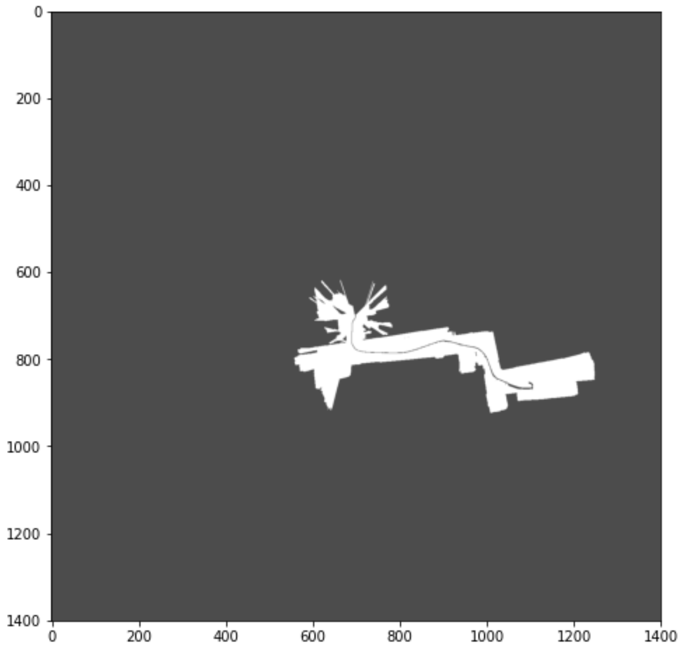
(a) Dataset 20: Map from lidar frame 200



(b) Dataset 20: Map from lidar frame 800

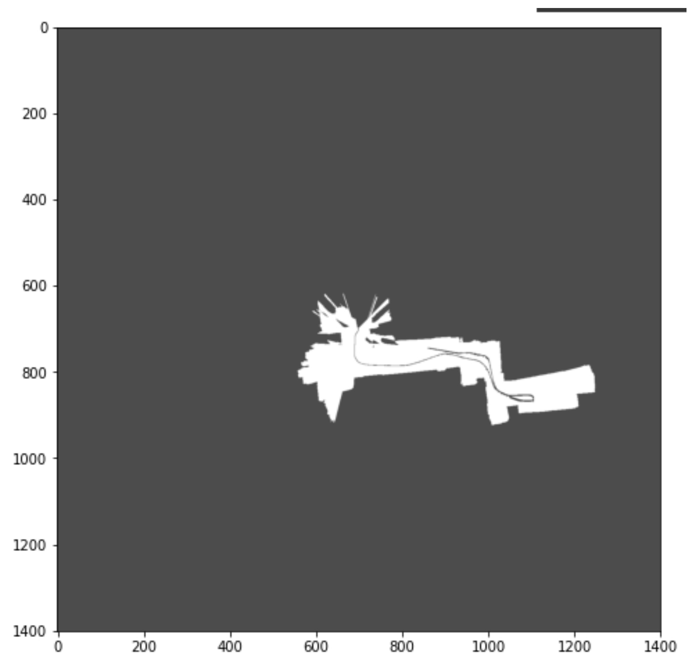


(c) Dataset 20: Map from lidar frame 1400

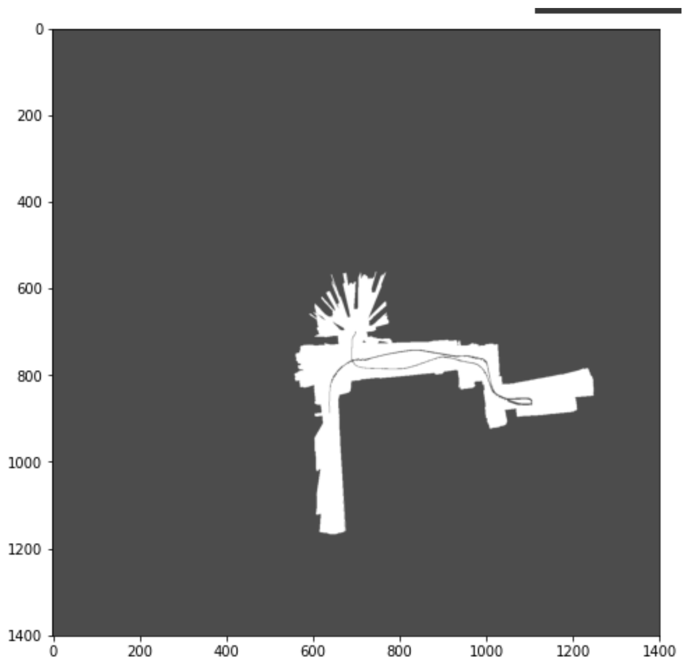


(d) Dataset 20: Map from lidar frame 1800

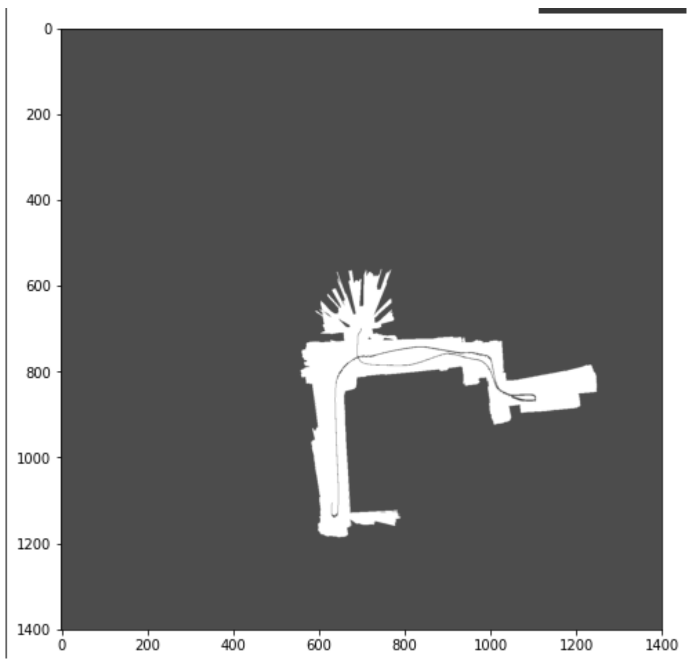
Figure 7: Final mapping and trajectory evolution over time



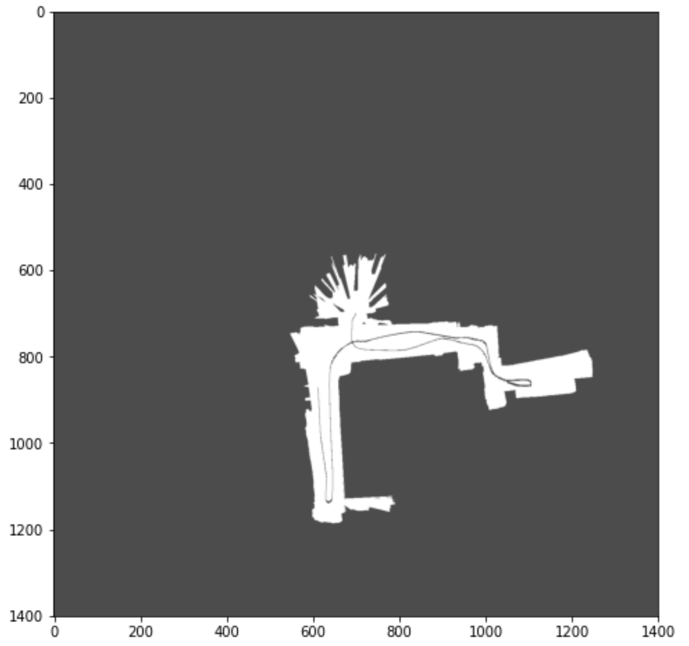
(a) Dataset 20: Map from lidar frame 2400



(b) Dataset 20: map from lidar frame 3000

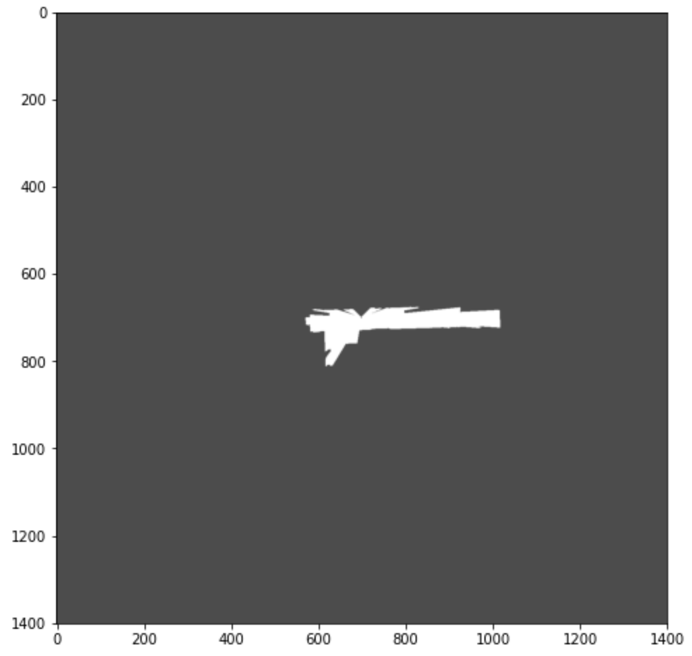


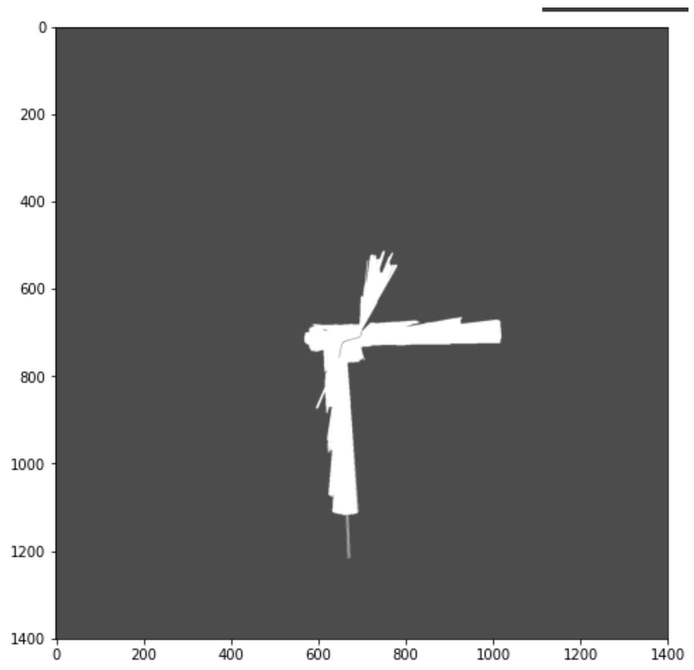
(c) Dataset 20: Map from lidar frame 3600

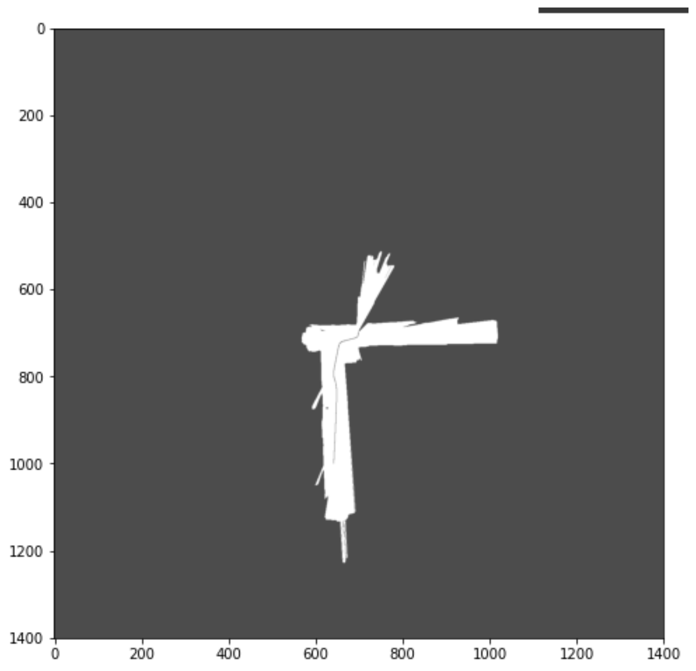


(d) Dataset 20: Map from lidar frame 4800

Figure 8: Final mapping and trajectory evolution over time


(a) Dataset 21: Map from lidar frame 200


(b) Dataset 21: Map from lidar frame 800


(c) Dataset 21: Map from Lidar frame 1400

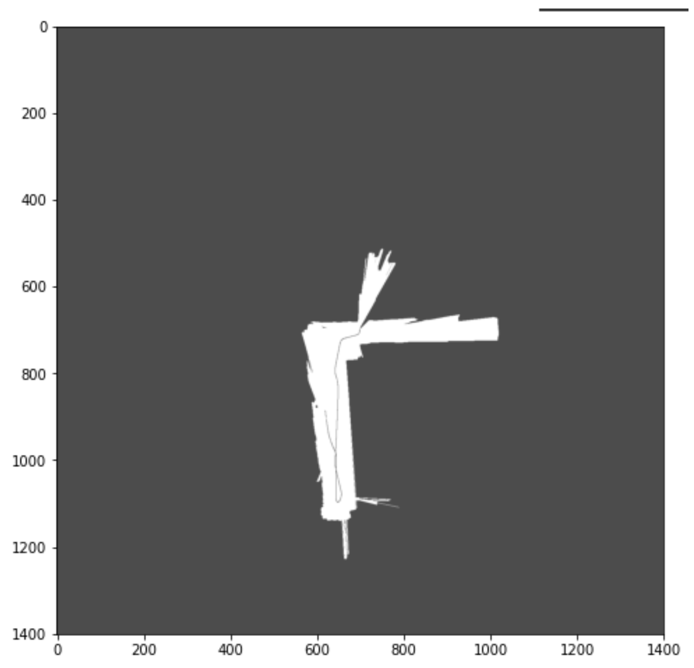
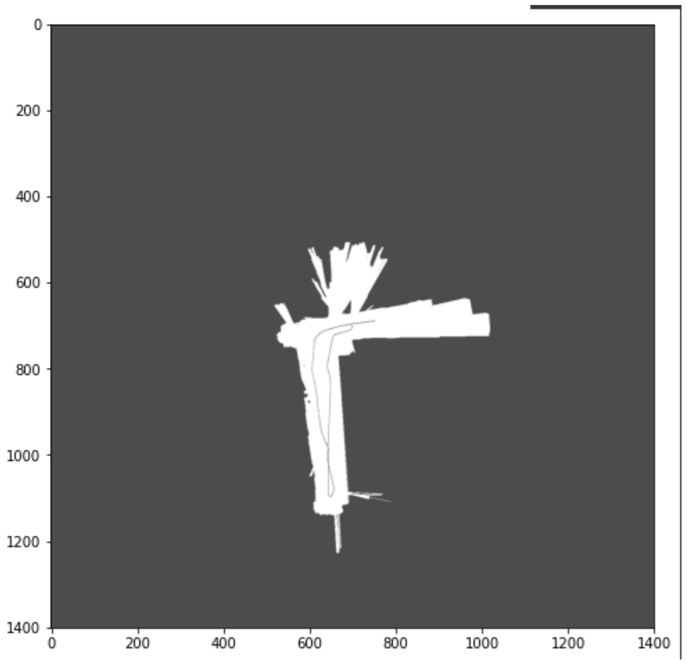
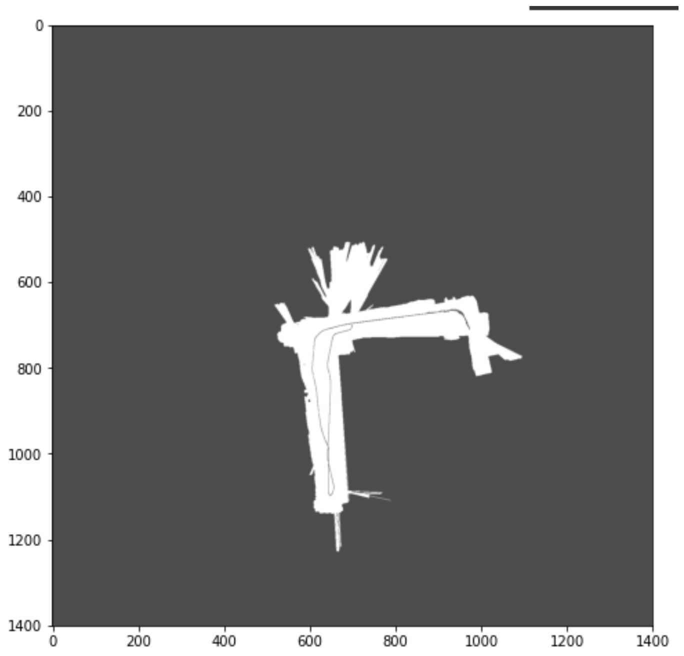

(d) Dataset 21: Map from lidar frame 2200

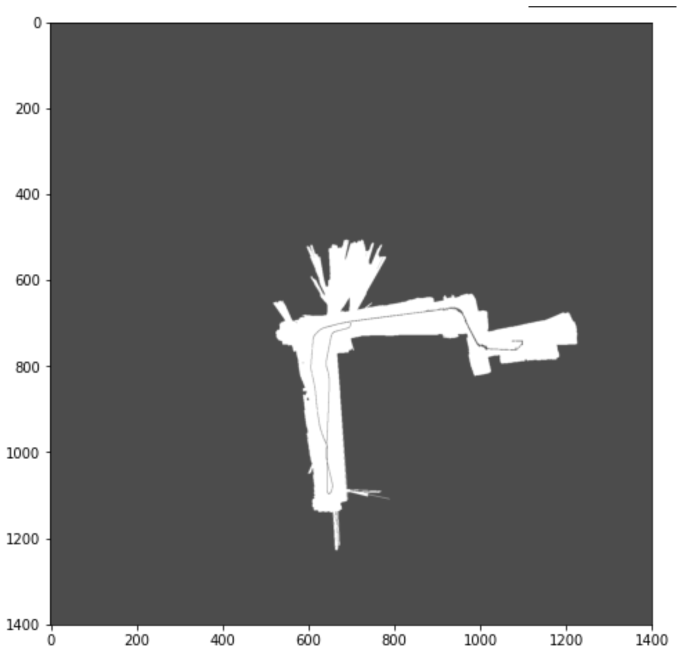
Figure 9: Final mapping and trajectory evolution over time



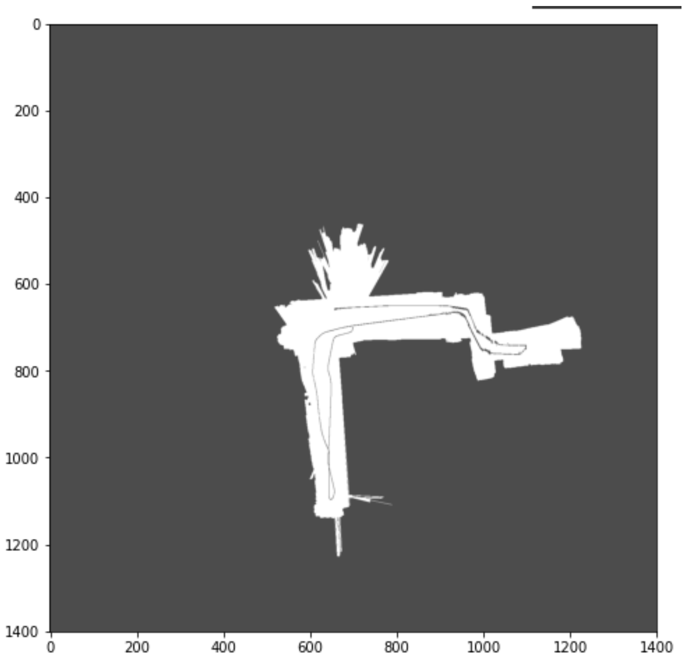
(a) Dataset 21: Map from Lidar frame 2800



(b) Dataset 21: Map from Lidar frame 3200

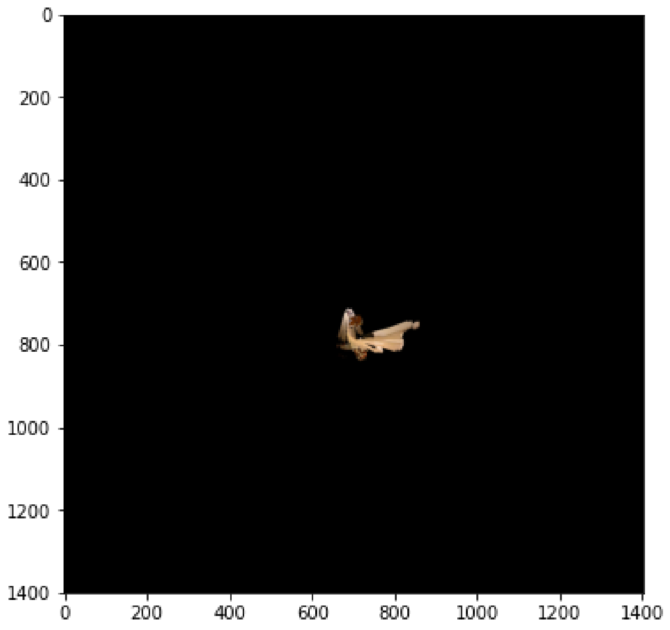


(c) Dataset 21: Map from Lidar frame 3600

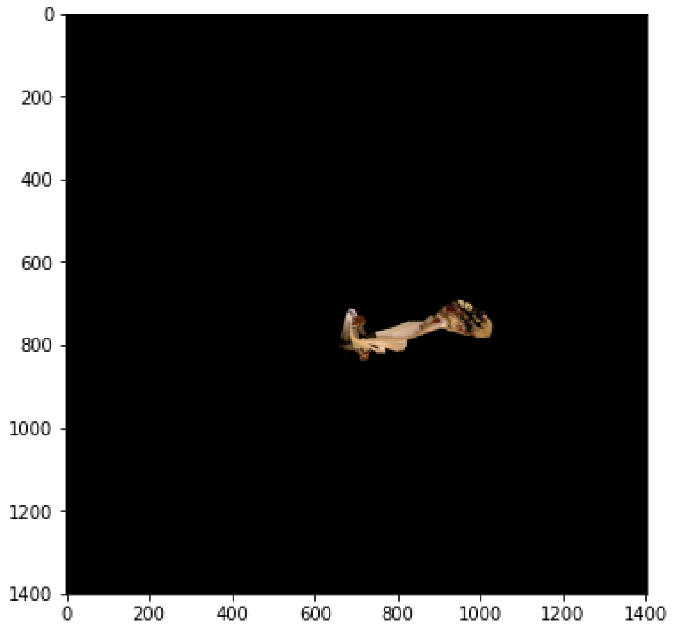


(d) Dataset 21: Map from Lidar frame 4600

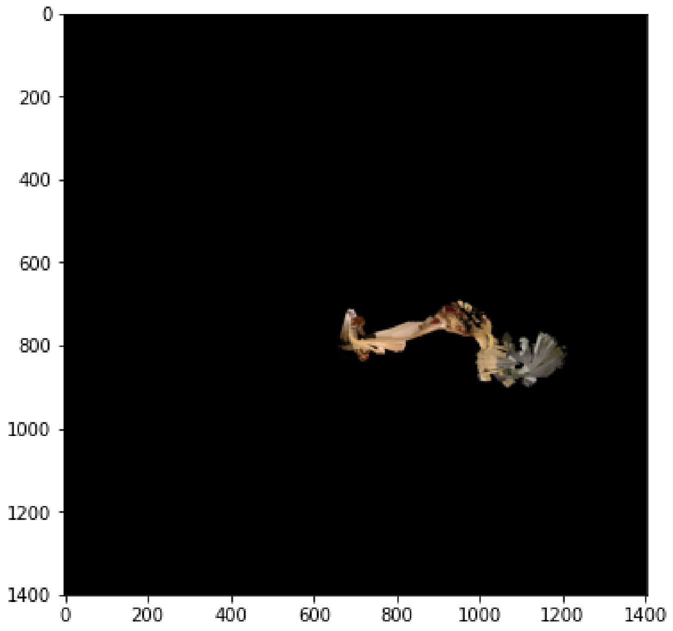
Figure 10: Final mapping and trajectory evolution over time



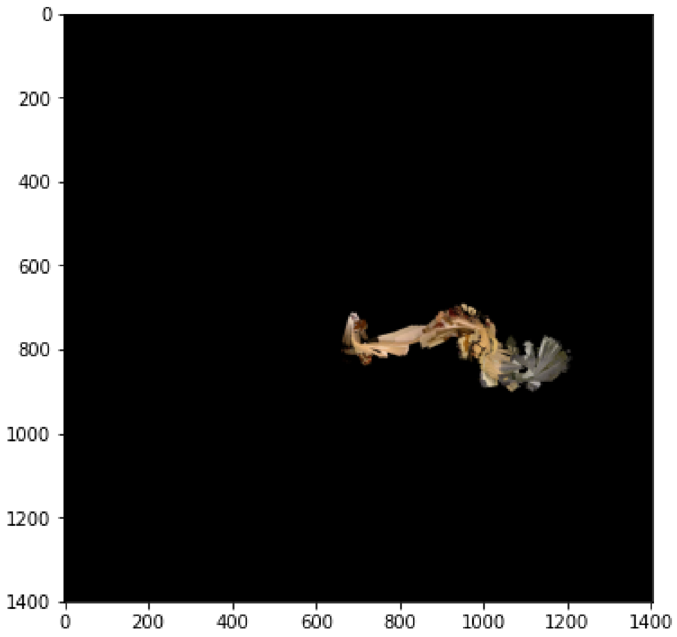
(a) Dataset 20: Texture Map from lidar frame 600



(b) Dataset 20: Texture Map from lidar frame 1000

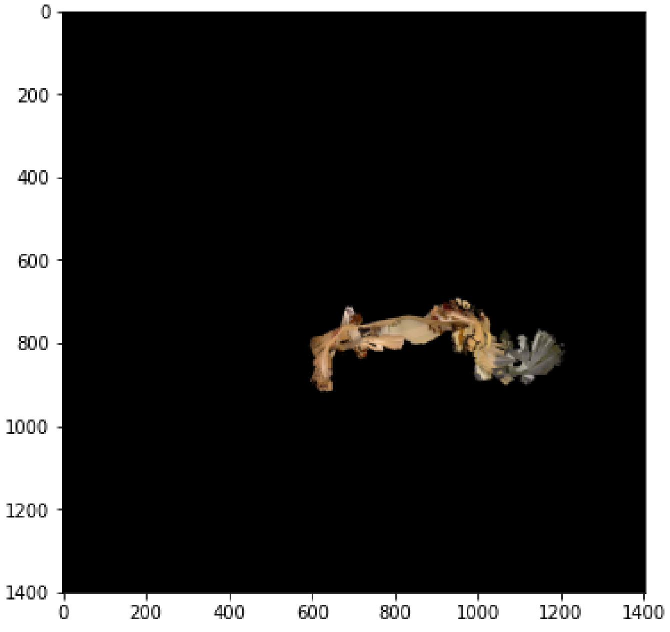


(c) Dataset 20: Texture Map from Lidar frame 1400

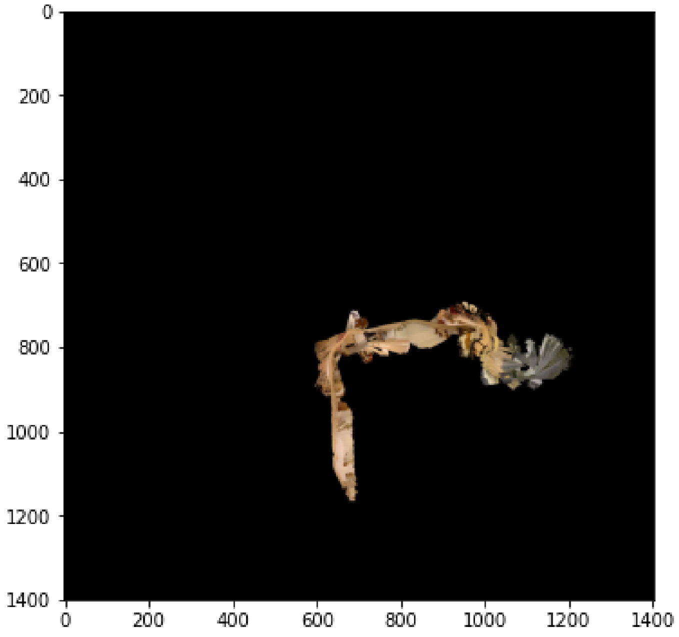


(d) Dataset 20: Texture Map from Lidar frame 1800

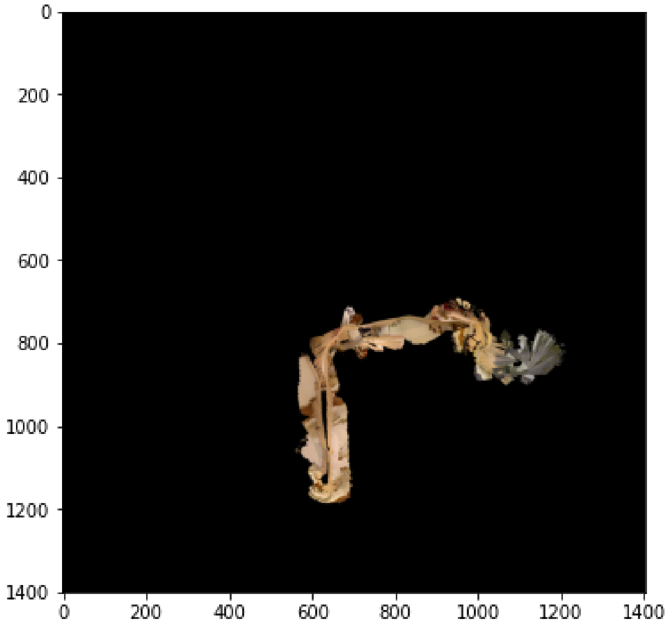
Figure 11: Final Texture map evolution over time



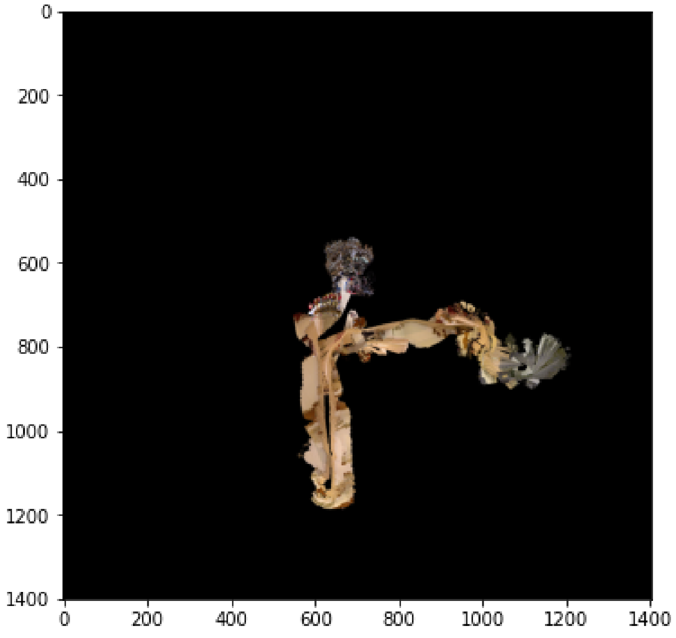
(a) Dataset 20: Texture Map from Lidar frame 1600



(b) Dataset 20: Texture Map from Lidar frame 3000

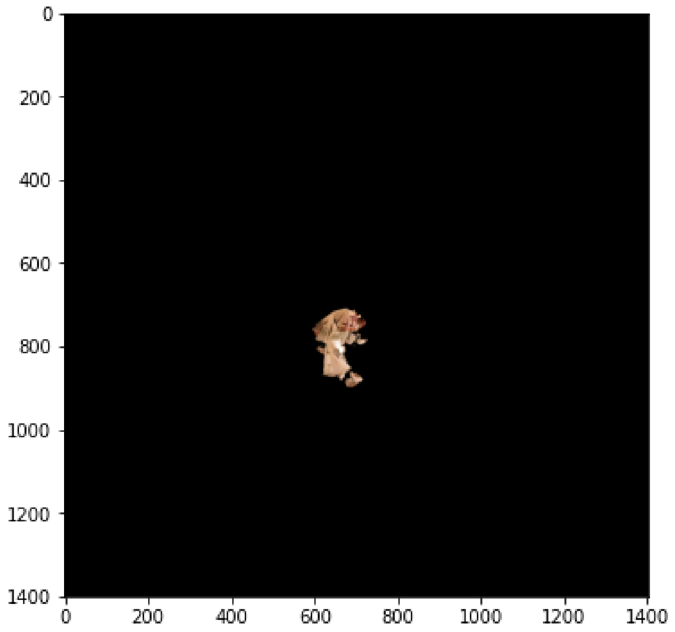


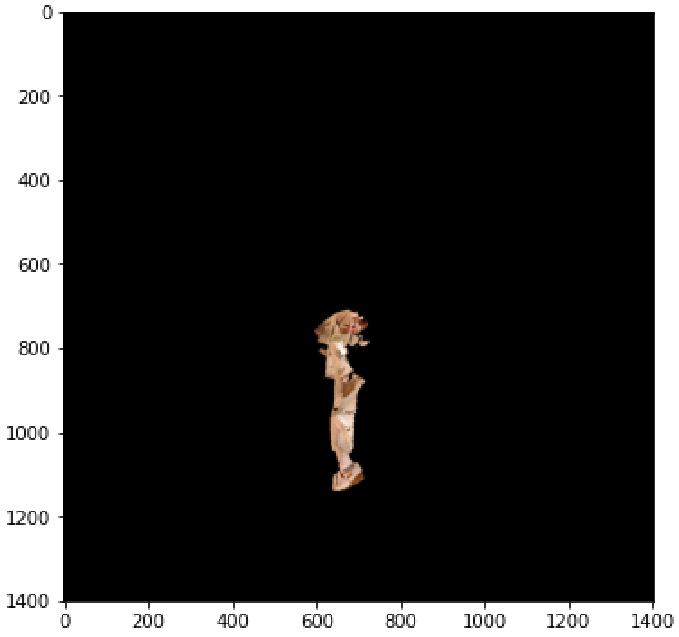
(c) Dataset 20: Texture Map from Lidar frame 3400

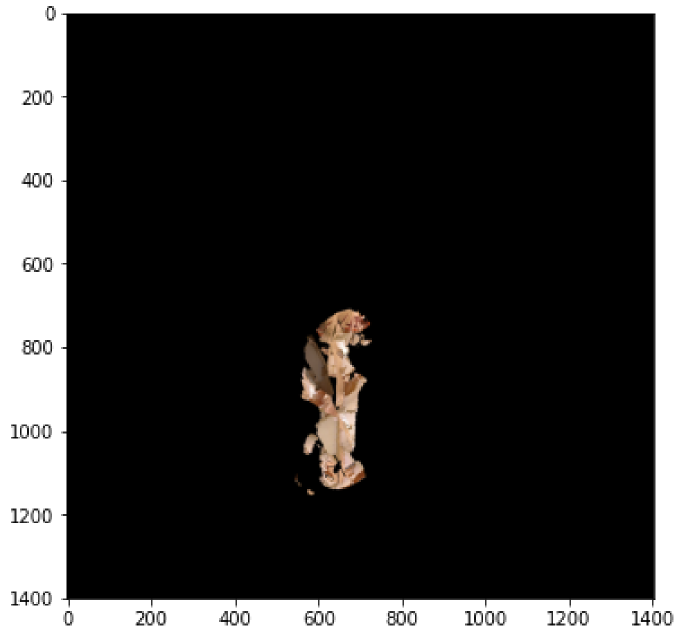


(d) Dataset 20: Texture Map from Lidar frame 4000

Figure 12: Final Texture Map evolution over time


(a) Dataset 21: Texture Map from Lidar frame 600


(b) Dataset 21: Texture Map from Lidar frame 1200


(c) Dataset 21: Texture Map from Lidar frame 1800

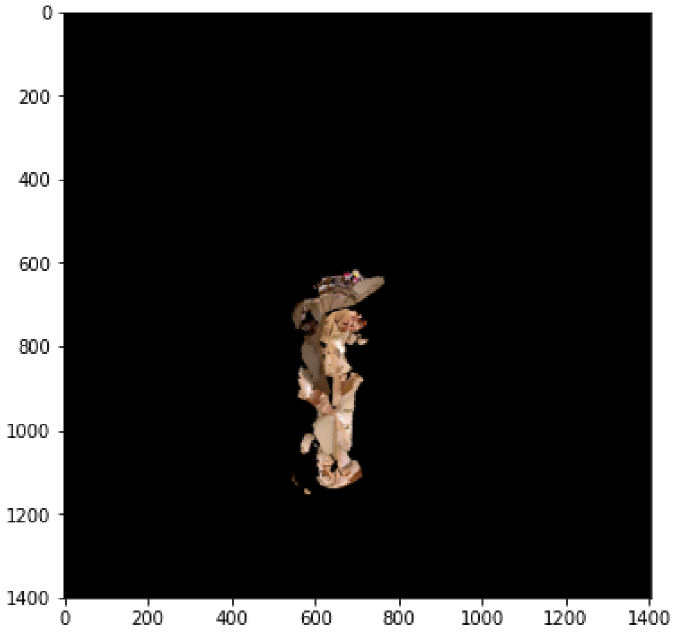
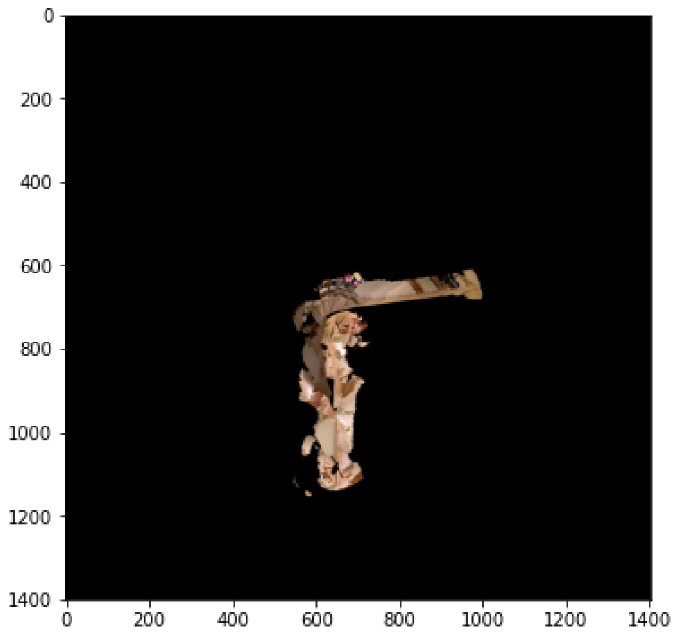
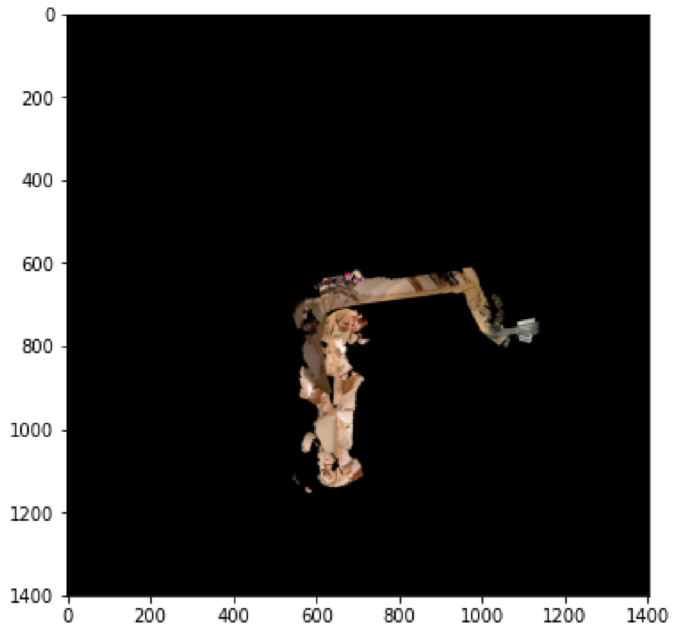

(d) Dataset 21: Texture Map from Lidar frame 2400

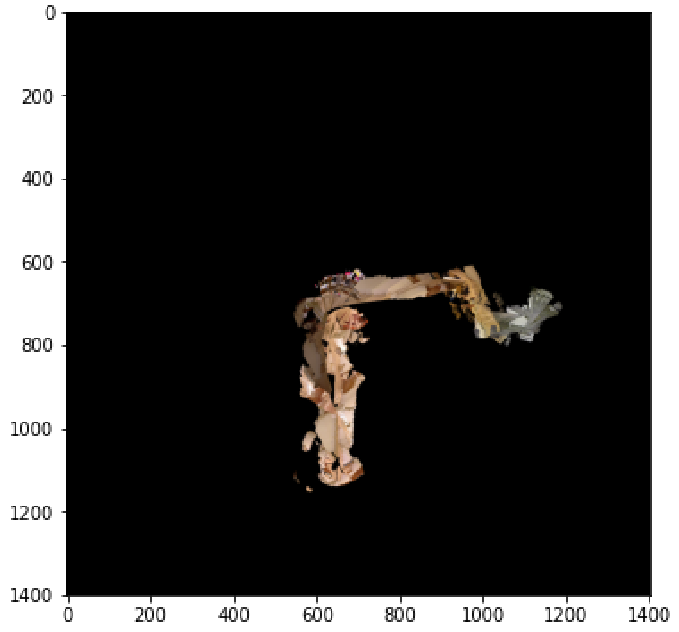
Figure 13: Final Texture Map evolution over time



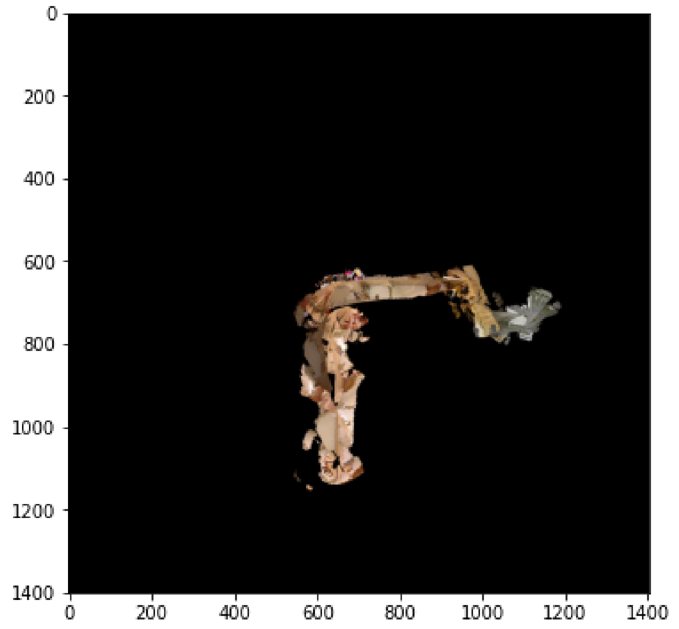
(a) Dataset 21: Texture Map from lidar frame 2800



(b) Dataset 21: Texture Map from lidar frame 3000



(c) Dataset 21: Texture Map from lidar frame 3600



(d) Dataset 21: Texture Map from lidar frame 4200

Figure 14: Final Texture map evolution over time