Simultaneous Localization and Mapping, and Texture Mapping using a Particle Filter.

Imoleayo Abel

I. Introduction

In many robotic systems, it is necessary to have a robot map an unfamiliar environment and locate itself using the created map. In addition to localization and mapping, it is also often important for the robot to identify and place other items in it's environment on the map. Example applications where these abilities are necessary include planetary exploration, or disaster relief scenarios where a robot would be deployed to access an environment inaccessible to humans. In these applications, the robot would need to create a map of a planet or disaster area for example, and then locate itself on the map, and also identify features in the environment and their respective locations on the map.

In this paper, we describe an implementation of a particle filter for simultaneous localization and mapping (SLAM), and texture mapping by the humanoid robot shown in Fig. 1. The robot has a lidar range sensor for detecting distances from objects, a Kinect™ sensor for generating depth and RGB images, inertial measuring units, and angle sensors for measuring the angles of the head and neck motors.

Notation: While the models used in this paper are discrete time models, we use the notation $x(t), x(t+1), \ldots$ instead of the standard $x_t, x_{t+1}, \ldots$ for clarity because most of the variables used already have subscripts.

II. Problem Formulation

Given a set of distance measurements $z(t) \in \mathbb{R}^{1\times \texttt{nbeams}}$ from the lidar at pose ${}_wT_l(t) \in SE(3)$ in the world frame, a set of head and neck angles $\varphi(t), \theta(t) \in (-\pi, \pi]$, a set of depth images $I_d(t) \in \mathbb{R}^{1\times \texttt{ndepth}}$ and RGB images $I_{rgb}(t) \in \mathbb{R}^{3\times \texttt{npixels}}$ from the Kinect™ sensor, as well as the intrinsic and extrinsic parameters of the cameras, the goal of this project is to use a particle filter to:

1) **SLAM**: Simultaneously generate an occupancy grid $\{m_i\}_{i=1}^{\texttt{ncells}}$ of the world where

$$m_i = \begin{cases} 1, & \text{if cell } i \text{ is occupied.} \\ 0, & \text{if cell } i \text{ is free.} \end{cases} \tag{1}$$

and to estimate the robot pose ${}_wT_b(t) \in SE(3)$ in the world frame over time.

2) **Texture Mapping**: Compute the mapping

$$f : p \to \{1, \ldots, \texttt{ncells}\}, \tag{2}$$

where $p = \{1, \ldots, \texttt{npixels}\} \times \{0, \ldots, T\}$ is the set of pairs (j, t) of RGB camera pixel locations and time, such that $f(j, t)$ is the occupancy grid cell location corresponding to pixel location j at time t.

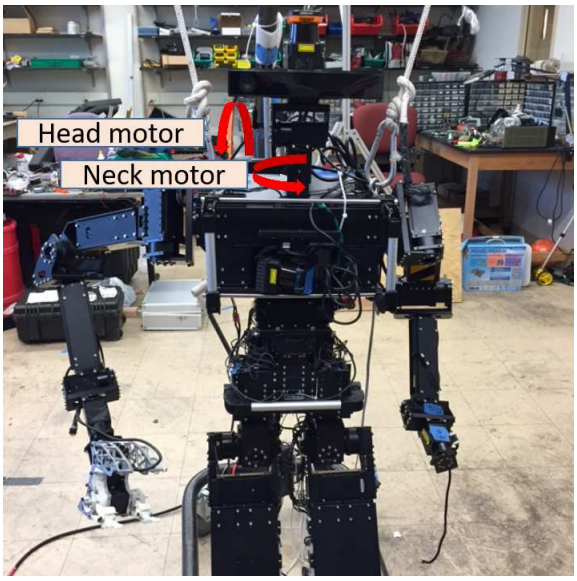

Fig. 1: Humanoid robot showing lidar range sensor and Kinect™ sensor for depth and RGB image capturing mounted on it's head.

III. Technical Approach

A. Simultaneous Localization and Mapping

A particle filter with N particles and corresponding weights is used for simultaneous localization and mapping. Each particle $P_{t|t}^k$ in the filter is a rigid transformation ${}_wT_b^k \in SE(3)$ representing the body pose of the robot in the world frame, and the corresponding weight $\alpha_{t|t}^k$ represents the relative likelihood of the particle being the actual robot pose. For this project, the rotation of the robot's body is assumed to be only around the vertical axis and as such the rigid transformations representing the body pose encapsulate just the position and yaw angle. All particles of the filter are passed through an odometry-based prediction motion model and an observation-correlation based update model. At each time step, the "best" particle is chosen as the closest estimate of the robot's pose. We initialize all N particles to the identity matrix I_4 which implies using the robot's initial pose as the origin of the map and the particle weights are all initializes to $\frac{1}{N}$.

Odometry-based Motion Model (Prediction): For the prediction step, the following model is applied to each of the

particles in the filter to transform forward in time:

$$P^k_{t+1|t} = P^k_{t|t} * T^k_{noise}(t) * T_{control}(t) \tag{3}$$

where T_{noise} is a noise pose constructed from a vector $\begin{bmatrix}\omega^k_x(t) & \omega^k_y(t) & \omega^k_\theta(t)\end{bmatrix}^T$ sampled from a 3-dimensional Gaussian normal distribution $\mathcal{N}(O_{3\times1}, W_{3\times3})$ as follows:

$$T_{noise}(t) = \begin{bmatrix}\cos\omega^k_\theta(t) & -\sin\omega^k_\theta(t) & 0 & \omega^k_x(t)\\ \sin\omega^k_\theta(t) & \cos\omega^k_\theta(t) & 0 & \omega^k_y(t)\\ 0 & 0 & 1 & 0\\ 0 & 0 & 0 & 1\end{bmatrix} \tag{4}$$

and $T_{control}(t)$[1] is given as follows.

$${}_bT_l(t) * {}_wT_l(t)^{-1} * {}_wT_l(t+1) * {}_bT_l(t+1)^{-1} \tag{5}$$

where ${}_wT_l(t)$ is the lidar odometry in the world frame at time t and ${}_bT_l(t)$ is the lidar to body transformation derived from the dimensions of the robot and the head and neck angles at time t. We point out that the control pose $T_{control}(t)$ is the same for all particles at each prediction step while the noise pose re-sampled for each particle and specific to that particle at each time step. This prediction step projects the particles forward in time to $t+1$ and capturing the potential noise in the motion dynamics of the robot. The weights of the particles are projected forward in time from to $\alpha_{t|t}$ to $\alpha_{t+1|t}$ without modification.

Scan Matching Observation Model (Update): After the particles are projected forward in time using the motion model, we update them using their correlation with the observation from the lidar measurements. Specifically, we start by using each predicted particle pose $P^k_{t+1|t}$ to transform the lidar measurements $z(t+1)$ into the world frame and then compute the correlation $corr^k$ between these lidar world coordinates and the occupancy grid at time t using a `mapCorrelation` function. For each particle $P^k_{t+1|t}$, the `mapCorrelation` function shifts the lidar world coordinates over a small range in the $x-$ and $y-$ directions and returns a measure of the correlation of the shifted coordinates with the occupancy grid at time t. The $x-$, $y-$ shift having the largest correlation is applied to the particle $P^k_{t+1|t}$ to get $P^k_{t+1|t+1}$, and the corresponding correlation for the "best" $x-$, $y-$ shift is chosen as $corr^k$. To improve performance of the filter, we also vary the yaw angle θ^k of the particles over a small range and call `mapCorrelation` for each yaw variation. We then use the $x-, y-, \theta-$ shifts with the maximum correlation as the correlation for the particle. The particles are updated as follows

$$P^k_{t+1|t+1} = P^k_{t+1|t} * \begin{bmatrix}\cos\Delta\theta^k & -\sin\Delta\theta^k & 0 & \Delta x^k\\ \sin\Delta\theta^k & \cos\Delta\theta^k & 0 & \Delta y^k\\ 0 & 0 & 1 & 0\\ 0 & 0 & 0 & 1\end{bmatrix} \tag{6}$$

where $\Delta x^k, \Delta y^k, \Delta\theta^k$ are the $x-, y-, \theta-$ shifts with the largest correlation value.

[1]The derivation of this expression is given in the appendix.

Weight Update: Once the particles are updated, we update the particle weights $\alpha^k_{t+1|t}$ using the expression below:

$$\alpha^k_{t+1|t+1} = \frac{\alpha^k_{t+1|t}\overline{corr}^k}{\sum_{j=1}^{N}\alpha^j_{t+1|t}\overline{corr}^j} \tag{7}$$

where $\overline{corr}^k$ is given as

$$\overline{corr}^k = \frac{\exp\left(corr^k\right)}{\sum_{j=1}^{N}\exp\left(corr^j\right)} \tag{8}$$

After the weight updates, we select the "best" particle $P^m_{t+1|t+1}$ defined below as the particle representing the robot's pose at time $t+1$

$$m = \arg\max_k \{\alpha^k_{t+1|t+1}\}_{k=1}^N \tag{9}$$

Particle Resampling: With the "best" particle selected, and the particle and particle weights updated, we resample the particles to avoid particle depletion. To do this, we check that the effective number of particles N_{eff} defined below is larger than a predefined threshold N_{thresh}

$$N_{eff} = \frac{1}{\sum_{j=1}^{N}\alpha^k_{t|t}} \tag{10}$$

If (10) holds, the particles are resampled using the Stratified Resampling algorithm shown below.

Algorithm 1 Stratified Resampling

Input: particle set $\{P^k, \alpha^k\}_{k=1}^N$
Output: resampled particle set

```
j ← 1, c ← α^1
for k = 1...N do
    u ~ U(0, 1/N)
    β = u + (k−1)/N
    while β > c do
        j = j + 1, c = c + α^j
    end while
    add (P^j, 1/N) to new particle set
end for
```

Occupancy Grid Update: To generate the occupancy grid map $\{m_i\}_{i=1}^{\texttt{ncells}}$, we maintain a log-odds map $\{\lambda_i\}_{i=1}^{\texttt{cells}}$ that is updated over time as follows:

$$\lambda_i(t+1) = \lambda_i(t) + \eta_i(t+1)\log(g) \tag{11}$$

where g is a measure of the belief in the lidar measurements defined mathematically as the ratio of the conditional probabilities of observing occupied and free cells and is chosen as a design parameter,

$$g = \frac{p_h(z|m_i=1)}{p_h(z|m_i=0)} \tag{12}$$

and $\eta_i(t)$ is defined as follows

$$\eta_i(t) = \begin{cases} 1, & \text{if } i \text{ observed as occupied at time } t \\ -1, & \text{if } i \text{ observed as free at time } t \end{cases} \quad (13)$$

We initialize the log-odds map to $\lambda_i(0) = 0 \ \forall i$ since the cells are equally as likely to be occupied and free initially and $\log(1) = 0$. To determine the grid cells observed as occupied and free at each time step, we use the best particle $P^m_{t|t}$ to compute the world Cartesian coordinates of the lidar hit points. We then use a Bresenham ray tracing algorithm to find the coordinates of the empty points between the robot and the hit-points. The points are then converted from meters to grid cells based on the resolution of the occupancy grid and the log-odds map updated according to (11). Once the iteration over time is complete, the log-odds map $\{\lambda_i\}_{i=1}^N$ is converted back to the occupancy grid $\{m_i\}_{i=1}^N$ as follows:

$$m_i = \begin{cases} 1, & \text{if } \lambda_i > 0 \\ 0, & \text{if } \lambda_i < 0 \\ \text{undefined}, & \text{if } \lambda_i = 0 \end{cases} \quad (14)$$

Undefined grid cells are grid cells that were never observed by the robot or that were observed to be occupied an equal number of times as they were observed as free, thus the uncertainty in their state.

B. Texture Mapping

For texture mapping, we maintain a texture map $tm \in \mathbb{R}^{3\times \texttt{ncells}}$ containing the RGB color of the corresponding cells in the occupancy grid $\{m_i\}_{i=1}^{\texttt{ncells}}$. To achieve this, we begin by using the inverse of the matrix of intrinsic parameters of the depth sensor K_d^{-1} to transform the "pixel" locations of the depth image I_d to normalized coordinates in the optical frame of the depth sensor. We then "un-normalize" these coordinates by multiplying by their corresponding depths. Next, we use the extrinsic depth-to-RGB transformation matrix ${}_{rgb}K_d$ to transform the coordinates in the optical frame of the depth sensor to the optical frame of the RGB camera. These coordinates in the optical frame of the RGB camera are then used to achieve the two following tasks:

- Derive the pixel locations in the RGB image I_{rgb} corresponding to the optical frame coordinates.
- Derive the world frame coordinates corresponding to the optical frame coordinates.

To achieve the first task, we normalize the optical frame coordinates and then use the intrinsic parameter matrix of the RGB camera K_{rgb} to transform the normalized coordinates to RGB pixel locations. For the second task, we convert the optical frame coordinates to a regular frame using the following equation:

$$p_{regular} = \begin{bmatrix} 0 & 0 & 1 \\ -1 & 0 & 0 \\ 0 & -1 & 0 \end{bmatrix} * p_{optical} \quad (15)$$

With the regular coordinates of the points in the RGB camera frame, we use the robot dimensions, head and neck angles, and the camera pose to arrive at the coordinates of the camera pixels in the world frame. We then convert the world coordinates of the pixel locations to their corresponding occupancy grid cells using the map resolution. Now that we have RGB pixel locations and their corresponding grid cells on the map, we set the color of the grid cells to the RGB values at their corresponding pixel location.

IV. Result and Discussion

A. SLAM

For simultaneous localization and mapping, we use the following design parameters.

$$\begin{aligned} N &= 100 \text{ particles} \\ N_{thresh} &= 10 \text{ particles} \\ W_{3\times 3} &= 0.1 I_3 \\ g &= 8 \qquad \text{(see (11))} \end{aligned}$$

The particle filter worked pretty well, especially when compared with the occupancy grid map and robot pose derived from just pure odometry as seen in Fig. 2 below for one of the datasets. While the maps using the particle filter

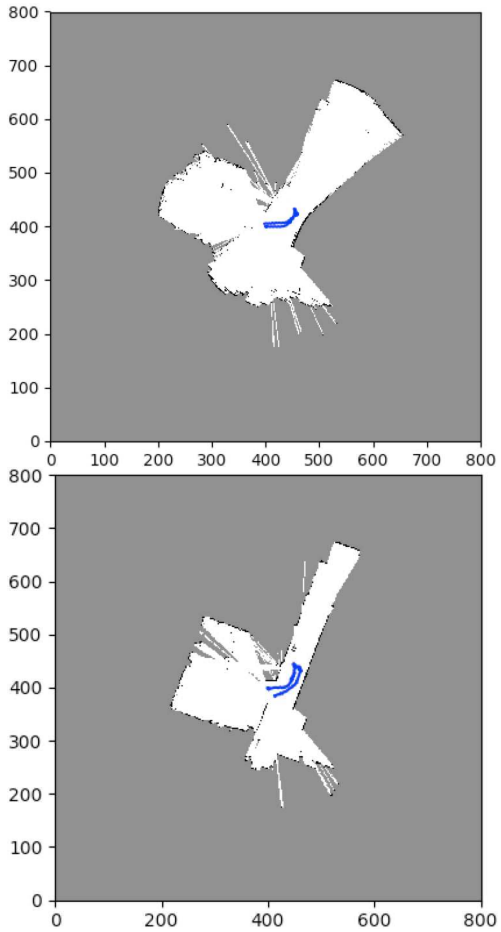

Fig. 2: Dataset `1`. Top: Odometry-only SLAM, Bottom: Particle Filter SLAM.

isn't exactly perfect with perfect straight edges, it does significantly improve the quality of the results as compared with the odometry-only map and pose. The results for the rest of the datasets are included in Figures 3–7.

B. Texture Mapping

The texture mapping algorithm performed impressively well albeit with some smudginess. This smudginess is probably due to the lack of consideration of camera distortion. When converting from the pixel locations in the depth image to the normalized coordinates in the optical frame of the depth sensor, the matrix K_d^{-1} actually converts the pixel locations to the the distorted coordinates. The relationship between the distorted and normalized coordinates are immensely nonlinear that the cost of designing a numerical algorithm to solve nonlinear equations involved outweighs the potential benefit of improving the texture mapping results. Figures 3–7 shows the result for texture mapping.

APPENDIX

Derivation of $T_{control}(t)$

Consider the odometry-provided pose of the lidar in the world frame ${}_wT_l(t)$, this pose can be expressed as:

$$ {}_wT_l(t) = {}_wT_b(t) * {}_bT_l(t) \tag{16} $$

$$ \implies {}_wT_b(t) = {}_wT_l(t) * {}_bT_l(t)^{-1} \tag{17} $$

Now consider the following odometry motion model of the body pose given as

$$ {}_wT_b(t+1) = {}_wT_b(t) * T_{control}(t) \tag{18} $$

$$ \implies T_{control}(t) = {}_wT_b(t)^{-1} * {}_wT_b(t+1) \tag{19} $$

Using the expression on the RHS of (17) in (19) gives:

$$ T_{control}(t) = \left[{}_wT_l(t) * {}_bT_l(t)^{-1}\right]^{-1} * \left[{}_wT_l(t+1) * {}_bT_l(t+1)^{-1}\right] \tag{20} $$

$$ = {}_bT_l(t) * {}_wT_l(t)^{-1} * {}_wT_l(t+1) * {}_bT_l(t+1)^{-1} \tag{21} $$

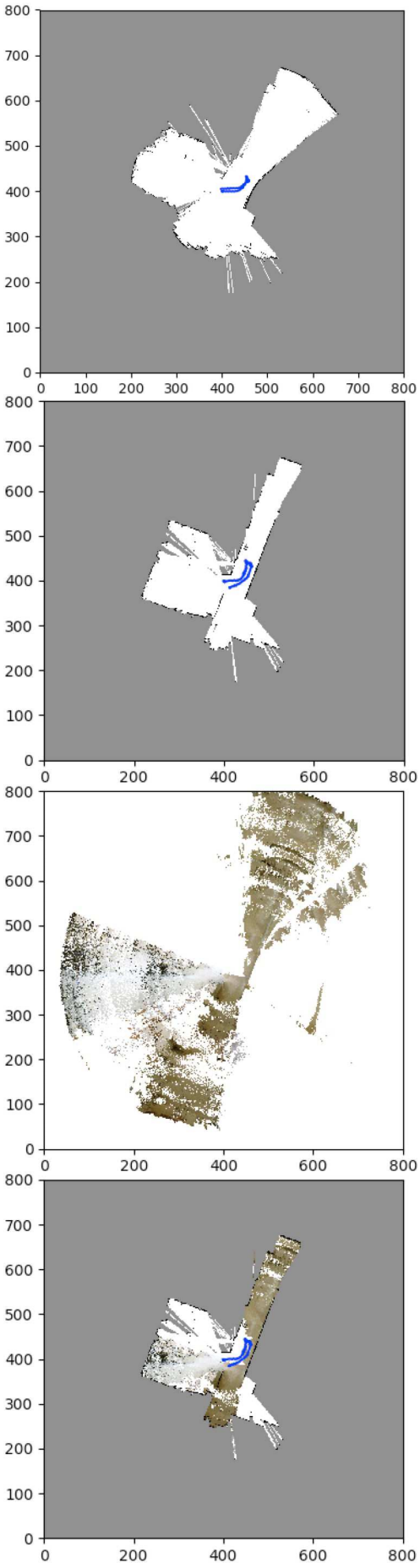

Fig. 3: Dataset `train0`. Top to bottom: Odometry-only SLAM, Particle Filter SLAM, Particle Filter Texture Map, SLAM and Texture.

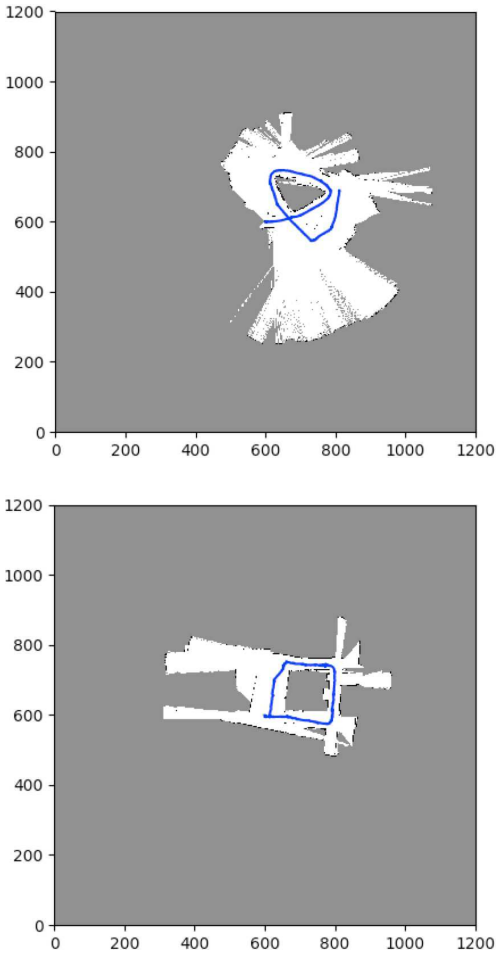

Fig. 4: Dataset `train1`. Top to bottom: Odometry-only SLAM, Particle Filter SLAM.

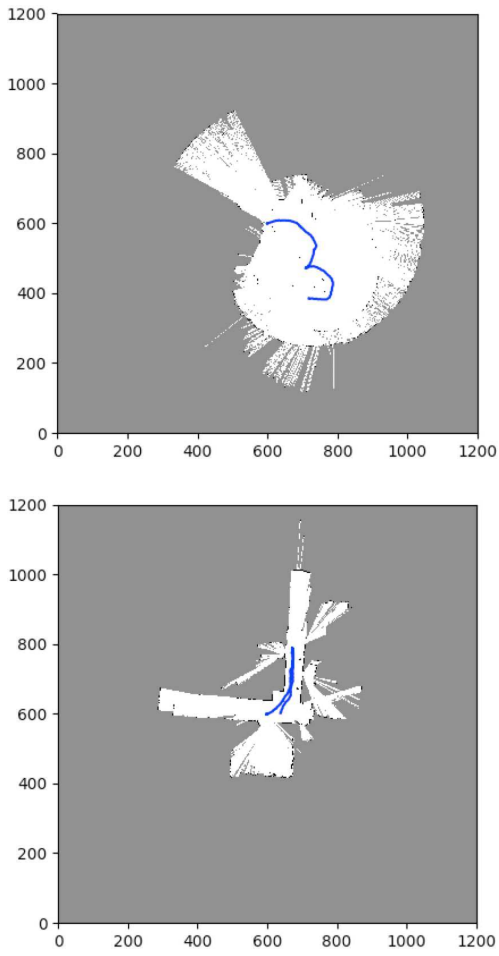

Fig. 5: Dataset `train2`. Top to bottom: Odometry-only SLAM, Particle Filter SLAM.

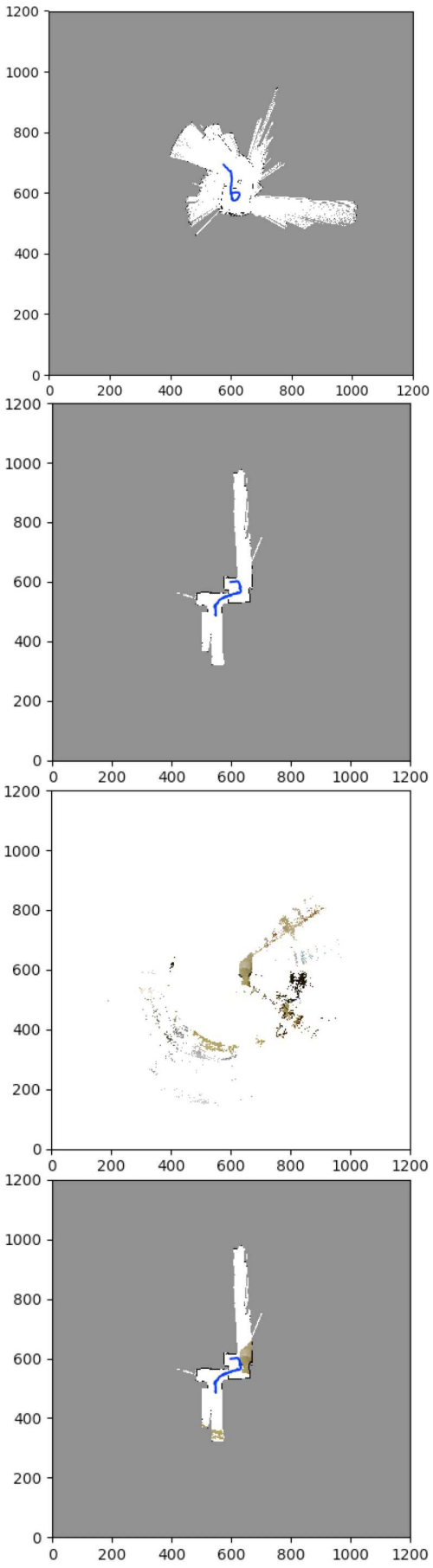

Fig. 6: Dataset `train3`. Top to bottom: Odometry-only SLAM, Particle Filter SLAM, Particle Filter Texture Map, SLAM and Texture.

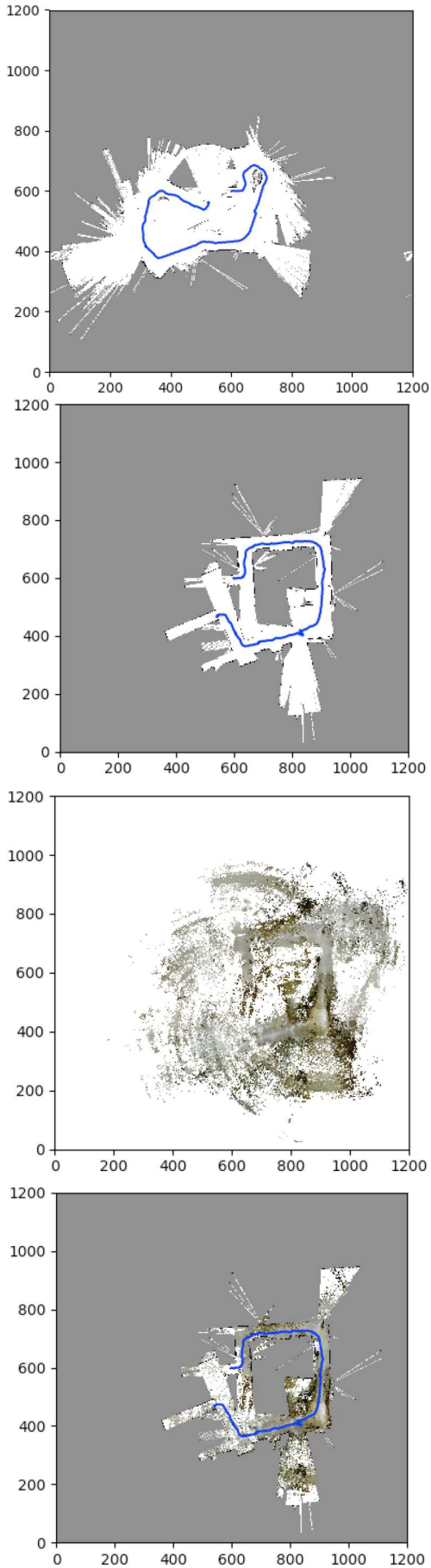

Fig. 7: Dataset `test`. Top to bottom: Odometry-only SLAM, Particle Filter SLAM, Particle Filter Texture Map, SLAM and Texture.