Simultaneous Localization and Mapping (SLAM)

Joni Lynne De Guzman
j5deguzm@eng.ucsd.edu
A53212113

1. Introduction

Autonomous navigation is a widely popular application in the world of robotics from self-driving vehicles to drones. For a robot to successfully perform autonomous navigation in its surrounding environment, it requires information about the world as well as its location in reference to the world. Simultaneous localization and mapping, or SLAM for short, addresses this application. Equipped with a variety of sensors such as LIDAR, cameras, and odometry, a robot can learn more about its environment.

2. Problem Formulation

Using sensor data from encoders, IMU, and lidar obtained from a differential drive robot, SLAM is performed to estimate the robot trajectory and build a 2D map of the environment. Additional RGB-D data from a camera is used to color the floor of the map. An occupancy grid is used for the mapping while a particle filter with a laser-grid correlation model is used for localization.

The problem can be decomposed into four parts:

1. **Mapping:** With a laser scan and the current robot position, main and update the 2D occupancy grid map.
2. **Localization Prediction:** Use a particle filter and a differential drive motion model to estimate the robot pose. With odometry and IMU yaw rate measurements, predict the next position of the particle.
3. **Localization Update:** Using a laser scan reading and the current pose of the particle, laser scan matching is used to correct the predicted position of the particle.
4. **Texture Map:** Using the RGB-D camera, the image points from the camera are projected onto the occupancy grid map to color the floor.

3. Technical Approach

Given a series of sensor observations over discrete timestamps, we update the robot's location x_t and simultaneously update a 2D occupancy grid map of the environment m_t. To make the occupancy map more interesting, we color it to match the floor of the environment. We use a particle filter with N particles to represent the possible locations of the robot in the environment and deploy our SLAM sequence to estimate the robot trajectory and build the map.

The sensors do not have the same sampling time, so we faced an issue of trying to align the measurements across the encoders, IMU, lidar, and camera data. We fix the encoders to be our reference time and find the closest IMU, lidar, and camera measurements to the timestamp of the encoders.

3.1. Mapping

We define N particles to symbolize N possible locations of where we think the robot might be. We choose $N = 10$ to speed up computation time. The best particle is chosen to represent the robot pose x_t (position and orientation):

$$x_t = \begin{bmatrix} x \\ y \\ \theta \end{bmatrix}$$

where x, y are the robot's coordinates in the world frame and θ is the yaw angle.

With each new lidar scan z_t, we update our 2D occupancy grid map. The occupancy map is represented with a log-odds map which accumulates values of confidence over time on whether a cell in the map m_i is occupied or free. The log-odds map reduces to the following:

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

where we update the log-odds by defining a measurement trust value on how much we trust the lidar data.

Our parameters for the map are chosen to be the following values, where `res` is the grid cell resolution, `size` is the size of the occupancy grid map, `occupied` is the log-odds value for cell occupancy, `free` is the log-odds value for a cell being free, and `map threshold` is the clipping threshold for the log-odds map.

```
MAP['res']           = 0.05 meters/cell
MAP['size']          = 70x70 cells
MAP['occupied']      = log(9)
MAP['free']          = log(1/9)
MAP['map threshold'] = 90
```

We detail the steps of updating the map as follows:

- Filter out noisy laser scan readings $z > 30$ meters and $z < 0.2$ meters as these readings are too far or too close. The updated laser scan is then converted from polar to homogeneous cartesian coordinates.
- The lidar readings are in the sensor's reference frame, so we must apply transformations to convert them from the sensor frame to the robot body frame and then to the world frame.

  $$\mathtt{L} \Rightarrow \mathtt{B} \Rightarrow \mathtt{W}$$

  The transformation from the sensor frame to the body frame consists of only a translation. No rotation is involved so the rotation matrix is identity. The transformation from the body frame to the world frame changes during each mapping step as it depends on the current robot pose. We provide the transformation matrices below.

  $$R_{bl} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} t_{bl} = \begin{bmatrix} 0.2983 \\ 0.005 \\ 0.494 \end{bmatrix}$$

  $$R_{wb} = \begin{bmatrix} \cos\theta & \sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} t_{wb} = \begin{bmatrix} x \\ y \\ 0 \end{bmatrix}$$

- To visualize the map, the lidar scan points are converted to grid cell coordinates. The lidar scan points correspond to locations where the sensor hit an object/wall. Since the lidar scan was able to reach those locations, we determine that the cells in between are free (i.e. unoccupied). We apply Bresenham's ray-tracing algorithm on the lidar scan to trace rays from the robot's position to the hit points of the lidar scans, which in turn gives us the unoccupied cells that the lidar passed through.
- The log-odds are decreased for the free cells and increased for the occupied cells. All unobserved cells retain their previous value. We experimented with different odds ratios (i.e. our measurement trust) for cell occupancy and settled on a 90%-10% odds. The odds of any given cell m_i being occupied is computed as:

  $$g(1) = \frac{P(z_t = 1 | m_i = 1)}{P(z_t = 1 | m_i = 0)}$$

  The odds of a cell being free is the inverse of it being occupied: $g(0) = \dfrac{1}{g(1)}$.
- Constrain the log-odds values to $-90 \leq \lambda_{i,t} \leq 90$ to prevent overconfident estimation

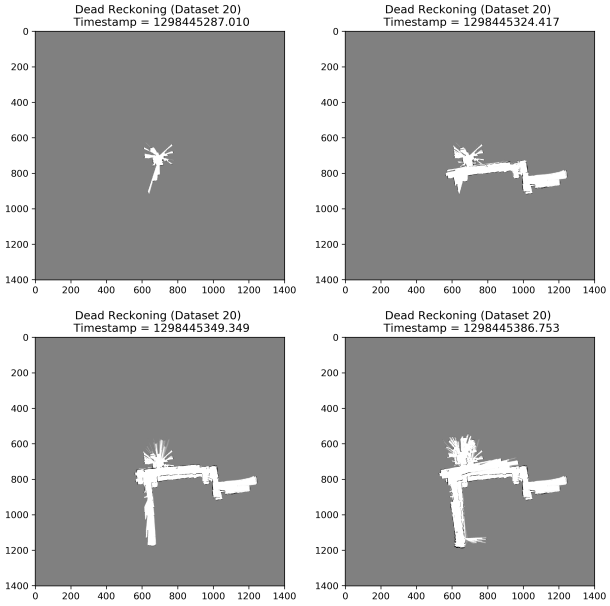

Figure 1. Evolution of dead reckoning map over time on dataset 20

3.2. Localization Prediction

A particle filter is used to localize the robot in the environment. The particle filter approximates the distribution of the robot state s_t at a certain time given all the observations up to that time. The state for our purposes is the pose μ_t^i for each of our 10 particles. As the robot moves throughout the environment, the particles need to be propagated through the model that describes the motion of the robot.

We fix the parameters of our particles as such:

```
PARTICLES['nums']    = 10
PARTICLES['weights'] = 1 / PARTICLES['nums']
PARTICLES['poses']   = zeros(3,PARTICLES['nums'])
PARTICLES['n_eff']   = 0.1 * PARTICLES['nums']
```

where the particle states i.e. `poses` are initialized to be $(0, 0, 0)$ to coincide with the world frame, `weights` are probabilities of each particle localizing on the robot, and `n eff` will be explained in the following section.

With each new encoder and IMU reading, we can compute the updated particle states using a discrete-time differential drive motion model. The motion model is of the form:

$$s_{t+1} = s_t + \tau \begin{bmatrix} v_t \mathtt{sinc}(\frac{\omega_t \tau}{2}) \cos(\theta_t + \frac{\omega_t \tau}{2}) \\ v_t \mathtt{sinc}(\frac{\omega_t \tau}{2}) \sin(\theta_t + \frac{\omega_t \tau}{2}) \\ \omega_t \end{bmatrix} \quad (2)$$

where $s_t = [x, y, \theta]^T$ is the previous state value.

Between every encoder reading τ, the robot's displacement is calculated by averaging the encoder counts for both the left and right wheels and multiplying the averaged

counts by 0.0022. The left wheel and right wheel displacements are then averaged to get τv_t, its displacement since the last encoder reading. The yaw rate from the IMU is multiplied with the time between consecutive encoder measurements to obtain the robot's angular velocity ω_t. Once these are parameters are calculated, we plug them into 2 to predict the new pose $\mu^i_{t+1|t}$ of each particle.

Gaussian noise $N(\mu, \sigma^2)$ with mean 0 and variance 1 is added to the updated states to make our model more robust. The noise added to θ is an order smaller than the noise added to x, y.

3.3. Localization Update

The observation model in the update step is the laser correlation model. The model uses the laser scan reading z_t applied on each particle pose μ^i_t to calculate the correlation scores between the particle's laser scan and the occupancy grid map 3. The laser scans are transformed to the world frame with the pose in order to find the matching hit points. A high correlation score means that the laser scans hit points matches very well with that of the occupancy grid map.

$$corr(z, m) = \Sigma \mathbf{1}\{m_i = z_i\} \tag{3}$$

Note that we do not use the log-odds map for the correlation matching. Instead, we create a probability map by shoving our log-odds into equation 4 to a cell's probability of occupancy.

$$\gamma_{i,t} = 1 - \frac{1}{1 + \exp(\lambda_{i,t})} \tag{4}$$

Since we have 10 particles, we calculate 10 correlation scores. We used the provided `mapCorrelation` to calculate the scores. We add variation in x and y of the particle's pose when doing the map correlation and take the max of the variations to represent the correlation score for that particle. The correlation scores are fed into a softmax function to obtain probability values for all particles. These probability values are multiplied with the current particle weights to obtain the updated weights $\alpha^i_{t+1|t+1}$.

Particle depletion is a situation where in most of the updated particle weights are nearly zero. To avoid this, we apply stratified resampling to create a new particle set still of size N but with uniform weights. Stratified resampling redistributes the particles by adding more particles to the locations with high weights and removes particles from locations with lower weights. Resampling is applied if the number of effective particles is below our `n eff` threshold of 1. The number of effective particles is calculated with:

$$N_{eff} = \frac{1}{\Sigma_i (\alpha^i_{t|t})^2}$$

At the end of this step, the best particle is chosen as the one with the highest weight. In other words, it is the one that has the best correlation with the occupancy map.

3.4. Dead Reckoning

Before running the full SLAM sequence, we first run dead reckoning on the two training datasets to test our mapping and prediction functions. Dead reckoning uses prediction only steps on a single particle without any additive noise to estimate the robot trajectory and construct the 2D map. Figure 1 shows how the occupancy map evolves over time for dataset 20.

As a second test, we run noisy dead reckoning to see how the addition of noise can corrupt the trajectory. Figure 2 shows the robot trajectory with dead reckoning and with noisy dead reckoning. The trajectories look very similar. The addition of noise did not corrupt the results as much as anticipated. Figure 3 shows how the map changes over time with noisy dead reckoning.

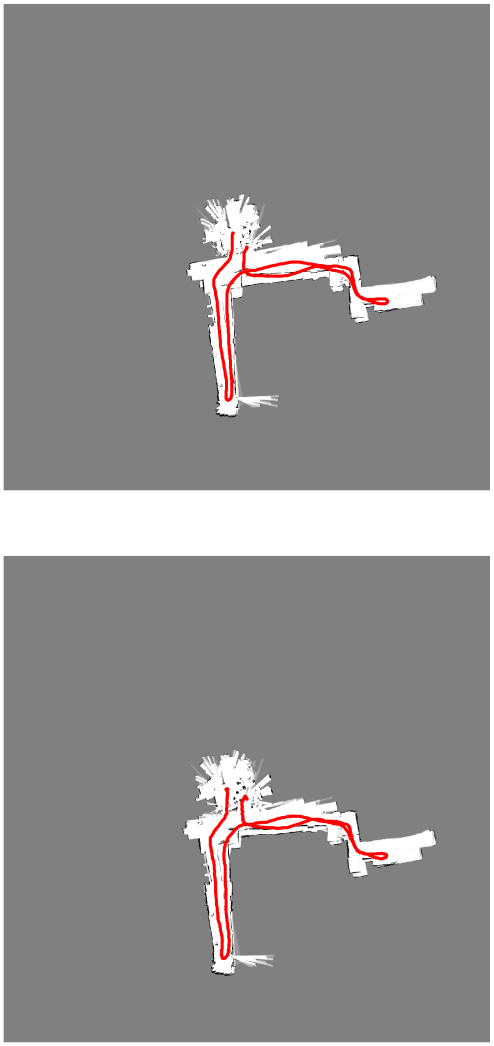

Figure 2. Dead reckoning (top) vs noisy dead reckoning (bottom) trajectory on dataset 20

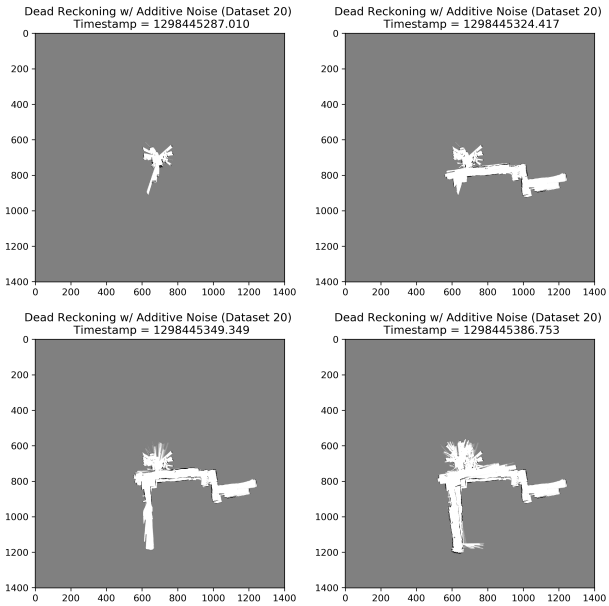

Figure 3. Evolution of noisy dead reckoning map over time on dataset 20)

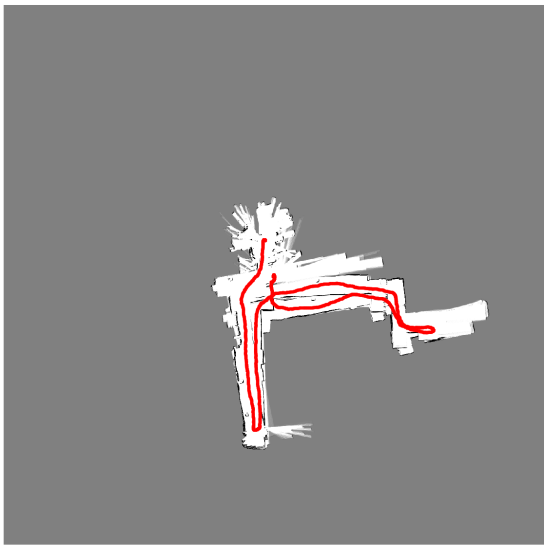

Figure 4. SLAM on dataset 20

3.5. SLAM

The full SLAM sequence now includes the localization update step with particle resampling. In addition, we use our N particles instead of just a single particle. Using the laser correlation model, we can correct our prediction and update the weights of the particles.

We shows the results on dataset 20 where we experiment with different values of noise. The standard Gaussian normal noise is used as detailed in 3.2 and the resulting trajectory is shown in Figure 4.

We lower the noise by removing the additive noise from the (x, y) components of the robot pose. Naturally, with less noise, the results are similar to that of dead reckoning (shown in Figure 5). Additionally, we show some images of how the motion of the particles propagate over time in Figure 6.

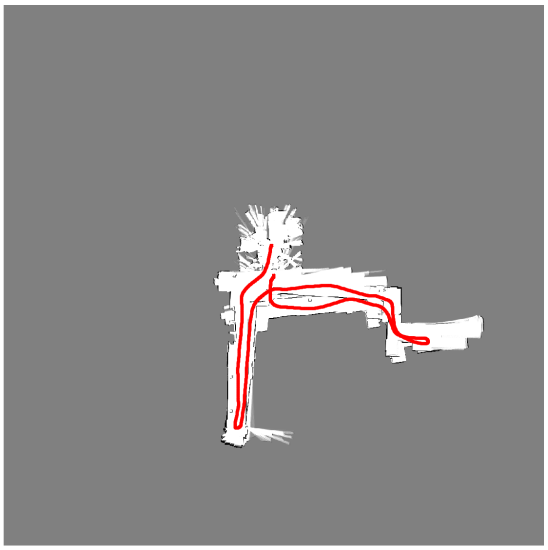

Figure 5. SLAM with less noise on dataset 20

3.6. Texture Map

An RGB-D camera is used to produce a colored version of the 2D occupancy grid map. Since the image points associated with the floor are in image coordinates, we must first backproject the 2D points (x, y) to its corresponding 3D point (X_o, Y_o, Z_o) in the optical frame. Then, we convert the points to the robot body frame and finally to the world frame.

$$\mathtt{I} \Rightarrow \mathtt{O} \Rightarrow \mathtt{B} \Rightarrow \mathtt{W}$$

These are the steps to apply texture mapping to the occupancy grid:

- There is an x-axis offset between the depth camera and the RGB camera so we apply a transformation mapping to map points between the two images. Once the depth points have been matched to the RGB points, this gives us our (x, y) image coordinates from which we extract the RGB pixel values.

- Backproject the image coordinates to the optical frame using the camera calibration matrix K and the depth

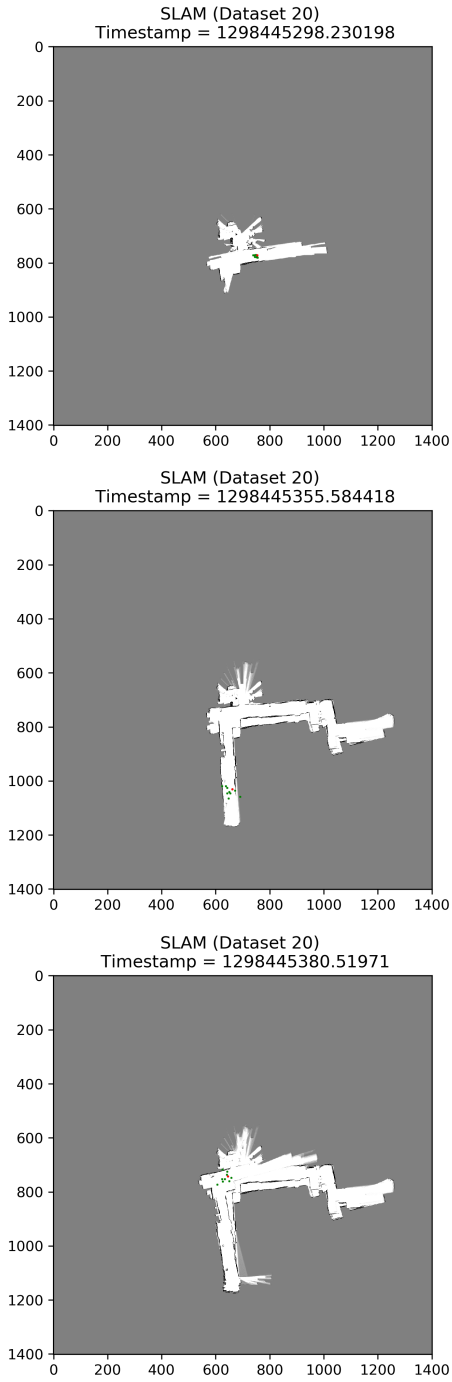

Figure 6. Propagation of motion of particles over time (dataset 20)

values Z_o with the following transformation:

$$\begin{bmatrix} X_o \\ Y_o \\ Z_o \\ 1 \end{bmatrix} = K^{-1} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} Z_o$$

$$K = \begin{bmatrix} 585.05108211 & 0 & 242.94140713 \\ 0 & 585.05108211 & 315.83800193 \\ 0 & 0 & 1 \end{bmatrix}$$

- Apply transformations to convert the 3D points to the world frame to find the floor points. We know the camera has the following orientation: $\phi = 0$ rad, $\theta = 0.36$ rad, and $\psi = 0.21$ rad. The camera center is also (0.18, 0.005, 0.36) meters from the robot center. This gives the following rotation and translation:

$$R = R_z(\psi)R_y(\theta)R_x(\phi)$$
$$t = [0.18, 0.005, 0.36]^T$$

- Threshold the height of the depth points to obtain the points associated with the floor, and color the occupancy map with the corresponding RGB colors of the floor image points. We found the optimal thresholds for height to be $-0.05 < z < 0.85$.

4. Results

4.1. SLAM

Our SLAM implementation performs decently well on the training and test data. It performs better on dataset 20 (see Fig 4) when compared to dataset 21. We show the results on dataset 21 and the testset in Figure 8. A sample of figures in Fig 9 for dataset 21 shows the motion of the particles throughout the map. It is similarly done for the testset in Fig 10.

To view the full SLAM implementation on dataset 20, 21, and 23, we provide links below that contain videos showing the how the occupancy grid map develops over time as well as how the robot moves in the map:

- dataset 20, dataset 21, dataset 23

To improve our results, we can either increase the number of particles or change our localization update step to add yaw variation around the particle's position. Of these two suggestions, it would be easier to increase the particle number.

4.2. Texture Map

We had surprising good results on our texture maps and found that what really helped was choosing appropriate threshold values to keep mostly floor points. Figure 7 displays our texture maps.

4.3. Observations

Some key observations we found during of implementation:

1. Less noise applied to the motion model resulted in an increased amount of resampling.

2. More noise applied to the yaw value produced a degraded robot trajectory and a poorer quality map.

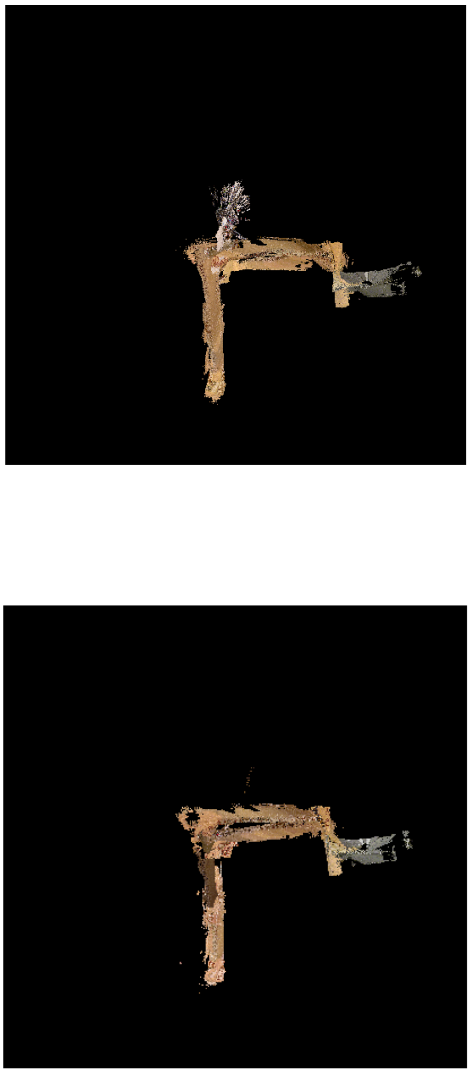

Figure 7. Texture maps of Dataset 20 (top) and Dataset 21 (bottom)

3. Tuning the threshold values for the log-odds map and the probability map resulted in more distinct walls in the occupancy map.

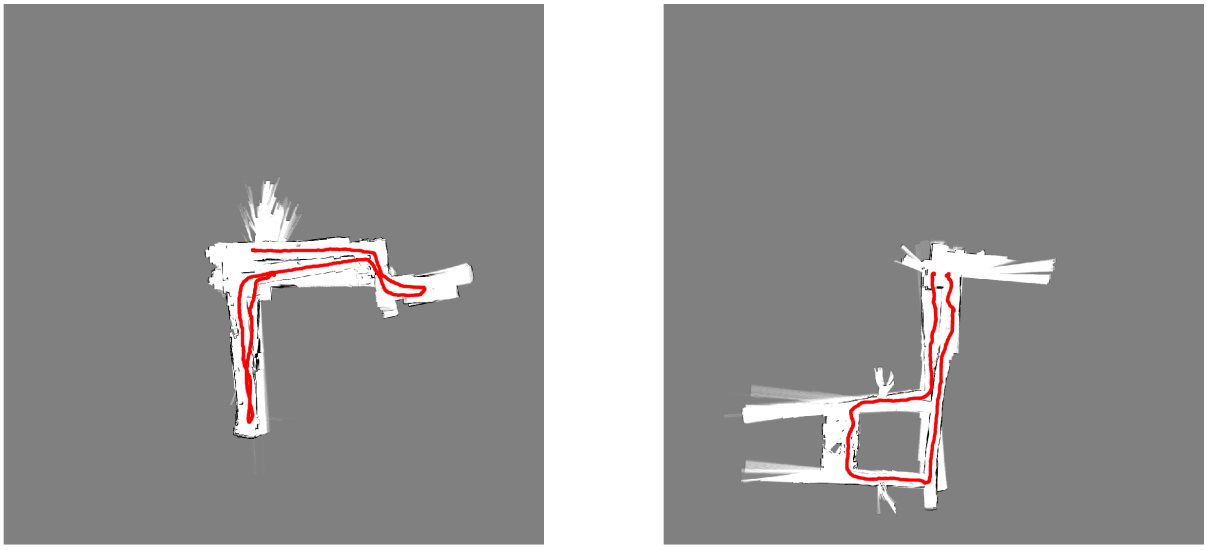

Figure 8. SLAM on (a) Dataset 21, (b) Testset

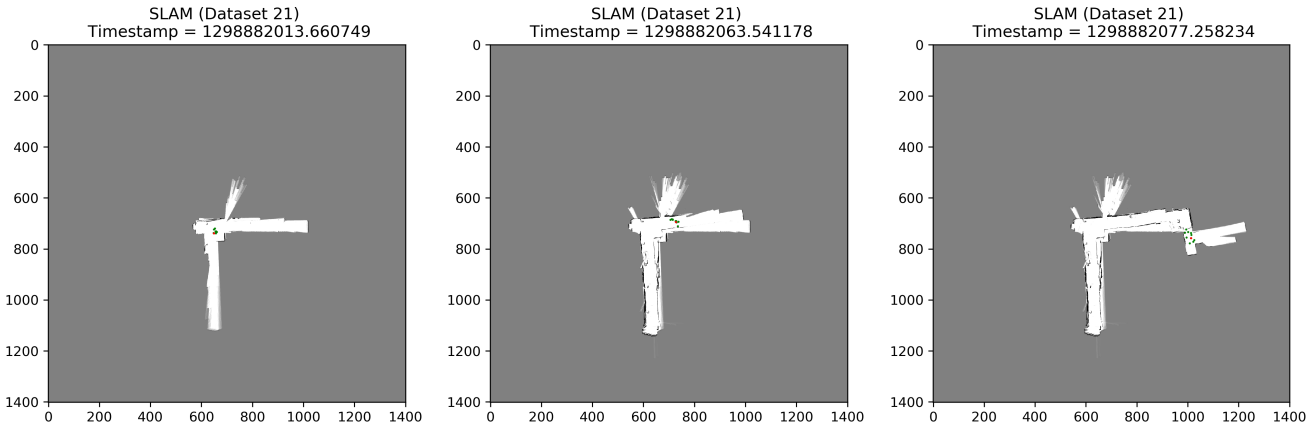

Figure 9. Movement of particles for Dataset 21

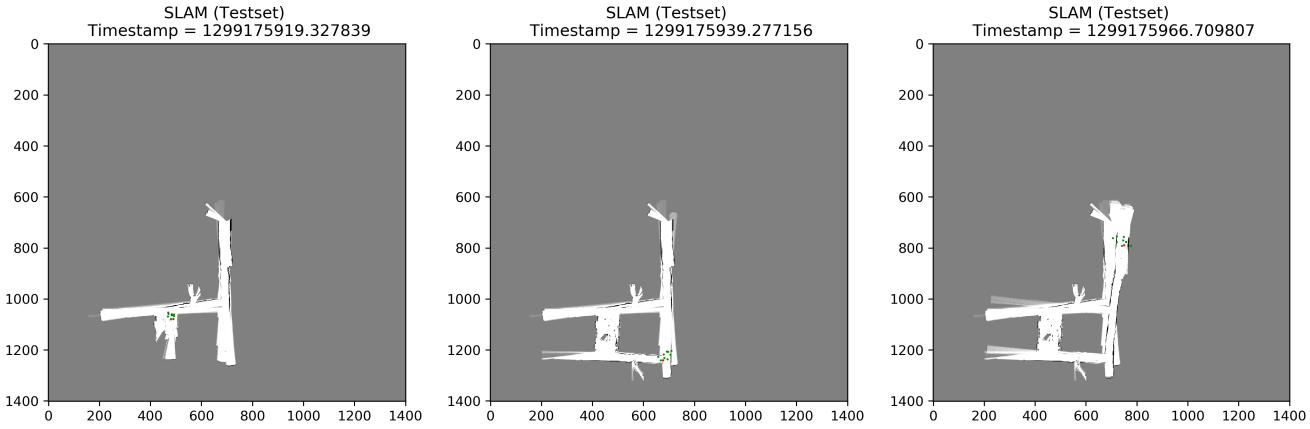

Figure 10. Movement of particles for Testset