# Simultaneous Localization and Mapping with Particle Filters

Michael Liu
University of California San Diego
Electrical Computer & Engineering Department
x4liu@eng.ucsd.edu

## Abstract

*This report discussed the approach of using particle filter to perform simultaneous localization and mapping on a humanoid robot (THOR). In this work, we implemented the particle filters and performed the mapping on several given lidar and joint dataset.*

## 1. Introduction

As the self-navigating vehicles become a promising field, simultaneous localization and mapping has become one of the most important challenges for robot to have robust estimation to its surroundings. However, the data recorded from the robot often present tremendous noise and to effectively eliminate the sensor noise, people have proposed several filter techniques. In this paper, we will focus on one particular filter: the particle filter, and test the robustness of this filter.

## 2. Problem Formulation

### 2.1. Dataset Overview

In this task, we use data collected by THOR (**Figure** 1), a humanoid robot equipped with a Lidar sensor, joint sensors (head and neck motor), and a Kinect-camera sensor (RGB-D camera). In this report, we discuss the result of these 5 of the total dataset collected by THOR on solving the simultaneous localization and mapping problem, and an overview of these dataset by the number of timestamps collected in the dataset is provided in **Table** 1.

Because there are more frames of joint sensor data than the lidar data (motor sensor is collecting roughly 3x faster than the lidar sensor), for each dataset, we synchronized the data frame based on the lidar timestamps. The synchronized dataset contains joint sensor data $\begin{bmatrix} \theta_{\text{neck}} \\ \theta_{\text{head}} \\ t \end{bmatrix}$ (in radian), lidar range data $\begin{bmatrix} \text{range} \\ \theta \\ t \end{bmatrix}$ (range in meters), and the change of robot pose from last reading of lidar sensor $\begin{bmatrix} \delta_x \\ \delta_y \\ \delta_\theta \end{bmatrix}$ (x, y in meters, $\theta$ in radian). The lidar sensor collects 1081 rays per timestamps with a range of (-130, 130.25) and interval of 0.25 (radian). Figure 2 has shown one scan of lidar sensor in the converted XY coordinate and figure 3 has shown one example of overhead Kinect-camera sensor collected in dataset 0.

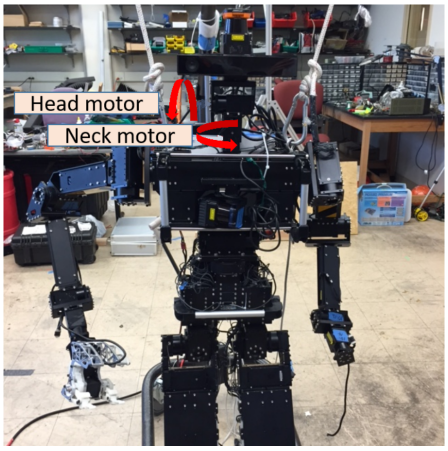


Figure 1. The data was collected by the THOR robot. The head is above the center of mass (body frame) and can rotate in two directions. The head motor rotates the head up and down, and the neck motor rotates the head left and right

| THOR | lidar | joint | RGB | depth |
|---|---|---|---|---|
| dataset 0 | 12048 | 38318 | 228 | 228 |
| dataset 1 | 34209 | 101563 | 0 | 0 |
| dataset 2 | 12679 | 38081 | 0 | 0 |
| dataset 3 | 12386 | 35701 | 996 | 996 |
| dataset 4 | 28224 | 83888 | 2440 | 2440 |

Table 1. THOR Dataset Overview (Number of Frames Collected)

Figure 2. Example of one scan from the lidar sensor in the XY coordinate

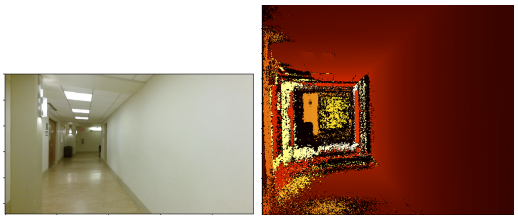

Figure 3. Example of RGB and depth images collected by the Kinect camera

### 2.2. Simultaneous Localization and Mapping Formulation

Given the dataset above, we formulate the problem as three parts for solving the Simultaneous Localization and Mapping problem.

1. **Robot Pose Localization (Motion Model)**: Given the change of robot position $\begin{bmatrix}\delta_x \\ \delta_y \\ \delta_\theta\end{bmatrix}$ and the previous position of the robot position, incorporate sensor noise to predict the next position of the robot. Here our motion model formulation is:

$$\begin{bmatrix}x \\ y \\ \theta\end{bmatrix}^{t+1} = \begin{bmatrix}x \\ y \\ \theta\end{bmatrix}^{t} + \begin{bmatrix}\delta_x \\ \delta_y \\ \delta_\theta\end{bmatrix}^{t} + \mathcal{N}(\mu,\, \sigma^2)$$

   where $\mathcal{N}(\mu,\, \sigma^2)$ is our assumption of the sensor noise, which is a Gaussian distribution around $\mu$.

2. **2D Lidar Mapping (Lidar/Joint Observation Model)**:

   Given the robot current position $\begin{bmatrix}x \\ y \\ \theta\end{bmatrix}^{t}$, use its observation to maintain and update the existing 2D map with lidar and joint motor sensoring data. Here we want to perform a series of transformation to transfer the observation dataset from Lidar frame to the World frame. The series of update is as followed:

$$\begin{bmatrix}r \\ \theta\end{bmatrix}_{\text{lidar}} \rightarrow \begin{bmatrix}X \\ Y \\ Z\end{bmatrix}_{\text{head}} \rightarrow \begin{bmatrix}X \\ Y \\ Z\end{bmatrix}_{\text{body}} \rightarrow \begin{bmatrix}X \\ Y \\ Z\end{bmatrix}_{\text{world}} \rightarrow \begin{bmatrix}u \\ v\end{bmatrix}_{\text{map}}$$

   Once we have calculated the u, v coordinates in the map frame, we updated the probability of occupied and free cell in map.

3. **2D Lidar Mapping Update**:

   Given the current position of the robot $\begin{bmatrix}x \\ y \\ \theta\end{bmatrix}^{t}$, update the logits probability of the occupied and free cells observed in the map, using the joint motor sensor on the head and neck angles and lidar data. The definition of logits for occupied cell in map is:

$$g(z=1) = \frac{P(z_t = 1|o_t = 1)}{P(z_t = 1|o_t = 0)} = \frac{P(\text{happens})}{P(\text{not happens})}$$

   And for the free cell in map:

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

   To simplify, the logits probability of occupied cell is the ratio between the conditional probability of current map agrees and disagrees with current observation. Logits is used to update the map probability allows value overflow by having a smaller range of number and it also avoids numerical stability by performing simple add operation at every map update.

4. **Texture Mapping (Camera Observation Model)**:

   In dataset 0, 3, 4, we performed the texture overlay on the existing map with the given RGB image and depth estimation provided by the Kinect camera. We used a camera observation model to perform back-projection using the depth information on the image, and we were able to convert the RGB pixel color points through the series of transformations to the map frame. Overlay the texture onto our existing map.

## 3. Technical Approach

### 3.1. Motion model of THOR Localization

For the motion model of our robot pose estimation, we used the noise standard deviation of 0.01 on all three parameters ( $\begin{bmatrix}\delta_x \\ \delta_y \\ \delta_\theta\end{bmatrix}$ ), with a 0 mean Gaussian distribution. Therefore, during our prediction step, we simply takes the change

of robot pose in the world frame and apply it with some sensor noise for spreading the particle hypothesis on the robot poses, which is described in the later section.

$$\begin{bmatrix} x \\ y \\ \theta \end{bmatrix}^{t+1} = \begin{bmatrix} x \\ y \\ \theta \end{bmatrix}^{t} + \begin{bmatrix} \delta_x \\ \delta_y \\ \delta_\theta \end{bmatrix}^{t} + \mathcal{N}(\begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 0.01 \\ 0.01 \\ 0.01 \end{bmatrix})$$

### 3.2. 2D Lidar Mapping (Lidar/Joint Observation Model)

From the THOR configuration, we can see that the maximum range of the lidar sensor is around 20 meters. The center of the robot mass is 0.93 meters above the ground, head is 0.33 meters above the body center, lidar is 0.15 meters above the head center, and Kinect camera is 0.07 meters above the head center. Given the yaw (neck angle from the joint sensor) and pitch (head angle from the joint sensor) in the head frame, and robot current position in the world frame, we can then performs a series of transformation of converting the lidar observation points onto the world frame. Our lidar to head frame transformation is:

$${}_lT_h = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0.15 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Given the $y$ (yaw) and $p$ (pitch) of the head of the robot, we can construct the transformation matrix from the head frame to body frame ${}_hT_b$.

$$R_y = \begin{bmatrix} \cos(y) & -\sin(y) & 0 \\ \sin(y) & \cos(y) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$R_p = \begin{bmatrix} \cos(y) & 0 & \sin(p) \\ 0 & 1 & 0 \\ -\sin(p) & 0 & \cos(p) \end{bmatrix}$$

$$t_h = \begin{bmatrix} 0 \\ 0 \\ 0.33 \end{bmatrix}$$

$${}_hT_b = \begin{bmatrix} R_yR_p & t_h \\ 0 & 1 \end{bmatrix}$$

Given the current robot position $\begin{bmatrix} x \\ y \\ \theta \end{bmatrix}$, since we use a 2D robot position, we only consider the rotation around the z-axis for the robot. Therefore, the transformation of our robot from body frame to the world frame is:

$${}_bT_w = \begin{bmatrix} \cos(\theta) & -\sin(\theta) & 0 & x \\ \sin(\theta) & \cos(\theta) & 0 & y \\ 0 & 0 & 1 & 0.93 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

To convert the point coordinates from the lidar to world, we performed the series of operations using the above transformation:

1. We first clipped all the lidar rays to the range value of (0.3, 15), this allows us to get rid of the noise in the lidar sensor by removing close and far points.
2. We converted the lidar points from Cartesian $\begin{bmatrix} \text{ranges} \\ \theta \end{bmatrix}$ to XYZ coordinates as $\begin{bmatrix} \text{ranges}\cos(\theta) \\ \text{ranges}\sin(\theta) \\ 0 \end{bmatrix}$.
3. We use the homogenized XYZ coordinates of the points in the lidar frame and transformed it to the world frame using the above transformation:

$$\begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}_{\text{world}} = {}_bT_{wh}T_{bl}T_h \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}_{\text{lidar}}$$

4. Once we get the homogenized points in the world coordinates, we can simply extract the X, Y coordinates and convert to the u, v coordinate system based on the map resolution:

$$\begin{bmatrix} u \\ v \end{bmatrix}_{\text{map}} \leftarrow \begin{bmatrix} X \\ Y \end{bmatrix}_{\text{world}} / \text{map resolution} + \text{map center}$$

### 3.3. 2D Lidar Map Update

With each new lidar scan, we update our 2D occupancy grid map based on the $\begin{bmatrix} u \\ v \end{bmatrix}_{\text{map}}$ map coordinates of observation $z_t$. The 2D occupancy map is represented with an accumulated log-odds map of confidence of grid observation over time. The log-odds map calculation can be further reduces to:

$$\lambda_i^{t+1} = \lambda_i^t + \log g_h(z_{t+1}|m)$$

This operation is performed on both free and occupied cells during updates. And to avoid our occupancy map from being over-confidence, we threshold the map value to be in the range of (-200, 200).

With the simple motion model and map update steps, we are able to perform dead reckoning of our initial robot SLAM system, and the result is shown in Figure 4.

### 3.4. Particle Filter for Robust Estimation

From our experiment with just the dead reckoning of the 2D SLAM process in Figure 4, we found that the sensor noise is very large that sometimes throw off the accuracy of

| | |
|---|---|
| Map Resolution | 0.05 |
| Map Total Size | 80 |
| Occupancy Log-odds | $\log(4)$ |
| Free Log-odds | $-\log(4)$ |
| Landmark Threshold | 0 |

Table 2. Occupancy Map Configuration (in meters)

the pose position. In order to perform a more robust estimation of the robot, we implemented the particle filter to better estimate the robot pose and accurately update the 2D map.

The process of implementing the particle filter is as follow:

1. Initialize a large number of $N$ particles (starting robot pose) at $\begin{bmatrix}0\\0\\0\end{bmatrix}$ with a uniform weight distribution of $\frac{1}{N}$.

2. For every timestamp, apply each individual particles (unique hypothesis of the robot positions) with the motion models using a sensor noise of $\sigma = 0.01$.

3. After performing 5 prediction steps, we evaluate the quality of the particle pose against our existing map (use only the occupied cell as landmarks) through a fast map correlation. This correlation score is essentially the total count of occupied cells in existing map that has the same position as the current observed points by particle $i$ in the world frame.

$$\text{corr}(\mu_i, z) = \sum \mathbf{1}\{o_\mu = z\}$$

We take the correlation scores and convert it to the Softmax probability to update the weight associated to each of the particle. To avoid the numerical instability of the exponent factor, we first subtract the correlation scores with the maximum scores to ensure all scores are negative.

$$\text{corr}_i = \text{corr}_i - \max(\text{corr})$$

we then use the correlation score to update the weights of each of our particles: ($\mu$ is the particle pose in the world frame)

$$P(z_{t+1}|\mu_k) = \frac{\exp(\text{corr})}{\sum_i \exp(\text{corr}_i)}$$

$$\alpha_{t+1} = \alpha_t * P(z_{t+1}|\mu_k)$$

To ensure after the weight update, the particle weights remains to sum up to 1, we further normalize the weight by the total weights to get the updated particle weight.

$$\alpha_{t+1} = \frac{\alpha_{t+1}}{\sum_i \alpha_i}$$

4. With each weight update, we updated the map log-odds with the current largest weight particle pose as our robot pose estimation and perform the transformation to the world frame from the lidar frame using such pose.

5. We used stratified resampling strategy to break up large weight particles to keep particles from being ineffective for getting small weights. The threshold $N_{\text{thres}}$ we used is .3 of the total number of particles created to estimate the robot pose, and we only perform the resampling on the total particles when:

$$N_{\text{eff}} = \frac{1}{\sum_{k=1}^{N}(\alpha_k)^2} < N_{\text{thres}} = .3 * N$$

.

### 3.5. Texture Map from RGB-D Kinect Camera

Once we performed the Particle SLAM and build our 2D occupancy map, we proceeded to overlay the map with texture through data observed in RGB-D Kinect Camera. Given the Kinect configuration, we can first construct the transformation from the Kinect camera to head frame.

$${}_kT_h = \begin{bmatrix}1 & 0 & 0 & 0\\0 & 1 & 0 & 0\\0 & 0 & 1 & 0.07\\0 & 0 & 0 & 1\end{bmatrix}$$

The rotation and translation from the depth camera to RGB camera is:

$${}_dR_{rgb} = \begin{bmatrix}0.9999 & 0.0005 & 0.0005\\-0.0005 & 0.9999 & -0.0001\\-0.0005 & 0.0001 & 0.9999\end{bmatrix}, {}_dt_{rgb} = \begin{bmatrix}0.0523\\0.0001\\0.00006\end{bmatrix}$$

The calibration matrix of the depth camera and RGB camera is:

$$K_d = \begin{bmatrix}364.4573 & 0 & 258.42248\\0 & 364.4573 & 202.48713\\0 & 0 & 1\end{bmatrix}$$

$$K_{rgb} = \begin{bmatrix}525 & 0 & 524.5\\0 & 525.5 & 267\\0 & 0 & 1\end{bmatrix}$$

To match the depth information with the xy-coordinate information in the camera space, we want to first perform a transformation to match the depth information with the pixels in the RGB image space. The homographic transformation from depth camera to image camera is as follow:

$$K_{rgb}({}_dR_{rgb}K_d^{-1}\begin{bmatrix}u\\v\\1\end{bmatrix}_d + {}_d\,t_{rgb}) = \begin{bmatrix}u/w\\v/w\\1\end{bmatrix}_{rgb}$$

Perform the backprojection with values $Z_o$ obtained from the depth image, we can then get the 3D point cloud in the RGB image optical frame.

$$\begin{bmatrix} X_o \\ Y_o \\ Z_o \\ 1 \end{bmatrix} = K_{rgb}^{-1} \begin{bmatrix} u \\ v \\ 1 \end{bmatrix}_{rgb} Z_o$$

Transform the point cloud from the optical frame to the regular frame in the kinect frame.

$$\begin{bmatrix} X_r \\ Y_r \\ Z_r \\ 1 \end{bmatrix} = \begin{bmatrix} 0 & -1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X_o \\ Y_o \\ Z_o \\ 1 \end{bmatrix}$$

And then we can apply the series of transformation to get corresponding 3D XYZ coordinates in the world frame. From there, we only need to find the points with $Z_{\text{world}}$ that is close 0 (the ground level in the world frame). We plotted to pixel colors in the image with the $X_{\text{world}}$ and $Y_{\text{world}}$ converted map coordinates in our map frame. The segmentation results frame depth camera perspective cane be seen in Figure 10.

## 4. Results

In Figure 4, we have demonstrated the effectiveness of particle filters applied to the sensing and estimation of the robot localization and mapping. We have also applied particle filters in **dataset 0** (Figure 5), **dataset 1**, **dataset 2** (Figure6), **dataset 3** (Figure 7) and **dataset 4** (Figure 8). To improve the results, we have also tried out various number of particles with different sensor noise ranges from 100 to 5000 particles. The system shows promising results with noise of 0.01 and 2000 particles and a downsample rate (predict:update ratio) of 5. We have also tried out various particles motion noise and the results show significant efficiency improvement when we increase the noise on the yaw angle and decrease the noise on the x, y axis of the robot. This result is reasonable since the robot's movement is considerably slow but created significant angular noise through its local-motion.

## 5. Discussion

In this section, we discussed the importance of three aspects of the particle filters which controls the stability of the systems: the noise standard deviation applied in the motion model, the number of particles used in the SLAM system and the down-sampling ratio between the predict step and the update step of the particles.

Before we investigate each of the elements, we want to first state an important insight on how the particle filters work. What the particle is essentially doing is to generate

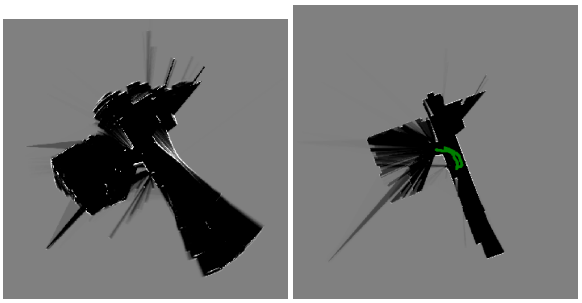

Figure 4. map estimation with and without particle filter for dataset 0. This has demonstrated the significant improvement on eliminating the noise and perform map reconstruction with only 100 particles.

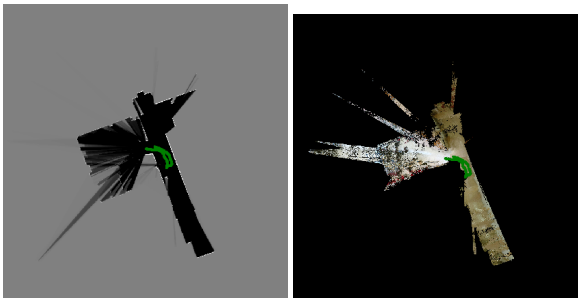

Figure 5. Lidar and texture mapping for dataset 0 (100 particles)

good quality samples of hypotheses in the solution space. This means the stability of the particle filter SLAM depends on two factors: the quality of each particles in terms its sampling efficiency and the range of potential solution space it is able to cover. Based on these two important factors, we will discuss the three hyper-parameters used in our systems, and each role of controlling the behavior of our overall result of the SLAM system.

### 5.1. Gaussian Noise in Motion Model

Applying a Gaussian noise is essential in the particle filter systems since this is the only factor that increases the variations in the particle positions and separate the particles out in the potential solution space. A small Gaussian noise assumes that the sensor drifts in a small ranges, which means that the potential solution space is small. However, the system becomes unstable when encounters large sensor drifts, because during the drift the noise applies to the particles is incapable of forcing it to cover the entire solution space. On the other hand, a large Gaussian noise provides larger potential space for sampling hypotheses. But by increasing the noise, the solution space grows exponentially fast, this means the system will then requires a lot more particles to fill up the space with the right amount of density.

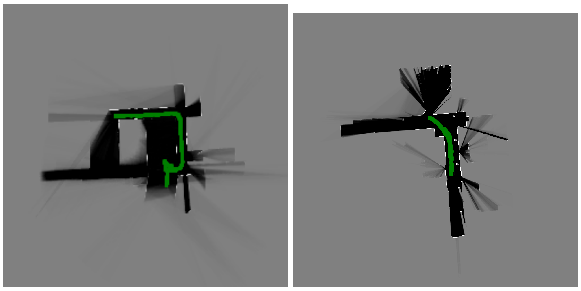

Figure 6. Lidar only mapping for dataset 1 and 2 (1000 particles)

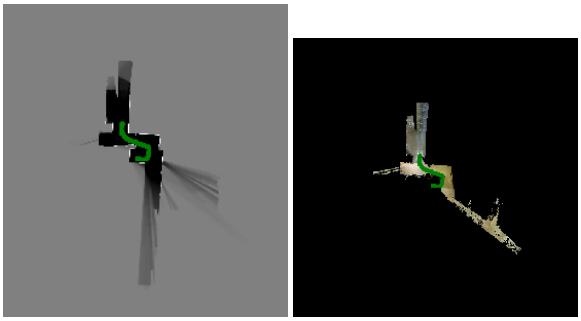

Figure 7. Lidar and texture Mapping for dataset 3 (2000 particles)

### 5.2. Number of Particles

As we have discussed the Gaussian noise in its effect to the stability of the system, we now move onto a close-related hyper-parameter in our particle system: how many particles to choose. The answer of this is also closely related to the variations among the particles which are caused by the Gaussian noise in the motion model. It is always reasonable to think the more particles there are the better solution the system results in, but by simply applying large amount of particles used in the system, it significantly slows down the performance as for each update, we performed map correlation to each of the particles. Therefore, choosing the right amount of particles to use is essentially the balance between efficiency and performance.

### 5.3. Down-sampling Ratio in Predict and Update to the Particles

The down-sampling ratio in the particle filter SLAM is the ratio between the number of prediction (applying the motion model) versus the number of updates (based on the best particles and observation model). Down-sampling ratio is also an important factor in balancing efficiency and performance of the system. If the system has a small Gaussian noise applied to the motion model, then the variations through one motion model sometimes may not be enough to cover a good amount of potential solution space. Without the need of increasing number of particles, increasing the down-sampling ratio can make the system more efficient and also provides larger variations to the particle sampling

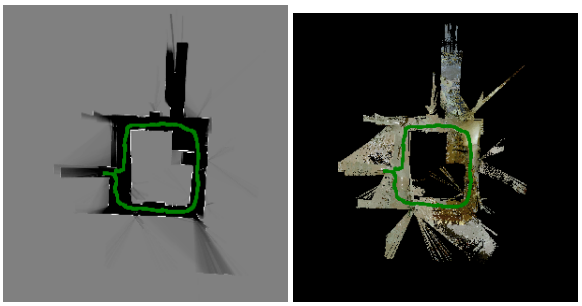

Figure 8. Lidar and texture Mapping for dataset 4 (5000 particles)

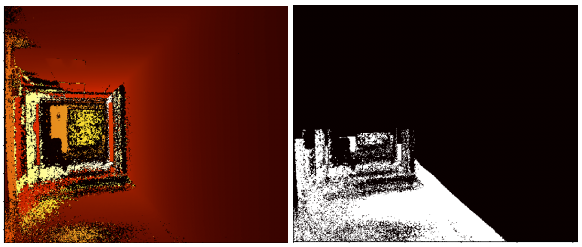

Figure 9. Example of back-projection and ground image segmentation

ability. However, if the sensor noise is constantly large and move very fast, by losing most of the update steps, the particle has less landmarks to hold on to, and this may cause significant drift in our robot pose estimation.

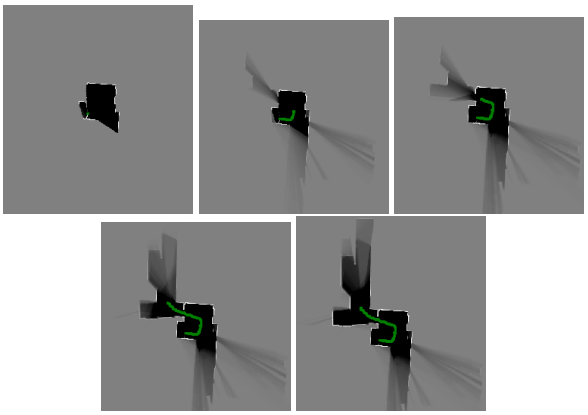

Figure 10. Progressive particle SLAM updates at different timestamps (1000 particles)