ESE650 Project 4:

Simultaneous Localization and Mapping

Nischal K N
nischal@seas.upenn.edu

Abstract
Simultaneous Localization and Mapping(SLAM) is a problem of locating the position of an object in an unknown environment and keep track of it as it moves in that environment. It also involves generation of a map of the surroundings. In this project we use data from a LiDAR range sensor to generate the map of the environment in which a humanoid robot, THOR is moving in. Also a texture map of the ground is generated from the RGBD data provided by a kinect camera. The mapping is achieved using Monte Carlo Localization.

Contents

1. Introduction

In this project we use the distances measured by a 2D laser range finder, use the odometry given by the kinematic model of the THOR and apply a particle filter with a Gaussian noise to localize as well as map the position of the robot in the environment it is moving in. The different data used and how they are accessed is explained in Section 2. Section 3 explains the various steps followed to perform SLAM. Section 4 explains the various tunable parameters use in the code and their values. The algorithm used to create the texture map is explained in Section 5. Finally the experiments and results are documented in Section 6.

2. Dataset

The dataset consists of data from lidar, kinect sensor, imu, joint encoders and kinematic model of THOR. The required data that was extracted and stored in a new file to simplify the process. This makes it easier to access the required data and also drastically reduces the dataset size preventing system overload. The script reFormatData.m performs this cleaning up. The following were the data extracted.

1. 2D lidar scans from an hokuyo 30lx LiDAR captured at a rate of 40Hz, with a maximum range of 30m and a 270^o view. The corresponding message time stamps are also recorded.
2. joint angles of neck and head indicating the yaw and pitch of the head with time stamps.
3. pose estimation from the kinematic model of THOR.
4. RGB and Depth images from the kinect camera along with timestamps.

3. Algorithm

The SLAM uses Monte Carlo localization with additional processing to remove ground and roof hits and skew correction due to tilt of the head. The basic flow of steps followed is shown in Fig. 1. Each of these steps are explained in the respective sections.

3.1 Ground Removal

The LiDAR is mounted on head that pitches up and down. This causes the LiDAR to pick up ground hits and detects them as walls. If these are not accounted for, then false walls may be recorded whenever the head pitches down or up. This removal was done by computing the following equation for each datapoint of LiDAR.

$$R \times sin(\theta) \times cos(\alpha) > H_{corrected}$$

If this condition was satisfied, then the scan was removed, else the scan was retained. $H_{corrected}$ was the recalculated height considering the tilt of the head

$$H_{corrected} = H_{actual} - 0.15 \times cos(\theta)$$

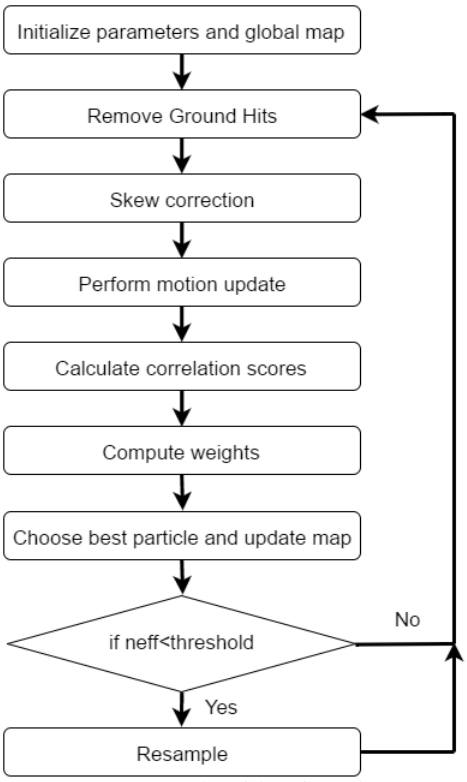


Figure 1. Algorithm flowchart

Where θ was the pitch of the head and α was the angle of the scan between -135^o to 135^o. R is the distance measured by the LiDAR. The result of using the raw scan and $Rsin\theta cos\alpha$ is shown in fig. 2.

Additionally a skew correction of the LiDAR scans are performed to compensate for the head angle. This is performed using

$$R_{new} = R \times cos\theta$$

This can be seen in 2c. The red is corrected scan and blue is the raw scan

3.2 Initialization

To initialize the SLAM algorithm, first the time stamps of LiDAR and joints are synced. Then a global map of a fixed dimension and resolution is initialized with 0 indicating unexplored area. Then the first scan is subjected to ground and skew removal. Then is placed on the center of the map by increasing the value of those cells of the global map where walls are assumed to be present by a logodd_occ value, and decreasing the value of the cells where free space is detected by logodd_free value. Also many parameters described in 4 are initialized here.

3.3 Fixing Odometry

The odometry provided by the kinematic model of THOR is not in the same global frame of reference as our map. Hence it has to be explicitly computed by measuring the change in state between local frames. The following steps were performed.

1. Compute delta increment in wrong global frame

$$[dxdyd\theta]_{wrong} = pose_{cur} - pose_{prev}$$

2. Convert to local frame

$$\begin{aligned} yaw_{wrong} &= pose_{\theta prev} \\ [dxdy]_{local} &= \begin{bmatrix} cos(yaw_{wrong}) & sin(yaw_{wrong}) \\ -sin(yaw_{wrong}) & cos(yaw_{wrong}) \end{bmatrix} \\ &\quad \times [dxdy]_{wrong} \\ d\theta_{local} &= d\theta_{wrong} \end{aligned}$$

3. Convert it to right global frame

$$\begin{aligned} [dxdy]_{global} &= \begin{bmatrix} cos(yaw) & -sin(yaw) \\ sin(yaw) & cos(yaw) \end{bmatrix} \\ &\quad \times [dxdy]_{local} \\ d\theta_{global} &= d\theta_{local} \\ x &= x + dx_{global} \\ y &= y + dy_{global} \\ \theta &= \theta + d\theta_{global} \end{aligned}$$

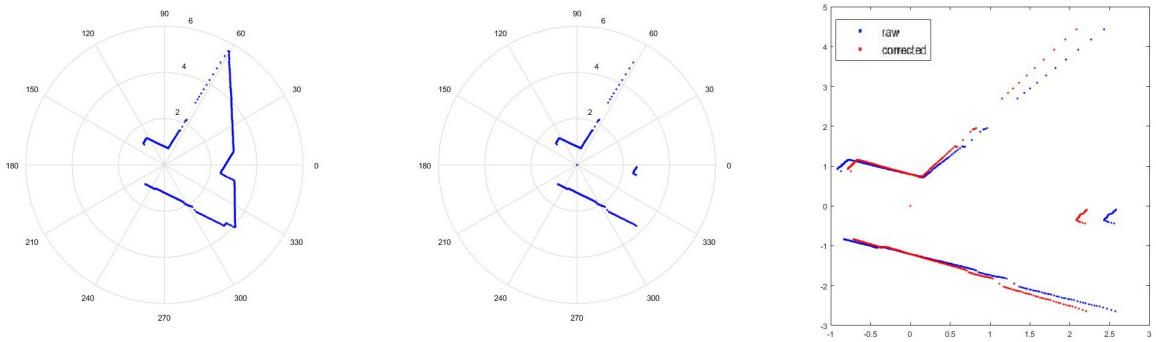
x,y and theta is the new corrected global odometry that can be used for motion update of particles.

3.4 Particle filter

A fixed number of particles were used. For each particle, a motion update was performed with a different noise parameter to the corrected global odometry. This resulted in n different orientations and position of LiDAR scans to be correlated with the global map. The noise parameters choosen are mentions in 4.

3.5 Correlation

For this project, both map_correlation mex file and corr2 matlab function was experimented. It was seen that both of them yield a similar result but corr2 being very slow compared to the mex file. The inputs to the mex file was positive version of the global logodds map. The negative values were reduced to zero and only the positive values were retained. The lidar scans passed to this fucntion were corrected for ground hit removal and skew correction. The correlation scores for each particle was recorded. It was multiplied with the weight of each particle and then normalized. The pose of the particle with the highest weight was chosen as the pose of the robot for that frame. This pose was used to update the global map by increasing and decreasing the logodd_occ and logodd_free values for walls and free space respectively. An upper and lower bound for logodd values were maintained.



(a) Raw lidar scan **(b)** Ground Removal **(c)** Skew correction

Figure 2. Ground hit removal

3.6 Re-sampling

When ever the majority of weight of the particles were accumulated within a few particles, the particles were re-sampled based on the current weight. More particles were chosen from the particle with more weight and vice-a-verse. A random sampling was performed. Due to high number of particles used, there was no difference between the performance of random sampling and low variance sampling.

4. Selecting Parameters

A number of parameters were used in the code that were tunable, the most prominent one that affected the performance were the motion update noise parameters, logodd saturation values and number of particles. The various parameters values used were

1. **No of Particles: 100**, less particles performed well then the motion was relatively straight like in dataset 2 and 1. But failed when there were turns.

2. **Number of Effective Particles: 40**

3. **Motion noise: 10cm(x&y),** 5^o**(theta)**, lower noise increases the trust on odometry and increasing it decreases the performance as particles are scattered over a wide area resulting in the need to increase the number of particles which in turn increases computation.

4. **logodd_occ = 2.5 and logodd_free = 0.625**, this is because the the datapoints of walls are far less compared to the free space data points.

5. **Map saturation threshold: 100 and -100**

6. **Map resolution: 5cm** Increasing the resolution increases the computation time but also improves the map quality.

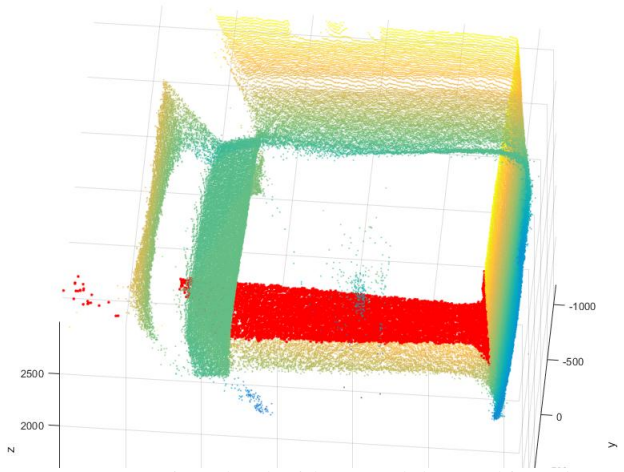


Figure 3. Point Cloud with ground detected in red

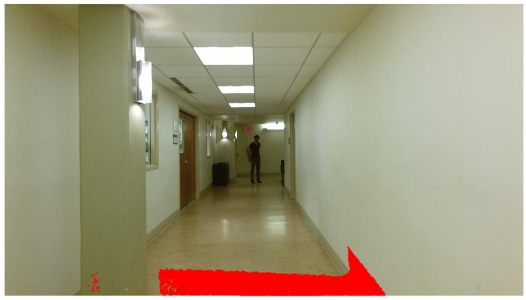


Figure 4. RGB image with detected ground points

5. Texture map

Once the map was generated by SLAM, the kinect's RGBD data was used to generate the floor texture of the the map.

Following were the steps followed

1. The depth data, z were thresholded between 500 to 3000, the usable range of kinect data.

2. The depth image was projected to a 3D point in space generating a 3D point cloud as shown in fig. 3 using

$$x = \frac{u \times z}{fc}, y = \frac{v \times z}{fc}$$

where u and v are pixel coordinates, z is the depth and fc is the focal length.

3. The ground plane was detected from the point cloud by determining the points that lie on the plane satisfied by the equation

$$a_0 x + a_1 y + a_2 z + a_3 < \varepsilon$$

where a_0, a_1, a_2 and a_3 are 0,-1,0 and h. ε is the threshold. The head pitch θ is also accounted for using

$$\begin{bmatrix} a'_0 \\ a'_1 \\ a'_2 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & cos\theta & -sin\theta \\ 0 & sin\theta & cos\theta \end{bmatrix} \times \begin{bmatrix} 0 \\ -1 \\ 0 \end{bmatrix}$$

4. The depth coordinates are transformed into RGB coordinates using $rgb_{x,y,z} = R \times depth_{x,y,z} + T$

5. The points are projected back onto RGB image using shown in fig. 4

$$u = \frac{fc \times x}{z}, v = \frac{fc \times y}{z}$$

6. The ground points are also converted to LIDAR frame using, (R and T between lidar and kinect is ignored)

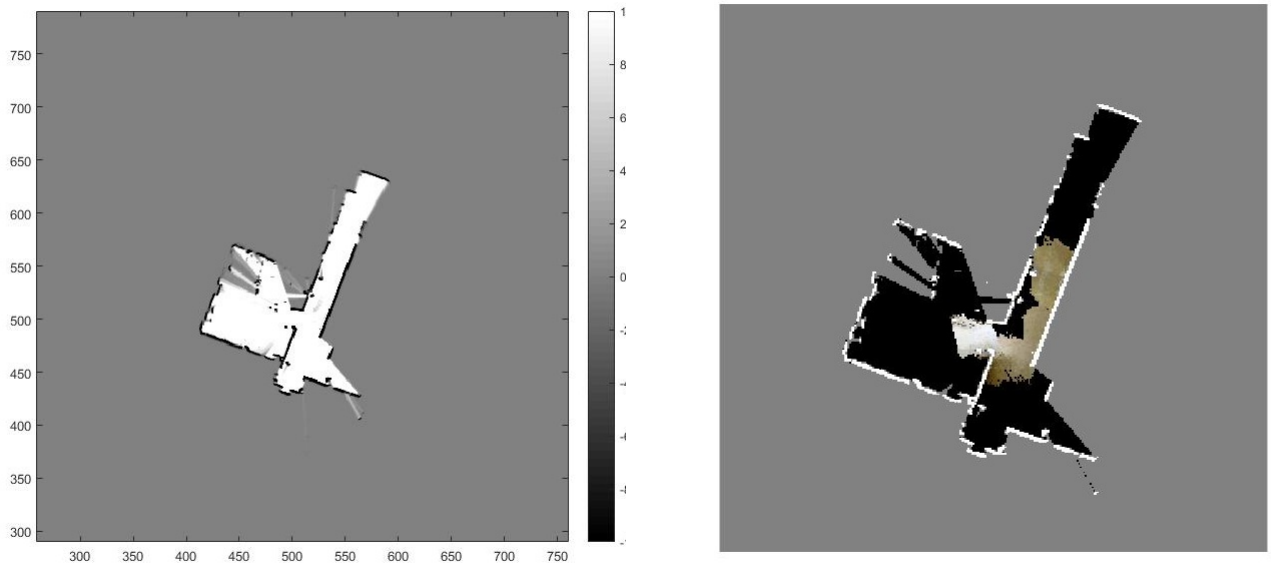
$$LiDAR_{x,y,z} = \begin{bmatrix} 0 & 0 & 1 \\ -1 & 0 & 0 \\ 0 & -1 & 0 \end{bmatrix} \times depth_{x,y,z}$$

7. These points are transformed with the pose of the best particle and then corresponding RGB pixel values are copied to the global map.

6. Experiments and Results

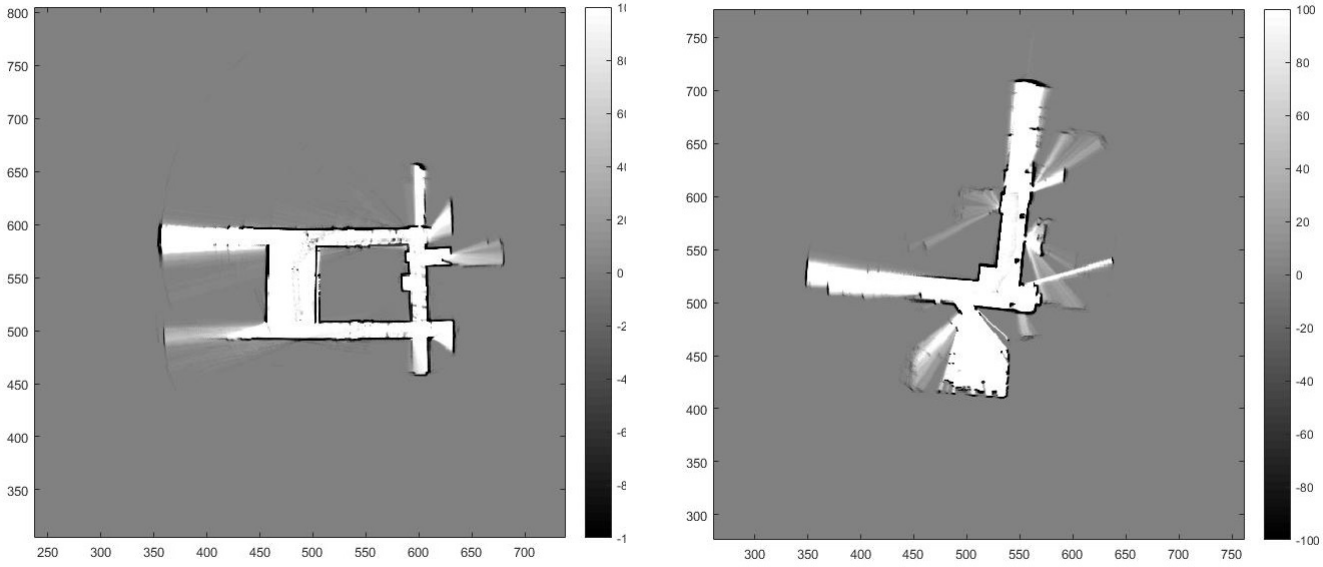
The algorithm was ran on 4 training datasets and also on the testing set. The output was consistent with the actual corridors. They are shown in fig. 5 to fig. 8.

The video of color and RGB map generation for test set is present at Grayscale map: **https://youtu.be/ykQvQrcE3Xk** Texture map: **https://youtu.be/SuZ3QX5SUrM**



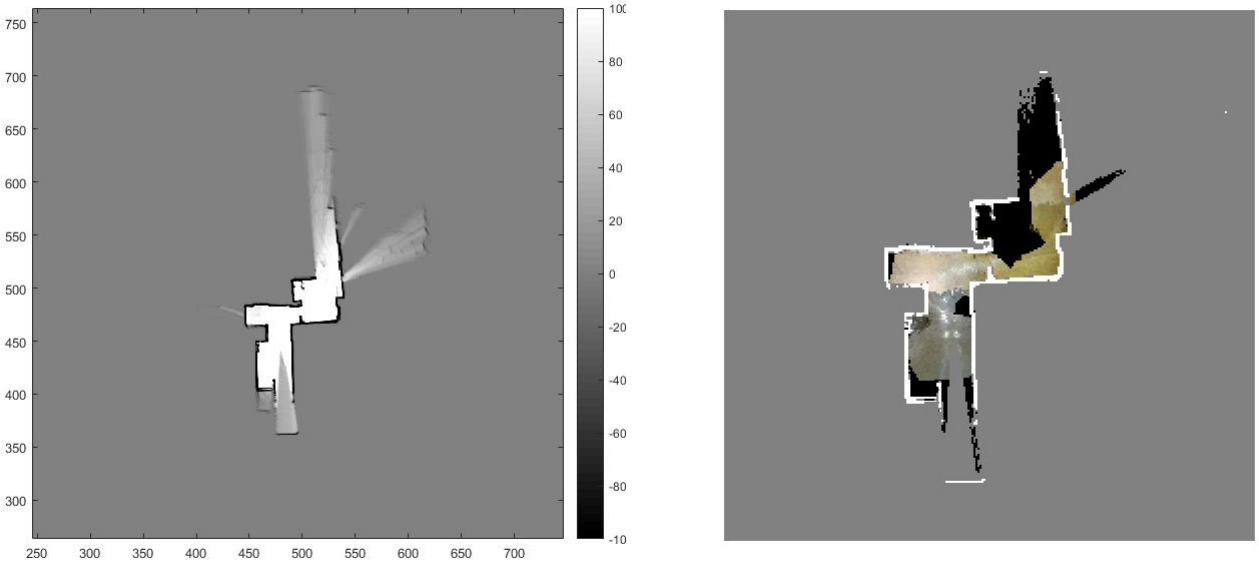
(a) gray scale Map **(b)** Floor Texture

Figure 5. Training Dataset 0



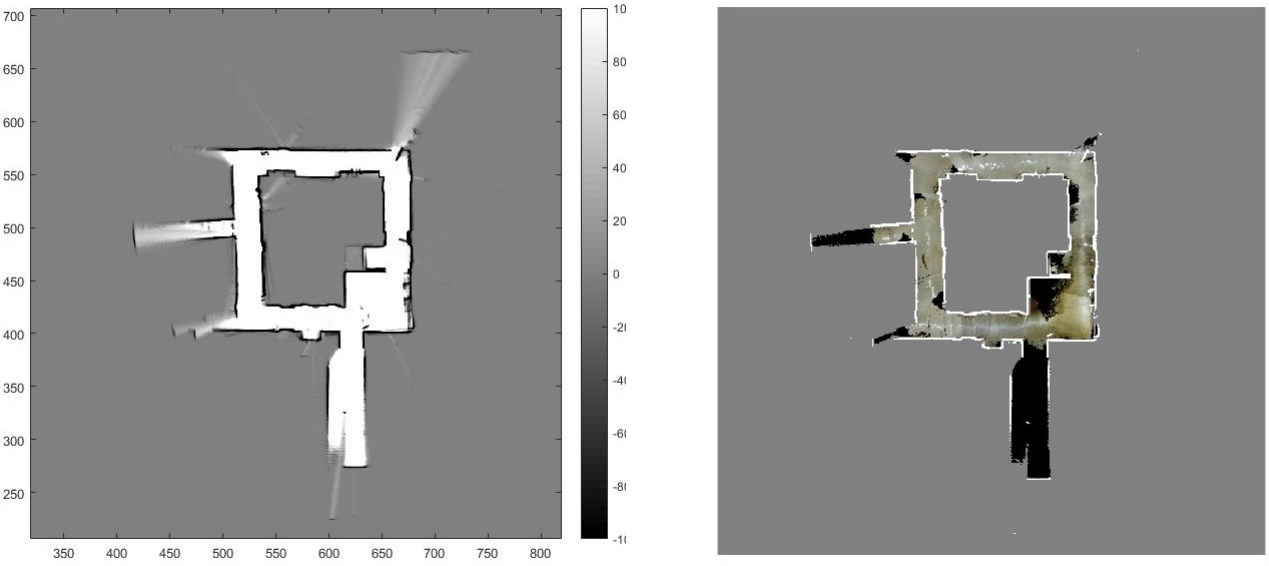
(a) gray scale map for training dataset 1 **(b)** gray scale map for training dataset 2

Figure 6. Training Dataset 1 and 2



(a) gray scale Map **(b)** Floor Texture

Figure 7. Training Dataset 3



(a) gray scale Map **(b)** Floor Texture

Figure 8. Test Dataset