Simultaneous Localisation and Mapping using the Particle Filter

Rachit Bhargava and Nikolay A. Atanasov

***Abstract*— This work addresses the task of accurately estimating the odometry of a humanoid robot and a 2D top view map of its surroundings using the raw odometry, the LIDAR and the IMU data. The robot state at each time step is represented chosen from a set of particles, each represented by a delta function. The raw odometry and the IMU were used to extract the control action that is used to update the particles. The best particle from the set, extracted after scan matching, is used to update the odometry and the map. The effectiveness of the algorithm was exhibited through videos by a side by side comparison of the robot moving in the real world and the result of the SLAM algorithm.**

I. INTRODUCTION

One of the most important problems in robotics is estimating the odometry of the robot and simultaneously estimate the map of the surroundings. This is known as the Simultaneous Localization and Mapping or SLAM problem. One way to solve the odometry estimation problem was the Kalman Filter and its variants as discussed in the previous project. Another popular state estimation algorithm is the Particle Filter, a Monte Carlo localization technique. In this project, the readings from a planar LIDAR scanner, raw odometry of the THOR robot, shown in Fig. 1 and the roll pitch yaw readings of the IMU are used to update the state of the robot and the map in its surroundings. The particle filter involves using this data to choose the best particle among several randomly distributed particles to perform the updates. In this report, a particle refers to a Delta distribution, having a zero covariance and some real valued mean. The following section discusses the data given to perform the SLAM. This is followed by a description of the implementation of the particle filter algorithm. The final section discusses the results of the algorithms and possible improvements that can be made to improve the results.

*This work is a project as part of ESE 650 Learning in Robotics at the University of Pennsylvania

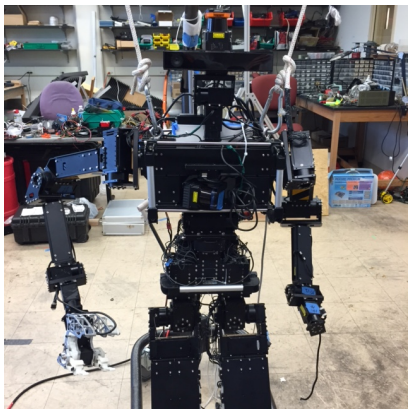


Fig. 1: THOR robot used for the project

II. DATA

This section discusses the data that was provided for the THOR robot that was used by the particle filter.

A. LIDAR

The Hokuyo LIDAR gives 2D laser scans at a frequency of 40 Hz. The range of the scans is from 0.1m to 30m. The scans span an angle of $270°$ from $-135°$ to $135°$ with the straight line. The resolution of the scans is $0.25°$. Thus at each time step the laser scans give 1081 readings, one distance value associated to each scan line. These distance value or range gives the distance the laser scan line traveled before encountering an obstacle. An visualization of the scan is shown in Fig. 2. At first, range values greater than 30m are discarded since they are beyond the working range of the LIDAR. The coordinates of the 'hit' points (points where the laser hits the obstacle) is estimated using the range values and the angle associated to the

scan line as:

$$x_s^l = range_l * cos(angle_l) \quad (1)$$

$$y_s^l = range_l * sin(angle_l) \quad (2)$$

$$z_s^l = 0 \quad (3)$$

Note that z_s^l is 0 for each point since the coordinates extracted above are in the local frame where the points all lie in a plane. Thus, at maximum 1081 'hit' points are attained. This data is used to estimate the world frame coordinates of the obstacles as will be discussed later.

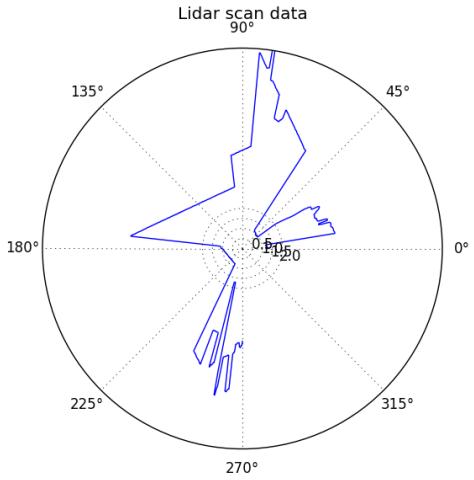


Fig. 2: LIDAR scan at pose 0 for training dataset 0. The blue points indicate obstacles.

B. Body Pose, IMU readings and Head angles

The body is assumed to be moving in a plane environment. Thus only the position x and y and yaw θ are used to represent the state of the body. For each time step, the body pose estimated using the dynamics of the THOR have been provided. Since the dynamics of the robot can never be perfectly calculated, these estimates consider a considerable amount of error. Hence the need for localization. The pose values given were used to initialize the position, the x_w and y_w coordinates of the robot. The initialization odometry for training dataset 0 is exhibited in Fig. 3.

The IMU comprises of a gyroscope and an accelerometer. The gyroscope readings can be used to get a pretty accurate reading of the yaw value. This yaw value is used as an initialization of the body yaw θ_w of the robot.

Finally, data was also provided about the angles of the head, pitch θ_p (forward backward notation) and yaw θ_y(rotation about the neck axis) with respect to the body.

In addition to this data, information is also provided about the height of the robot and the distance between the various sensors.

All of this data is used to transform the 'hit' points in the local frame to the world frame. First, the 'hit' coordinates in the frame of the scan are converted to the head frame using the following transformation matrix:

$$T_l^h = R_z(\theta_y) * R_y(\theta_p) * R_{transz}(L2H) \quad (4)$$

where R_z, R_y and R_{transz} represent the rotation about z, rotation about y and translation about z axis respectively. L2H represents the distance of the LIDAR to the base head of the THOR when the neck (yaw) and head (pitch) angles are 0.

Next, the coordinates are transformed from the head frame to the base frame using the transformation:

$$T_h^b = R_{transz}(H2B) \quad (5)$$

where H2B is the height of the head from the base of the body / ground.

Finally the coordinates are transformed to the world frame from the base frame using the transformation:

$$T_b^w = R_{transx}(x_w) * R_{transy}(y_w) * R_z\theta_w \quad (6)$$

As mentioned before, $(x_w, y_w, \theta_w,)$ represents the state of the robot.

It is important to note above that each of the transformations described above are of the form SE(3) 4x4 matrices.

Thus the coordinates of the hit points $(x_s^l, y_s^l, z_s^l,)$ can be converted to the world coordinates as follows:

$$[x_s^w, y_s^w, z_s^w, 1]' = T_b^w * T_h^b * T_l^h * [x_s^l, y_s^l, z_s^l, 1]' \quad (7)$$

The equations derived above will be very important in later sections.

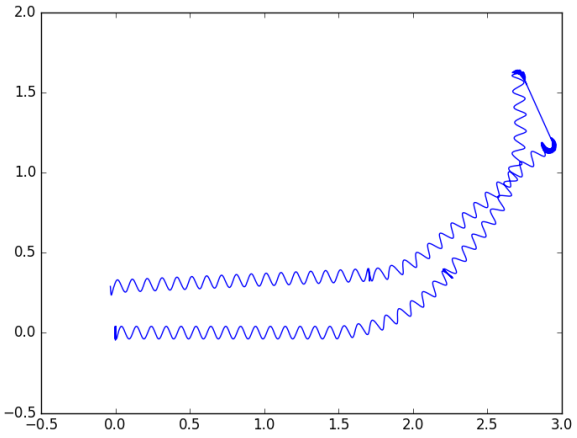


Fig. 3: Trajectory of the humanoid for dataset 0 as estimated by using the odometry data.

C. RGB images and Depth images

Finally, RGB and Depth images have also been provided. Information about the intrinsic parameters for both these cameras has also been provided. This information will be used to verify the trajectory of the robot.

III. Particle Filter

This section describes the initialization of the particle filter and the map, the prediction step, the and the parameters affecting its performance.

A. Initialization

Before beginning the loop for SLAM, the map and the particles need to be initialized.

1) Map: The first step involves initializing the map. First the map is set up by deciding the size and resolution. The entire map is initialized to 0. Furthermore, a log-odds map is initialized to all 0 to suggest that the probability of the cell being occupied (black) and unoccupied (white / gray) in the map is 1/2. Formally the log-odds is related to the probability of cell m being occupied $p(m = 1|z)$ given measurements z as:

$$p(m = 1|z) = 1 - \frac{1}{1 + exp(\lambda(m = 1|z))} \quad (8)$$

To initialize the map, the log-odds map must be updated. This is done by calculating the coordinates of the scan hits in the world frame using the initial pose of the robot and the initial laser scan. The log odds of the coordinates which are occupied are updated by $log(9)$. All the points along the scan line which are not occupied are updated by $log(1/9)$. Finally, those cells which have a log odds value greater than 0 (or probability greater than half) are assigned the value of 1 and those which are below are assigned 0.

2) Particles: Particles are basically Delta distributions with 0 covariance but some real valued mean which in this case represents one of the possibilities of the pose of the robot at a particular time step. Besides the pose, each particle has some weight associated which is determined by the correlation factor which is discussed later. Initially all the particles are initialized to the starting pose and have a weight equivalent to 1/(number of particles). For this project 100 particles were assumed.

Once the map and the particles have been initialized the loop for the particle filter begins.

B. Prediction step

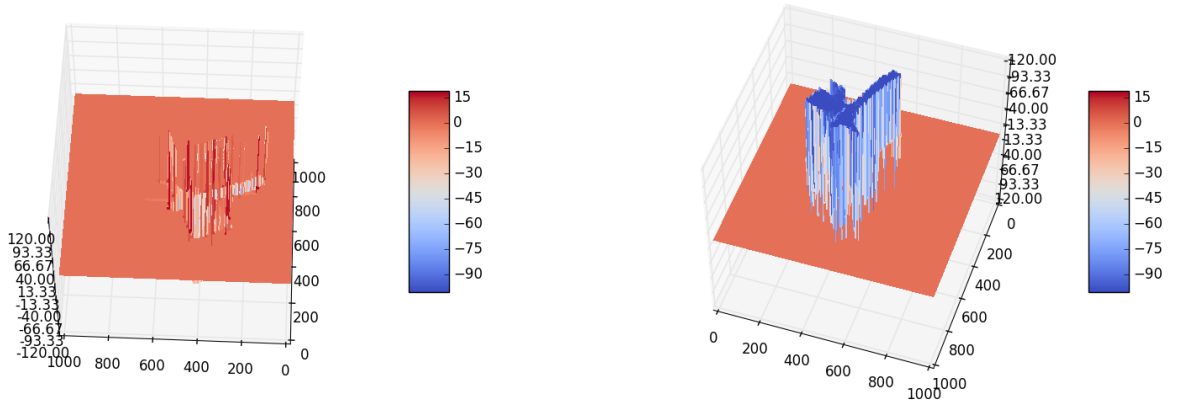
The loop begins by finding the control action. Given the odometry the change in position is estimated as the difference in the positions of the current and previous time step. The change in position is then converted to the local frame using the rotation matrix formed using the yaw from the previous pose. The change in yaw is found by taking the difference in the yaw derived from the IMU at the current and previous time step. The change in yaw and position in the local frame form the control action c. 100 different noise samples are extracted from a multivariate Gaussian centered at 0 and having some user defined covariance matrix. Thus upon combining the control action and adding noise, 100 different control actions are attained.Given noise n_i the particle i is updated as,

$$p^i_{t+1} = (p^i_t \oplus c) + n_i \quad (9)$$

where $\oplus$ is the smart plus explained in class to apply the transformation $c = (\delta x, \delta y, \delta\theta)$ on the current pose of the robot (x, y, θ). In such a manner each particle's pose is updated.

C. Correlation using Log Odds and Weight Update

Using the pose of the particle, the hit coordinates, which the scan believes to be occupied, are



(a) Top view (with positive z axis going up)

(b) Bottom view (with positive z axis going down)

Fig. 4: Log odds map for Dataset 0

estimated in the map using (7). Following this, the correlation of the particle is estimated. In the present algorithm, instead of using the binary map to find the correlation, the log odds map was used.

A new temporary map was created which was basically the log odds map scaled to a value such that the cell with the maximum log odds value has a value of 100. For each particle, value of the cells in the temporary map at the coordinates which the particle believes were occupied were summed up. This value was divided by 100 to get the correlation. The advantage of this method is that cells which have a higher log odds are weighted more when estimating the correlation, which makes sense since its occupancy concurs with multiple scans. Equally weighting all the occupied cells seemed like an incorrect way to estimate the correlation since a lot of information is lost in the binary map assumption.

The weights of the particles were then updated using these correlation values using the relations,

$$\begin{aligned} l_i^t &= log(w_i^t) + corr_i \\ l_{max}^t &= max(l_i^t) \forall i \in [1...N] \\ logsum &= l_{max}^t + log(\sum_i \exp^{l_i^t - l_{max}^t} \\ w_i^{t+1} &= \exp^{l_i^t - logsum} \end{aligned}$$

where N is the number of particles and w_i^t and $corr_i$ are the weights and correlation of particle i at time step t. Once the weights are updated, the particle with the highest weight is chosen as the best particle.

D. Update Step

The best particle is used to update the map and the odometry. The pose of the particle is directly used as the odometry of the robot at the next time step. The cells that the best particle believes are occupied are incremented by a value of log(9). On the other hand cells along the scan line from the robot position to the occupied cells were incremented by 0.5*log(1/9). This was done using the cv2.drawcontours() function by using the hit coordinates (cells believed to be occupied by the best particle) and robot position as the vertexes. Once the log odds are updated, the map is recalculated by setting values where log odds is greater than 0 as 0 (black) and values where log odds is less than 0 as 255 (white). The pose of the best particle was marked by coloring the appropriate pixel as red.

It is important to note that a limit of 100 and -100 was put on the log odds value so that no cell becomes exceedingly confident in its occupied or unoccupied. It is highly possible that a cell may be erroneously updated in the wrong direction multiple times. This constraint ensures that the cell occupancy can be updated to a correct value even when its log odds were incorrectly updated

multiple times due to previous scans. Fig 4. shows the log odds texture map for training dataset 0 after implementing the SLAM algorithm.The points corresponding to walls have high log odds as seen in Fig 4a. In Fig. 4b. it can be clearly seen that the "empty" cells have very low log odds values.

E. Stratified Importance Sampling

After performing the update step the effective number of particles p are estimated as,

$$p = \frac{1}{\sum_i w_i^2} \tag{10}$$

This value gives an indication of how the particles are dispersed or spread out. For the particle filter to work effectively, the particles must be sufficiently spread out or dispersed to have a greater range of poses of which the filter can estimate the best pose.

If the number of effective particles falls below a threshold, the particles need to be resampled. Stratified Importance sampling (algorithm explained in class slides [1]) is the method used to resample from the existing set of particles. This method is more likely to sample from particles with higher weights in comparison to lower weights. Upon sampling, it resets the weights of all particles equal to 1/(number of particles).

IV. Results

The performance of the algorithm relied heavily on how the parameters were tuned. Thus, it is important to discuss the measures taken to improve the performance of the algorithm before discussing the quality of the results.

Noise: The performance was really sensitive to noise.By plotting the x and y values of the odometry, it was observed that it was really inaccurate. Since a jump of 10 time steps was taken, a noise of 10 cm seemed like an apt value for the position. Since a large number of particles were being used during the algorithm, a noise of 0.01 proved to be a good estimate for the yaw. Note that increasing the yaw allows testing a greater range of values but also increases the chance of an incorrect particle being chosen if the number of particles is not large enough and the number of time steps are not small enough.

Correlation using log-odds map: As mentioned in section 3C, using log-odds map to find the correlation ensures that the cells which concur with more scans are weighted more during scan matching.

Log-odds limits: As mentioned in section 3D. a limit of -100 and 100 was implemented on the log-odds to ensure any cell does not become too confident about being occupied / unoccupied.

Updating log-odds: When occupied the log-odds was incremented by log(9) but was decremented by 0.5*log(1/9) when unoccupied. This is because it was observed that scans are often more accurate when cells are occupied than when they are occupied (especially at large distances). Thus occupied cells should be weighted by a greater extent.

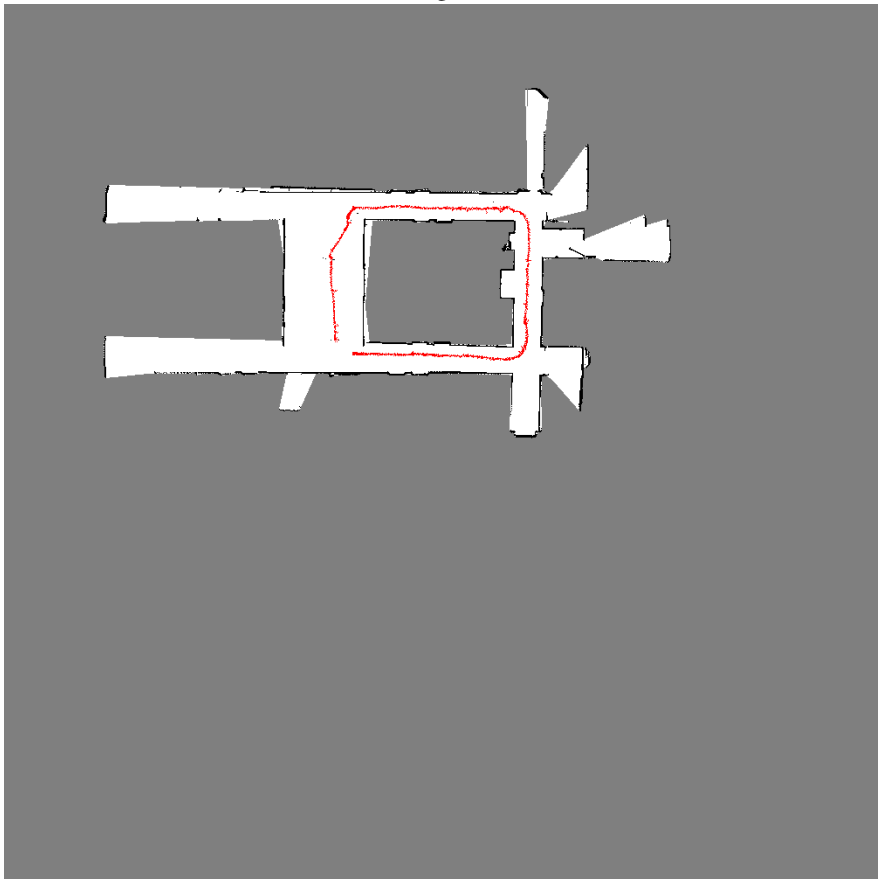
Correlation grid: A grid of 5x5 was assumed around the current particle position to account for errors in the measurement of the scan distance values (due to some vibrations for example). Note that if the maximum correlation did not occur at the center of the grid (the position of the particle), the pose of the particle was updated to coincide with the position of maximum correlation.

Number of particles: Since the map correlation function and the code to get the cells from the rays was written optimally and a time jump of 10 was taken every iteration, the code ran extremely fast. In fact if the number of particles was taken as 100, the code runs under a minute for the datasets 0, 2 and 3 with less than 12000 time steps. But for larger datasets, like dataset 1 and the test dataset (with more than 25000 time steps), 1000 particles could be assumed. For this case the code completed within 30 minutes with very good accuracy (from visual quality of the map.)

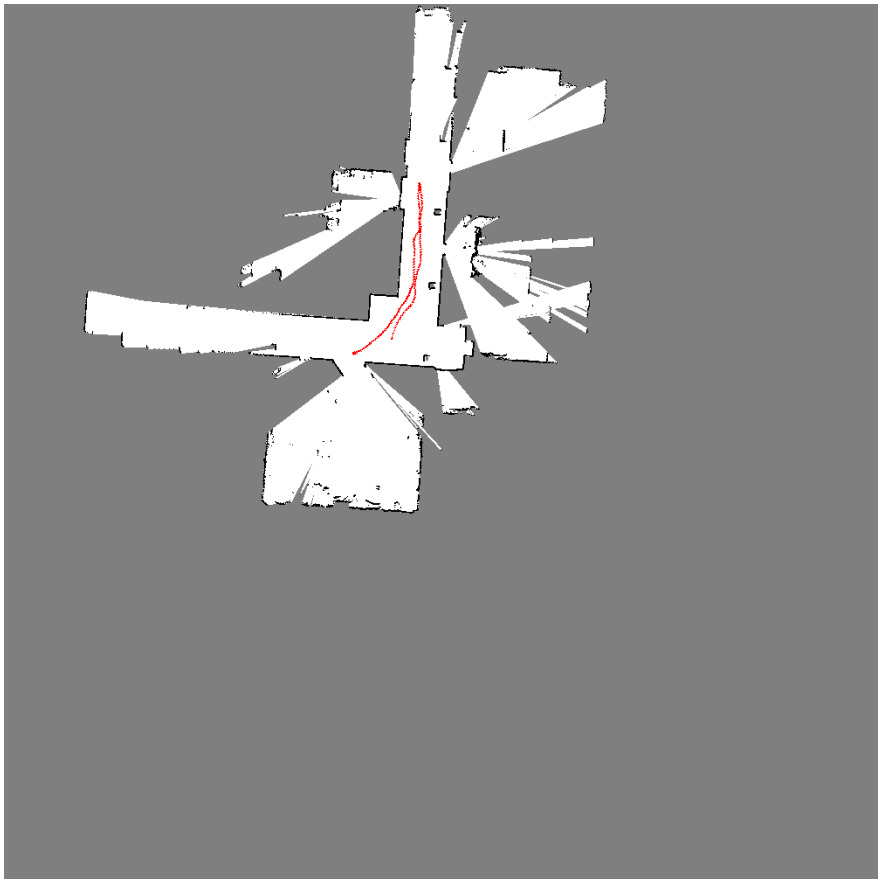
Upon properly tuning these parameters, very good results with aberrations were observed as can be seen in Fig 5. For each of the results, a noise of 0.01 for all 3 state parameters, correlation grid of 5x5, 1000 particles, time jump of 10 time steps and log odds limits of -100 and 100 was implemented. Fig 5. displays the results using these parameters. The code implemented to get these results can be found here : Project 3

The efficacy of the algorithm can be observed from its performance on dataset 1 in the training set (Fig 5b.) and the testing set (Fig 5e.) where the cells corresponding to walls perfectly match up as

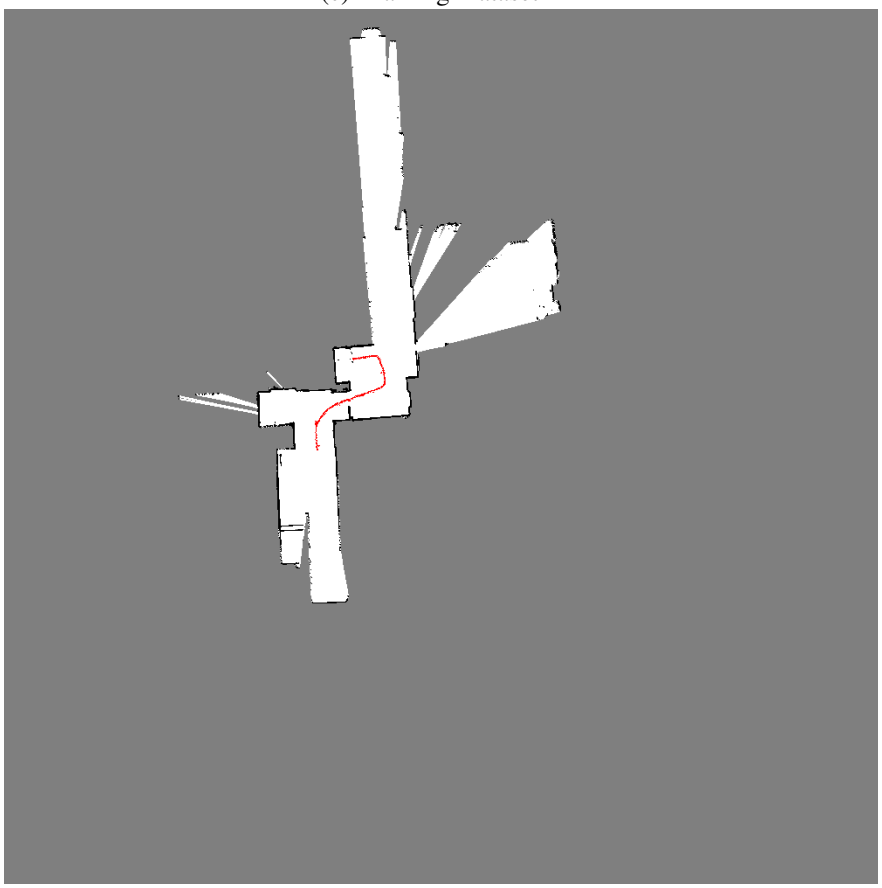
(a) Training Dataset 0



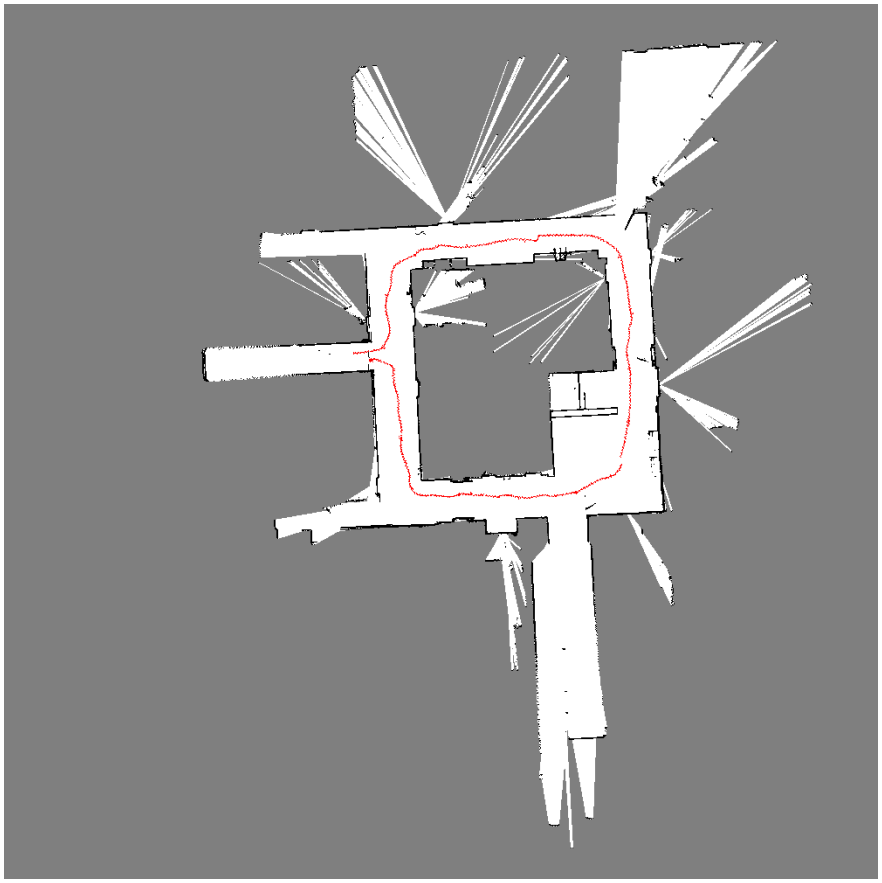
(b) Training Dataset 1



(c) Training Dataset 2



(d) Training Dataset 3



(e) Test Dataset

Fig. 5: Performance of the presented algorithm on various datasets

the robot completes its looped trajectory. There is negligible drift in the map when the robot visits the same area again. This implies that the SLAM algorithm is able to estimate the pose correctly throughout the trajectory and avoid accumulation of error.

To visualize the results, a video comparing the generated map and trajectory to the real world motion of the robot was made. This was done by matching the time steps of the RGB images to the time steps of the LIDAR. The videos can be found at Videos

V. CONCLUSIONS

In this paper, an algorithm to accurately estimate the odometry of a humanoid and simultaneously generate the map of its surroundings has been presented. The algorithm performed pretty well on all the datasets as seen in the results section.

This project helped develop a very good understanding about processing LIDAR scan data, particle filters, scan matching and of course SLAM. It helped develop an intuition about the common problems that can be encountered during implementation of state estimation and map generation algorithms. In future, I hope to work on 3D variants of this algorithm using the depth data provided.

ACKNOWLEDGMENT

I want to thank Dr. Atanasov and the TAs for helping me with my doubts on Piazza.

REFERENCES

[1] Slides for ESE 650 from Dr. Atanasov