Particle Filter SLAM

Abhiram Iyer
University of California, San Diego
abiyer@ucsd.edu

***Abstract*—This paper presents an approach to using particle filters and LIDAR data to perform SLAM (Simultaneous Localization and Mapping). The LIDAR data is collected from a robot that travels through various indoor environments.**

I. Introduction

The task of SLAM for mapping and localization is widely used today in robotics and navigation and is a tool for determining the environment around a robot and its approximate position within it.

In this paper, we present an algorithm that uses SLAM given a series of LIDAR scans and odometry measurements taken by a walking robot (nicknamed "THOR") as it traverses through a few indoor environments. Using these LIDAR scans, we generate a trajectory of the robot's travel as well as a map of the environment it encounters. Specifically, we use particle filters and a laser-grid correlation observation model for localization and a LIDAR-based occupancy grid for mapping.

II. Problem Formulation

The THOR robot has a center of mass 0.93 meters above the ground. 33 cm above the center of mass lies the head upon which rests both the LIDAR and Kinect sensors. More specifically, the LIDAR sensor is 15 cm above the head and the Kinect sensor is 7 cm above. The LIDAR sensor has a 270 degree field of view (135° to its left and 135° to its right). With an angular resolution of $0.25°$, each LIDAR scan produces a total of 1081 measurements (i.e. 1081 laser rays). The LIDAR sensor has a maximum range of 30 meters. Both the LIDAR and Kinect sensor are fixed on the head and do not rotate individually. However, the head and neck of the robot can rotate up or down and left or right respectively.

Given a set of observations $z_{0:T}$ (i.e. LIDAR scans) and control inputs $u_{0:T-1}$ (i.e. to describe movement, also from the robot), generate an occupancy grid *m* that describes the global environment and estimate $x_{0:T}$ that describes the robot's trajectory within this environment. In other words, given this trajectory $x_{0:T}$, build the environment map *m* while simultaneously using *m* to estimate a trajectory $x_{0:T}$. Mathematically, this means:

$$\max_{x_{0:T},m} p(x_{0:T}, m, z_{0:T}, u_{0:T-1}) \tag{1}$$

where $p(x_{0:T}, m, z_{0:T}, u_{0:T-1})$ is the joint probability density function over the robot trajectory, map, observations, and controls. To implement this, use Bayesian Inference to maintain the posterior likelihood of the parameters *m* and $x_{0:T}$ given the data: $p(x_{0:T}, m \mid z_{0:T}, u_{0:T-1})$.

Thus, create an algorithm that uses LIDAR scans, odometry measurements, and the head and neck angle of the robot to generate the map that the robot "sees" and plot its trajectory within this environment.

III. Technical Approach

A. Overview of SLAM

The SLAM problem can be summarized as the following:

$$p(x_{0:T}, m, z_{0:T}, u_{0:T-1}) \tag{2}$$

$$= p_{0|0}(x_0, m) \prod_{t=0}^{T} p_h(z_t \mid x_t, m) \prod_{t=1}^{T} p_f(x_t \mid x_{t-1}, u_{t-1}) \tag{3}$$

where p_h is the observation model and p_f is the motion model. Equation (2) can be established from (1) because of the decomposition of the joint probability density function due to the Markov assumptions.

From a basic formulation of SLAM, we can extend it to particle filters. The particle filter uses a mixture of delta functions (particles) with weights $\alpha^{(k)}$ to represent $p_{t|t}$:

$$p_{t|t}(x_t) = p(x_t \mid z_{0:t}, u_{0:t-1}) = \sum_{k=1}^{N_{t|t}} \alpha_{t|t}^{(k)} \delta(x_t; \mu_{t|t}^{(k)}) \tag{4}$$

Similarly, $p_{t+1|t}$ can be expressed as:

$$p_{t+1|t} = \sum_{k=1}^{N_{t+1|t}} \alpha_{t+1|t}^{(k)} \delta(x_{t+1}; \mu_{t+1|t}^{(k)}) \tag{5}$$

Since $p_{t+1|t}(x)$ is a mixture probability density function with components $p_f(x \mid \mu_{t|t}^{(k)}, u_t)$, it is possible to approximate this function by drawing samples from it. Thus, the prediction function for SLAM can be derived as follows:

$$p_{t+1|t}(x) = \int p_f(x \mid s, u_t) \sum_{k=1}^{N_{t|t}} \alpha_{t|t}^{(k)} \delta(s; \mu_{t|t}^{(k)}) ds \tag{6}$$

$$\approx \sum_{k=1}^{N} \alpha_{t+1|t}^{(k)} \delta(x; \mu_{t+1|t}^{(k)}) \tag{7}$$

The prediction rule (i.e. applying the motion model here) predicts the position of the particles. We can update the

weights of each particle through the particle filter update rule, which can be derived by evaluating Bayes rule with the predicted delta-mixture probability density function:

$$p_{t+1|t+1}(x) = \frac{p_h(z_{t+1} \mid x) \sum_{k=1}^{N_{t+1|t}} \alpha_{t+1|t}^{(k)} \delta(x; \mu_{t+1|t}^{(k)})}{\int p_h(z_{t+1} \mid s) \sum_{j=1}^{N_{t+1|t}} \alpha_{t+1|t}^{(j)} \delta(s; \mu_{t+1|t}^{(j)}) ds} \tag{8}$$

$$\sum_{k=1}^{N_{t+1|t}} \left[\frac{\alpha_{t+1}^{(k)} p_h(z_{t+1} \mid \mu_{t+1|t}^{(k)})}{\sum_{j=1}^{N_{t+1|t}} \alpha_{t+1}^{(j)} p_h(z_{t+1} \mid \mu_{t+1|t}^{(j)})} \right] \delta(x; \mu_{t+1|t}^{(k)}) \tag{9}$$

Finally, after predicting and updating, it is often the case that the particle weights become close to zero because the particles are not accurate hypotheses of the agent's location. To avoid this problem, we also use particle resampling. If the number of effective particles $N_{eff} = \frac{1}{\sum_{k=1}^{N} (\alpha_{t|t}^{(k)})^2}$ falls less then a threshold, resampling will create a new particle set of equally-weighted particles by adding many particles to areas that had high weight in the previous set and few particles to locations that had relatively lower weights. We used stratified sampling in this paper in order to maintain optimality in terms of variance, since this version of resampling guarantees that samples with large weights appear at least once and those with smaller weights appear at most once.

Now that a general formulation of SLAM has been discussed, we can begin to discuss this paper's specific implementation.

B. Mapping

Mapping is done with an occupancy grid with a laser-grid correlation model. Estimating the probability density function of a cell in the occupancy grid (m_i) conditioned on past observations (LIDAR scans) is the equivalent of accumulating the log-odds ratio. The log-odds ratio is a ratio of true positives over false positives given an observation z_t:

$$\frac{p(m_i = 1 \mid m_i \text{ is observed occupied at time t})}{p(m_i = -1 \mid m_i \text{ is observed free at time t})} = \frac{80\%}{20\%} = 4 \tag{10}$$

Thus, we must keep track of each cell's log-odds value in order to generate an occupancy map - any cell with greater than 0 log-odds value will be considered to be occupied. Similarly, any cell with value less than 0 should be considered empty, and any cell with value equal to 0 should be unknown (i.e. not explored yet). Consistent with this rule, for any free space observed, we must decrement a given cell in the map by $log(4)$ and increment the cell by $log(4)$ if we observe to be occupied given z_t. For the purposes of this project, we created a modular "MAP" class that would keep track of all necessary details and handle all functionality behind the occupancy grid. We also experimented with constraining the log-odds values between a minimum and maximum threshold so as to avoid overconfident estimation - these will be discussed in further detail in the "Results" section.

Converting the LIDAR data into an occupancy grid is not a trivial process, as we must first transform these scans to the world frame (at the robot's base) from the LIDAR frame at the top of the robot. First, we remove scan points that are too close or two far with a simple threshold: only scans within 30 meters and 0.1 meters are considered. Next, we generate the scans into a 3D Cartesian-coordinate map by multiplying each scanned point by its cosine and sine counterparts.

After obtaining a map (with axes x, y, and z), we translate the scans down 15 cm to the head frame. From the head frame, we must multiply the coordinates by the transformation matrix ${}_BT_H$ to get to the body frame, where the rotation matrix is a product of R_z and R_y (R_x is the identity matrix) where R_z is:

$$\begin{bmatrix} cos(\alpha) & -sin(\alpha) & 0 \\ sin(\alpha) & cos(\alpha) & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{11}$$

where α is the neck angle the robot. Similarly, R_y is:

$$\begin{bmatrix} cos(\beta) & 0 & sin(\beta) \\ 0 & 1 & 0 \\ -sin(\beta) & 0 & cos(\beta) \end{bmatrix} \tag{12}$$

where β is the head angle of the robot. The "p" value in the transformation matrix ${}_BT_H$ is a 3x1 vector with the last element of value 0.33 to describe a translation down to the body frame from the head. At the body frame, we can remove all scan points that are less than or equal to -0.93 meters, since this is the equivalent of scans touching the ground.

After transforming the data down to the body frame, we must transform it to the world frame with another transformation matrix ${}_WT_B$. Here, the only rotational component is in the "z" axis, so R_z is:

$$\begin{bmatrix} cos(\theta) & -sin(\theta) & 0 \\ sin(\theta) & cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{13}$$

where θ is the overall yaw of the robot (i.e. which direction it is facing). The final map at the world frame is what we can use to populate our log-odds occupancy map. The OpenCV function "polylines" performs a rasterization technique and draws all the scanned rays onto the occupancy map. For every cell where the ray ends (presumably meaning that an object has been detected), we increase the log-odds by *log(4)*. For every cell that the ray passes through (meaning that there is probably empty space in that region), we decrease the log-odds by *log(4)*. We now move on to the localization aspect of this paper.

C. Localization

We use particle filters with an odometry-based motion model and laser-grid correlation observation model for localization. We start off with a set number of particles which we vary throughout our experiments (to be discussed more in the results section) that are zero initialized.

The motion model is first applied to each particle - a small amount of noise sampled from a Gaussian distribution with 0 mean and fixed covariance (also experimented with, to be discussed more in "Results") along with the delta pose values (values describing how the robot has moved from the last time step, given by the robot). The noise allows the particles to spread out to different parts of the map initially, and they eventually converge to the location of the agent.

After the motion model is applied, all scans are transformed down to the body frame. Because there are *N* particles, there are a total of *N* different transformations from the body to the world frame, since each particle represents a possible global location of the robot within the map. Once the *N* transformations are generated, we pick the best one with the "mapCorrelation" function provided to us, which uses a 9x9 grid to check how similar our baseline map is to each transformed observation in the world frame. Picking the highest element in this 9x9 grid yields a "score" value, and continuing in this manner for all *N* particles will yield *N* different correlation scores.

We then subtract the largest element from these *N* scores before normalizing them through a softmax function. The weights of the particles are then updating to be the product of the previous weights α and correlation scores calculated. The index of the maximum value (which represents the highest probability) of these new weights is found, which corresponds to the most accurate particle to use. This particle is saved, and essentially represents the most likely position of the robot at the current time step. Using the current particle, we update our map appropriately (described in the previous section), and we can generate an occupancy map for visualization.

After these prediction and update steps, we can choose to resample if the number of effective particles (defined in the "Overview of SLAM" section) falls below a certain threshold. We define a ratio to take care of this: for every *N* particles, at least 40% of the particles must be effective to avoid resampling. If less than 40% are effective, we must resample with stratified resampling.

D. General Approach

Before we present our results, we would like to discuss our general approach to this project.

The first step that we took was to use dead reckoning to estimate the map and the robot's trajectory. This baseline version involves calculating the trajectory only through the delta pose information provided by the robot. Once the dead reckoning occupancy maps were generated, we had a better idea of how to move forward. The code for our dead reckoning approach is included for reference.

In order to tune the various hyperparameters (number of particles, resampling threshold, log-odds constraint thresholds, covariance of noise) of this project, we downsampled. In other words, the particles accumulated the delta poses given in the training set for a fixed number of steps before being injected with noise. Conversely, without downsampling, the particles were added with noise and a delta pose for every time step, slowing down the algorithm and preventing us verifying changes in the hyperparameters. Once the hyperparameters were tuned, we executed the original code without downsampling. The code for downsampling is also included for reference.

IV. Results

We will now present our results for each training set provided. (As a note, all plots are by default flipped about a diagonal starting from the top left to the bottom right. The plots should still match.) The black color means empty cells, the white color means occupied cells, and gray signifies unknown or undecided. The trajectory of the agent is plotted in green and is overlaid onto the map. The particles are visible in some pictures, and are plotted as blue dots.

A. Training Set 0

The dead reckoning map we found for the first training set was as follows:

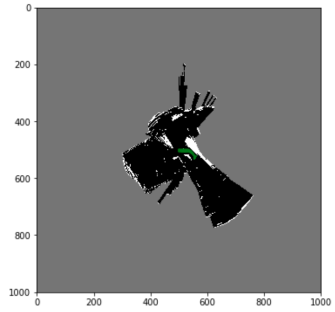

Fig. 1: Dead Reckoning for LIDAR0

We will first describe the methodology we followed to achieve some decent results. At the very beginning, we tried a 3x3 correlation grid for "mapCorrelation", along with 80 particles and a resampling threshold of 30. However, the results were incredibly noisy:

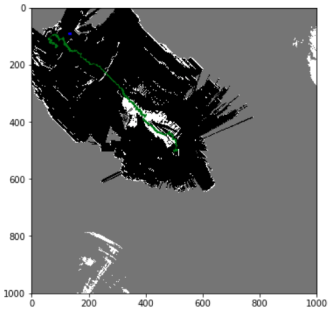

Fig. 2: Bad Example of LIDAR0

After changing to a 9x9 correlation grid for "mapCorrelation", the results were a little less noisy but still nonetheless bad.

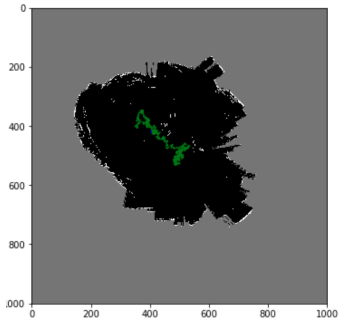

Fig. 3: Another Bad Example of LIDAR0

After downsampling to enable efficient hyperparameter tuning, we found that using 100 particles with a 40 particle resampling threshold along with lower amounts of noise for x, y, and yaw worked better. Additionally, our particle weight update previously involved resetting all the weights to just the scores we received from "mapCorrelation". For this run and onwards, our weight update involved multiplying the scores by the old weights and normalizing them which generated better results.

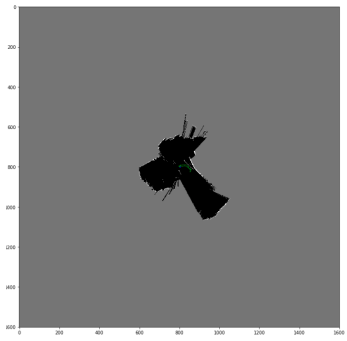

Fig. 4: Better Example of LIDAR0

Because there was little variation in yaw in this training example, the trajectory was relatively smooth. Small covariance weights of 10e-5 were used for x, y, and yaw to achieve good results. The following pictures describe the trajectory and environment being generated over time. These images are the best versions we could generate with SLAM.

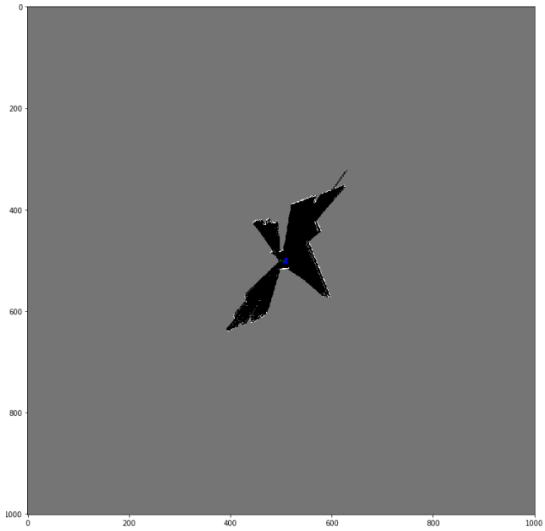

Fig. 5: LIDAR0 - 1

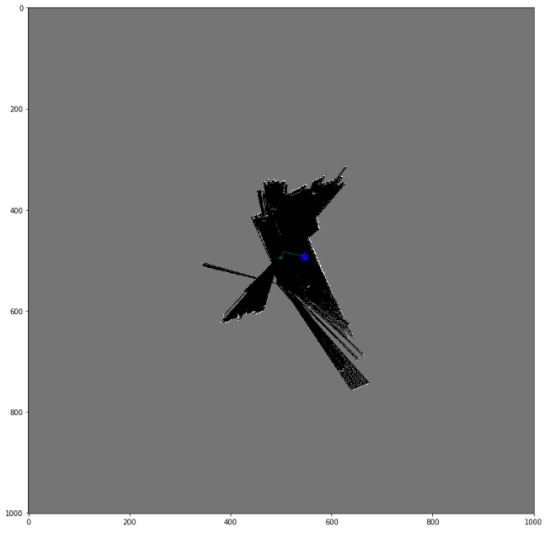

Fig. 6: LIDAR0 - 2

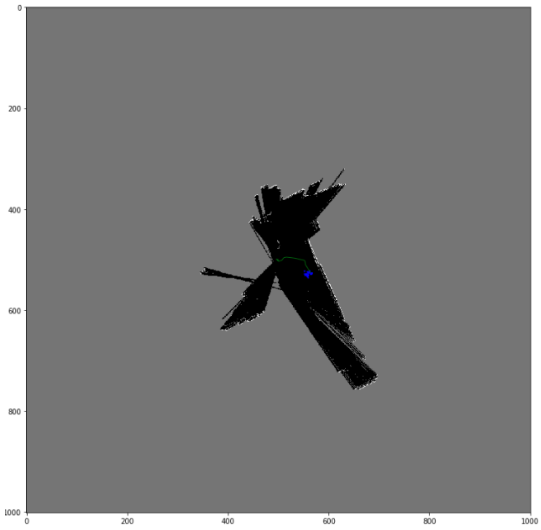

Fig. 7: LIDAR0 - 3

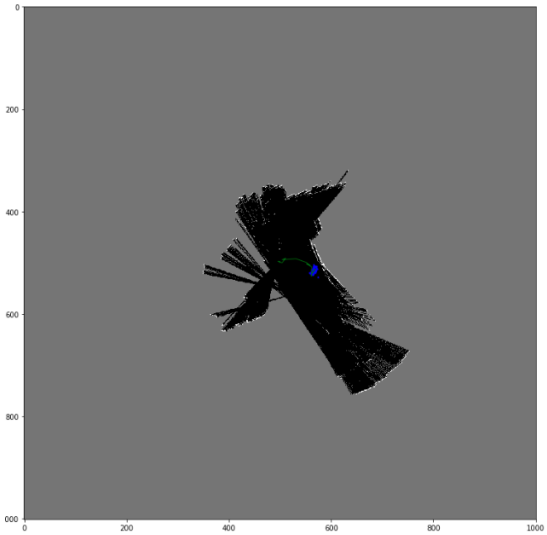

Fig. 8: LIDAR0 - 4

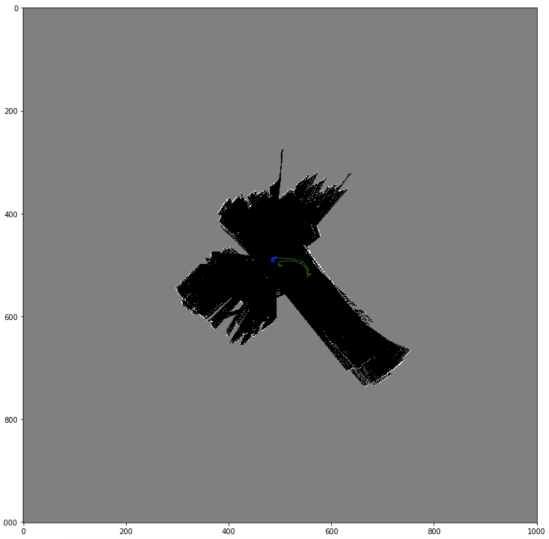

Fig. 9: LIDAR0 - 5

The plot above is the finished occupancy map for training set 0. Another example of this is attached below. Figure 10 below has less noise on some edges, but the hallway is not as well defined and straight as the one in Figure 9.

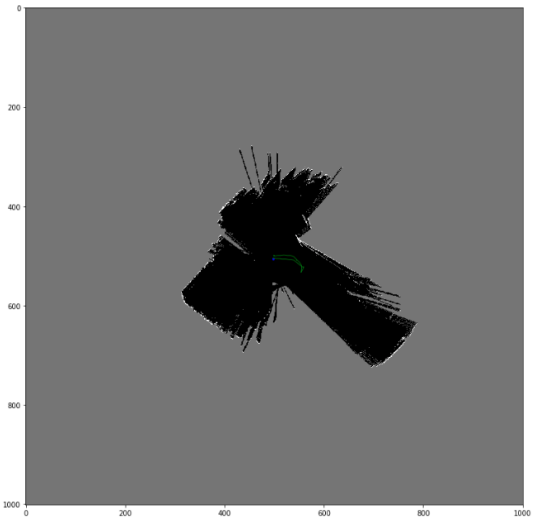

Fig. 10: Another good example for LIDAR0

B. Training Set 1

The dead reckoning plot for training set 1 is as follows:

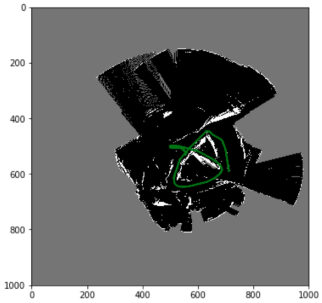

Fig. 11: Dead Reckoning for LIDAR1

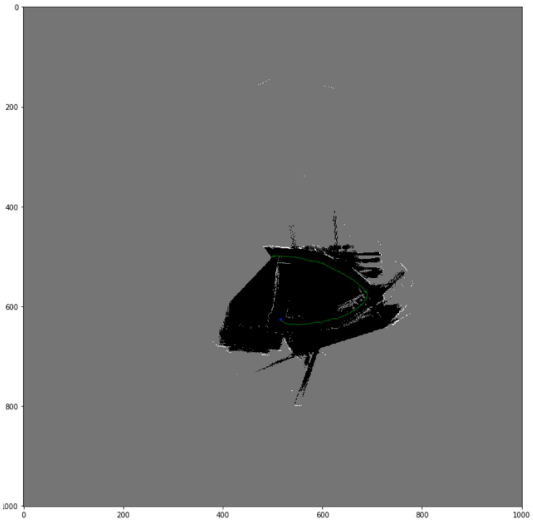

Fig. 13: LIDAR1 - 2

After observing the dead reckoning map, we noticed that the robot turned a lot. In order to avoid a lot of rotational noise, our yaw noise was very smaller than the x, y noise components. Even then, the SLAM algorithm does not work well in this case, as there is still a lot of noise. Additionally, walls inside the environment are not fully recognized.

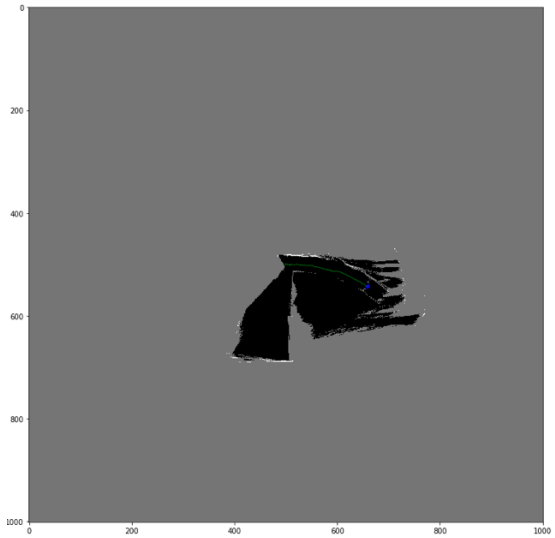

Fig. 12: LIDAR1 - 1

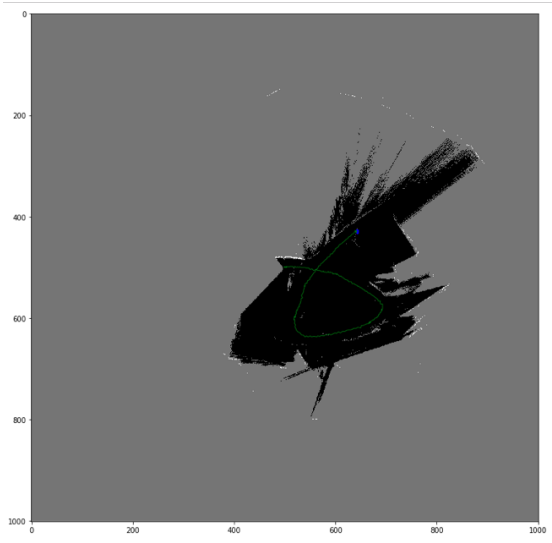

Fig. 14: LIDAR1 - 3

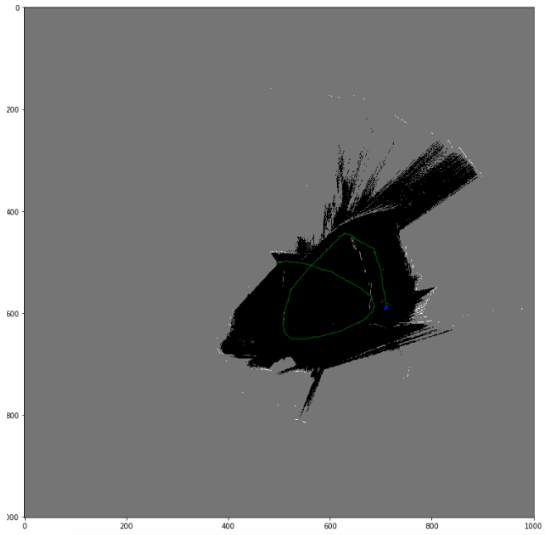

Fig. 15: LIDAR1 - 4

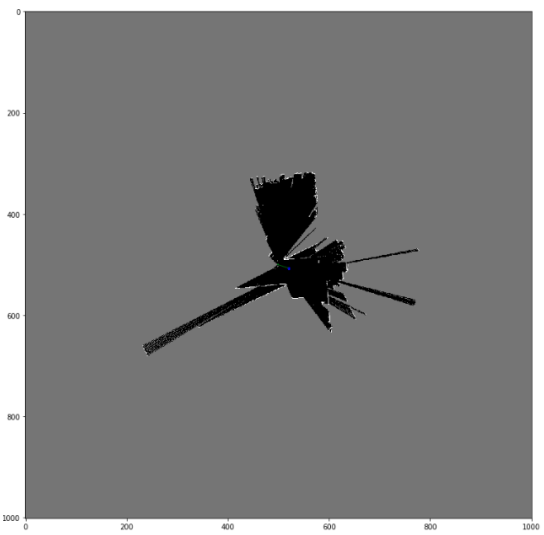

Fig. 17: LIDAR2 - 1

C. Training Set 2

The dead reckoning plot for training set 2 is as follows:

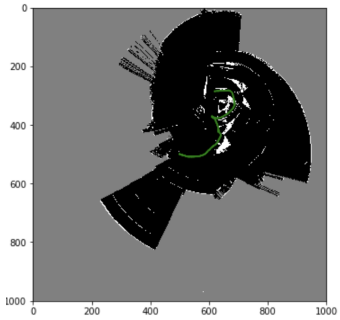

Fig. 16: Dead Reckoning for LIDAR2

Similar to the last training set, there is a lot of rotational noise present in this one. We tried reducing the amount of noise provided to the yaw component by a significant amount but without any real results. We also tried to add some variation to this yaw component - for every particle, there were 5 different noise values added to yaw (if N = 100 particles, this meant technically 500 particles to decide between). However, this attempt did not generate any different results. We also tried particle predicting and updating at a ratio: for every 5 predictions, only update once - the intuition was to give the particles time to spread out before converging to the most likely position. We also tried to vary the resampling threshold (we tried 10, 20, 30, and 40). However, SLAM performs poorly on this training set, as evidenced by the figures below.

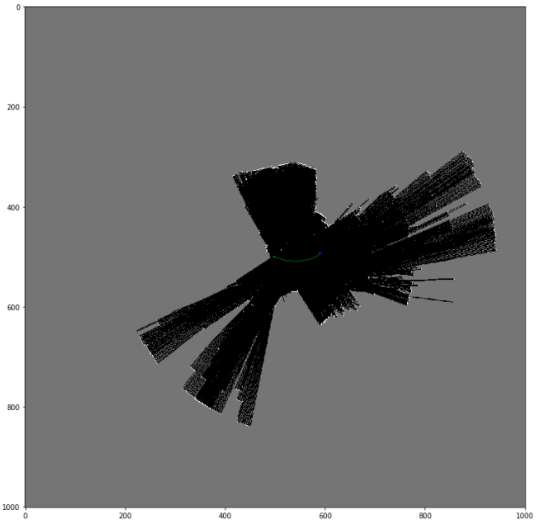

Fig. 18: LIDAR2 - 2

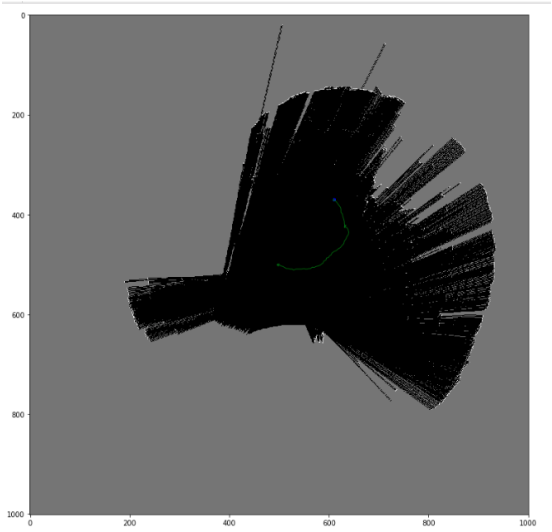

Fig. 19: LIDAR2 - 3

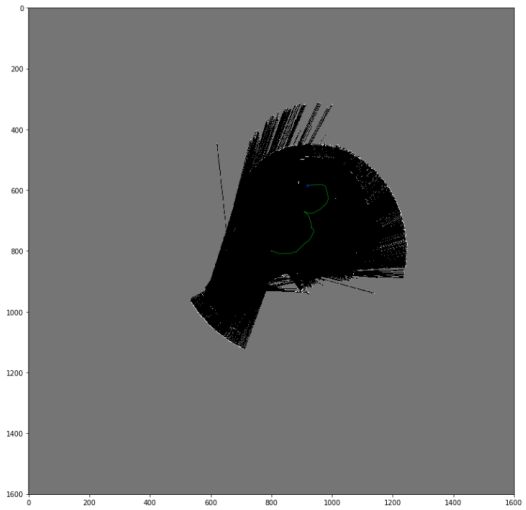

Fig. 20: LIDAR2 - 4

D. Training Set 3

The dead reckoning plot for training set 3 is as follows:

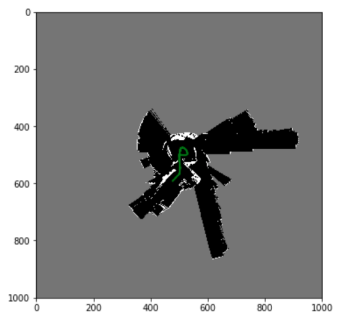

Fig. 21: Dead Reckoning for LIDAR3

SLAM performs somewhat decently at identifying one of the hallways in this image (the one on the left of the map), but fails to pick up the other one - you can see some edges on the right side of the map where the second hallway should be. In addition, the room the robot starts in is somewhat blurred due to rotational noise. We tried to remove noise as possible with hyperparameter tuning but SLAM does not recognize the full, clear map. The plotted trajectory is somewhat correct as this too is affected by the rotational noise.

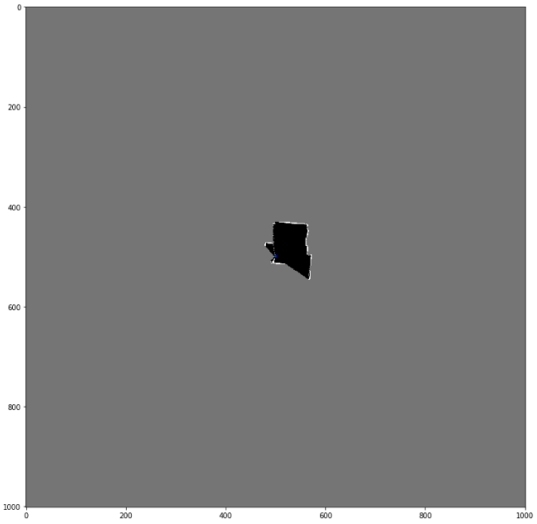

Fig. 22: LIDAR3 - 1

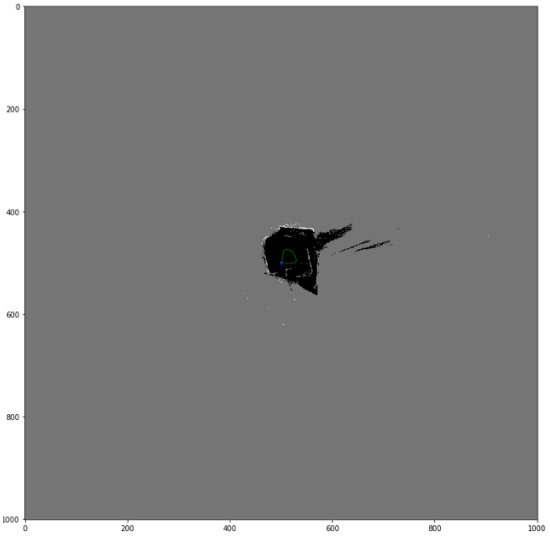

Fig. 23: LIDAR3 - 2

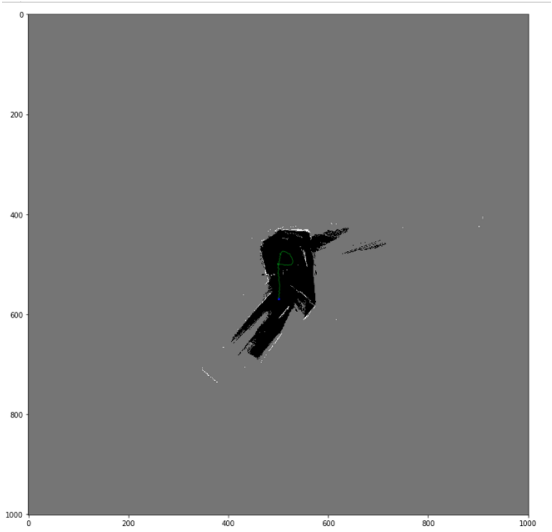

Fig. 24: LIDAR3 - 3

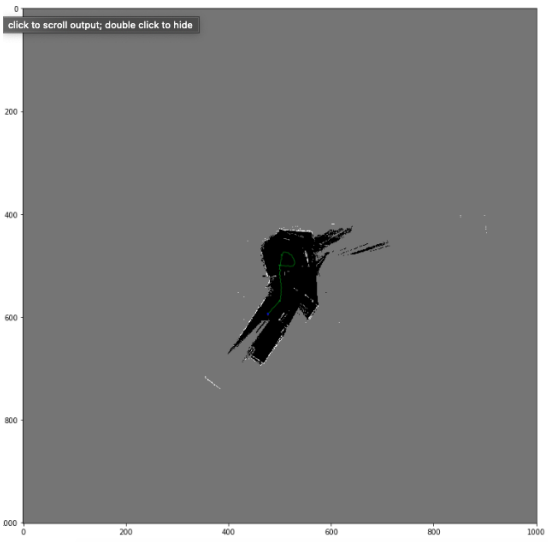

Fig. 25: LIDAR3 - 4

E. Training Set 4

The dead reckoning plot for training set 4 is as follows:

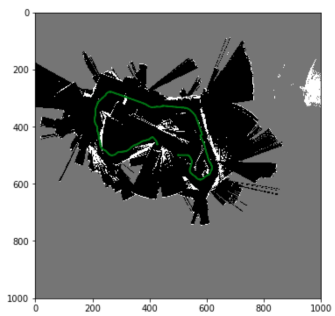

Fig. 26: Dead Reckoning for LIDAR4

The map generated by the baseline dead reckoning approach has a lot of rotational noise about the edges. This environment has the most turns by the robot and is thus the most sensitive to small changes in variation added to yaw. We discarded our approach to add variations in yaw for this environment, and only added a very small amount of noise (around 10e-7) to the yaw component. While the results for SLAM are far from perfect, much of the noise is removed around the edges compared to the original dead reckoning map. Additionally, the walls within the map are very vaguely defined upon closer inspection.

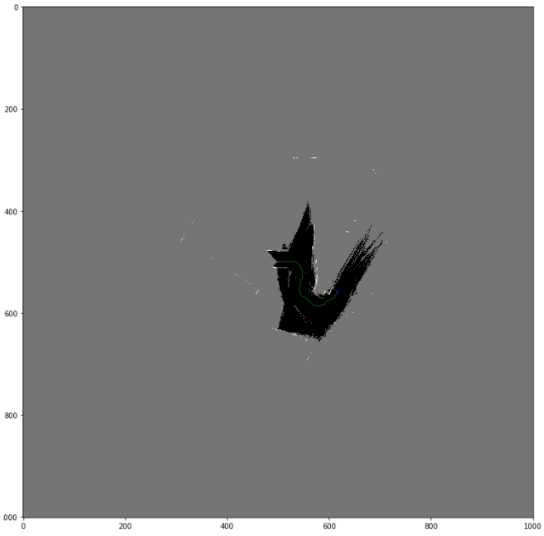

Fig. 27: LIDAR4 - 1

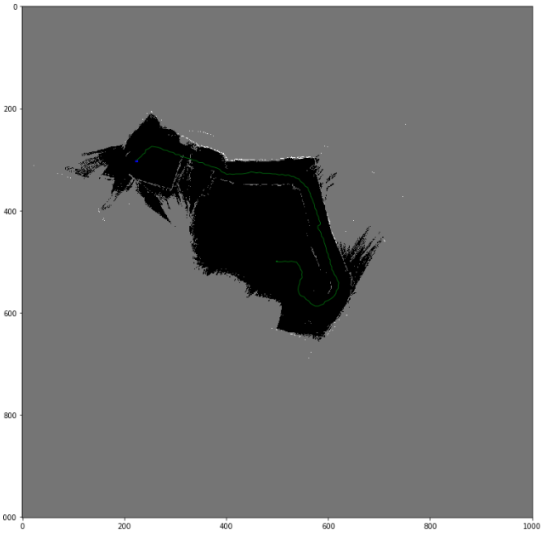

Fig. 28: LIDAR4 - 2

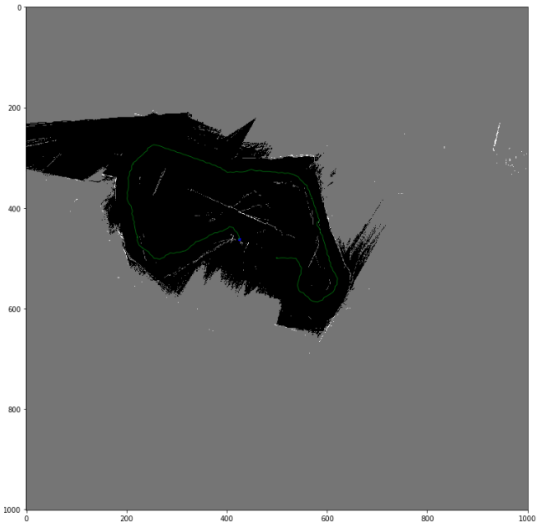

Fig. 29: LIDAR4 - 3

V. Conclusion

SLAM with particle filters for localization produces decent results when there is little rotational noise as evident by the first training set. Further work on this project will largely involve trying to make the particle filter algorithm more robust to this noise.

Acknowledgment

I would like to thank Professor Atanasov and the class TA's for their guidance through this project. I would also like to thank the OpenCV website for their useful open-source tools such as "polyline".