2D SLAM Using Particle Filter

Sanjay John

***Abstract*— The hallmark of understanding an environment through sensor data lies in SLAM - Simultaneous Localization and Mapping. In this project, we implement a 2D SLAM algorithm utilizing particle filters and various sensors in a moving robot to create a 2-dimensional map of a floor of a building observed from the top down. LiDAR and Kinect data allows for mapping of general structure and texture respectively, while IMU and encoder data allow for localization of the robot. We present results of this algorithm on three datasets, one of which is a test set released a few days prior to submission.**

I. INTRODUCTION

Understanding an environment through remote sensors has infinitely many applications, such as self-driving cars, unmanned machinery for mining, bomb-defusing, space exploration etc. The state-of-the-art in representing sensor data in a user-friendly manner lies in SLAM - Simultaneous Localization and Mapping. The gist of the algorithm is straightforward - given that you start at (0,0), start by mapping your environment. Then, predict your next position using your data, correlate this position with your current map, update your position based on this, and repeat.

In this project, we will use a 4-wheel robot equipped with a 2D Hokuyo LiDAR, encoders, IMU, and an RGB-D camera to map out the environment that the robot is exploring. The LiDAR data will be our observational input, the IMU and encoders will serve as our control inputs, and the RGB-D camera will provide superfluous texture mapping of the floor. It becomes necessary to model the control inputs, observations, and states as probabilities. To therefore bring in sensor data into this probability space and effectively compute probable trajectories and maps, it becomes necessary to implement a form of a Bayesian filter. In this project, we will use Particle Filters.

II. PROBLEM FORMULATION

SLAM ultimately becomes a chicken-and-egg problem, namely that:

- Mapping: given the robot state trajectory $x_{0:T}$, build a map m of the environment.
- Localization: given a map m of the environment, localize the robot and estimate its trajectory $x_{0:T}$.

This ends up becoming a parameter estimation problem for the parameters $x_{0:T}$ and m. Since we are given a dataset of observations $z_{0:T}$ and control inputs $u_{0:T}$, we can use MLE, MAP and Bayesian filtering to find the appropriate posterior likelihood.

SLAM exploits the Markov decomposition of the joint probability density function such that:

$$p(x_{0:T}, m, z_{0:T}, u_{0:T-1}) = p_{0|0}(x_0, m)\prod_{t=0}^{T} p_h(z_t|x_t, m)\prod_{t=1}^{T} p_f(x_t|x_{t-1}, u_{t-1}) \quad (1)$$

where p_h refers to the observation model, and p_f refers to the motion model. Therefore it becomes necessary to arrive at an evaluation of the observation model and motion model for every time step. It also becomes necessary to define an approach for propagating this density over time. We will do this using the Bayesian filter.

To implement the Bayesian filter, we use a set of particles with assigned weights. These can be represented as delta functions with weights to represent the probability densities $p_{t+1|t}$ and $p_{t|t}$. The Particle filter can therefore be derived from the Bayesian by substituting the delta functions. The prediction step becomes

$$p_{t+1|t}(x) = \sum_{k=1}^{N} \alpha_{t|t}^{(k)} p_f(x|\mu_{t|t}^{(k)}, u_t) = \sum_{k=1}^{N} \alpha_{t|t}^{(k)} \delta(x; \mu_{t+1|t}^{(k)}) \quad (2)$$

Since $p_{t+1|t}(x)$ is a mixture probability density function, we can approximate it with particles by drawing samples directly from it. For the update step we have

$$p_{t+1|t+1}(x) = \sum_{k=1}^{N} \frac{\alpha_{t+1|t}^{(k)} p_h(z_{t+1}|\mu_{t+1|t}^{(k)})}{\sum_{k=1}^{N} \alpha_{t+1|t}^{(k)} p_h(z_{t+1}|\mu_{t+1|t}^{(k)})} \delta(x; \mu_{t+1|t}^{(k)}) \quad (3)$$

Therefore, for the practical application of Particle filters for our SLAM process, we will need to compute

$$\mu_{t+1|t} = f(\mu_{t|t}, u_t + \varepsilon_t) \quad (4)$$

for the prediction step, where $\mu_{t+1|t}$ refers to the best particle's new state (best particle meaning highest weight), f refers to the motion model, u_t being the current motion input, ε_t being some Gaussian noise. We will also need to update our weights by computing the correlation of the particles with the current scan.

III. TECHNICAL APPROACH

A. Dataset

The data consists of 4 parts.

- LiDAR: LiDAR ranges from the Hokuyo at regularly spaced angles, as well as the minimum angle and maximum angle range. We will use this to build the occupancy grid.

- IMU: Accelerometer and Gyroscope data from an IMU attached to the robot. We will extract angular velocity from this, after filtering through a low-pass filter to reduce noise. We will only need to use the yaw signals as this is 2D from the top down.
- Encoder: Ticks from encoders attached to the wheels. Extracting displacement of each wheel requires finding meters per tic, which is the wheel diameter 0.254m multiplied by π (to get meter per revolution) divided by 360 tics per revolution. Therefore the wheel travels 0.0022 meters per tic. The displacement d_r of the right wheels of the robot would be calculated as $(FR+RR)*0.0022/2$
- Kinect: RGB-D camera data given as disparity images and corresponding RGB images. The disparity image will give the corresponding depth and therefore is used in the optical frame to world coordinate transformation.

B. Synchronization

Since each LiDAR scan, IMU reading, encoder reading and Kinect image were captured at differing times (as indicated by their timestamps) it became necessary to synchronize the data as much as possible. This was achieved by taking one to be the standard - we chose this to be the LiDAR as our mapping (and therefore correlation, thus update) depended on it. Running through each LiDAR scan, we found the absolute difference between the current LiDAR timestamp with the timestamps from everything else and found the index of the smallest, which would correspond to the closest timestamp for each.

C. Mapping

Mapping involves transforming the information from our depth sensors into world coordinates, then transforming it into map coordinates. Converting world to map coordinates required calculating

$$x_{map} = (x_w + \frac{m_h + m_w}{2})/m_{res} \tag{5}$$

where m_h, m_w referred to the map height and width (put at 60 meters) and m_{res} referred to the resolution of the map (0.05 meters per cell).

1) Occupancy Grid: We attempted to represent our map m as an occupancy grid. This is a map with $m_i = 1$ for occupied cells and $m_i = 0$ for unoccupied. We updated these cells through log-odds, which allowed us to simply add or subtract values to the cell per each time step. We used the LiDAR to update our log-odds map as each LiDAR hit corresponded to an addition of log-odds occupancy value, and thresholding was used to assign occupied cells. Our log odds value was taken as the log of the sigmoid of a certain belief (= 0.7) that our LiDAR hit corresponded to an occupied cell. LiDAR hits were first transformed to the world frame by using the current particle's state as well as the offset from the LiDAR sensor to the body. Then these world coordinates were converted to the corresponding map coordinates using (5). When plotting the occupancy grid, the grid values were thresholded such that high values would correspond to occupied while low would correspond to free.

2) Texture Mapping: Texture mapping can be thought of as an inverse operation from the pixel value of an image to the world frame, then bringing it to the map (using the same procedure as with the LiDAR). In our case, we are given disparity images and their corresponding RGB images. With this, we moved from the optical frame to world frame by using the depth derived from the disparity. To do this, we needed to take the inverse operation of the usual pinhole camera model procedure for finding corresponding pixels as shown in equation (6).

$$(X_w, Y_w, Z_w) = H_{cw}^{-1} R_{oc}^{-1} Z_0 K^{-1}(u, v) \tag{6}$$

where (u, v) referred to the pixels of the IR camera, K referred to the camera matrix, Z_0 referred to the optical axis and therefore the depth (calculated from disparity), R_{oc} referred to the transform that flips the axes from camera to the optical frame, and H_{cw} referred to the world to body to camera frame transform. Once these world coordinates were found, we thresholded the Z_w coordinate to find the ground plane, and used (5) with the X_w and Y_w coordinates to bring them back into the map, and swapped out the corresponding IR pixels with RGB pixels using a transformation provided.

D. Localization

Localizing the particle involves deriving the appropriate motion model (prediction), then using a correlation with the current map to assign appropriate weights to the particle (update).

1) Differential Drive Model: Since we have the displacements of each side of the car from the encoder and yaw rate ω_t from the IMU, we defined our motion model as the following

$$v_t = \frac{(d_r + d_l)/2}{\tau} \tag{7}$$

$$x_{t+1} = x_t + \tau v_t \sin(\frac{\omega_t}{2}) \frac{\cos(\theta_t + \frac{\omega_t}{2})}{\omega_t/2} \tag{8}$$

$$y_{t+1} = y_t + \tau v_t \sin(\frac{\omega_t}{2}) \frac{\sin(\theta_t + \frac{\omega_t}{2})}{\omega_t/2} \tag{9}$$

$$\theta_{t+1} = \theta_t + \tau \omega_t \tag{10}$$

However, it was found that the IMU data was not entirely accurate especially in large turns. Therefore yaw rate was also found from the encoder data by

$$\omega_t = \frac{(d_r/\tau - d_l/\tau)}{\text{width of robot}} \tag{11}$$

The yaw rate was then calculated as a weighted sum of the IMU (98%) and the encoder (2%) yaw rates as the encoder yaw rate was extremely noisy.

2) Particle Update: The update of particles relies on the current state of the particles, as well as their correlation with the map. We do this by summing number of grid cells in the map that are occupied from each particle's point of view, then taking the softmax of this value with respect to the other particles' to reach a probability. Based on these correlations, we update the weights. When iterating to the next mapping, we pick the best particle (highest weight). The more particles are used, the more accurate but more computationally intensive it becomes. 100 particles were chosen for this project.

3) Stratefied Resampling: On occasion, the particle update is not sufficient to arrive at the best estimate and therefore resampling is required. This is done by first checking if the sum of the particle's squared weights are above a certain threshold; if so, we used stratefied resampling to determine the particles to be resampled. The procedure is detailed in slide 28 of slide set 6. We found that resampling more often than not made the maps worse, and so a threshold was chosen that would only affect very large discrepancies.

IV. Results

A. Training Set

Figures 1 through 2 show the first dataset (identified as #20) without texture mapping, then with.

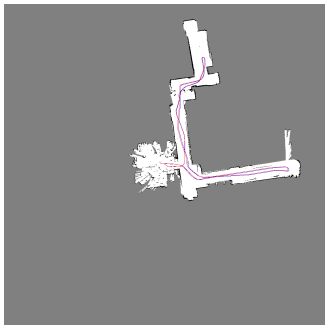

Fig. 1. SLAM Map of dataset #20, red shows trajectory, blue shows dead reckoning

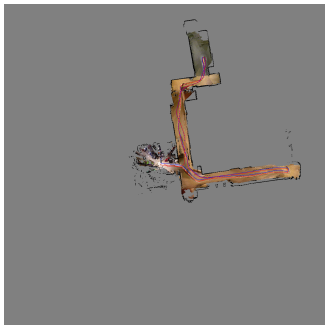

Fig. 2. SLAM Map of dataset #20, red - trajectory, blue - dead reckoning. Notice the floor changes from room to room

Figures 3 and 4 show the second dataset (identified as #21) without texture mapping, then with.

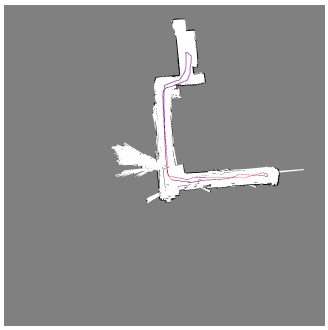

Fig. 3. SLAM Map of dataset #21, red shows trajectory, blue shows dead reckoning. Notice how this corresponds to the same map as #20, but starting from a different position

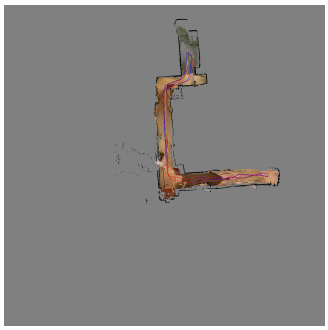

Fig. 4. SLAM Map of dataset #21, red - trajectory, blue - dead reckoning. Notice how the different starting position shows the different lighting affecting the texture mapping negatively.

B. Test Results

From Figure 5 onwards, we can see the progression of the test set's map. We can see that the southern hallways gets slightly wider at the end, possibly indicating an inability to correlate the particle position with the LiDAR readings.

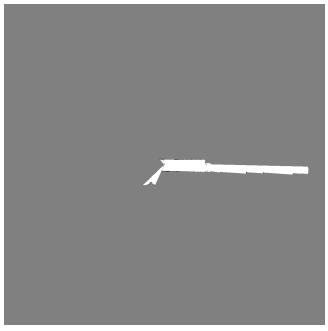

Fig. 5. Test set, 10% done, red - trajectory, blue - dead reckoning

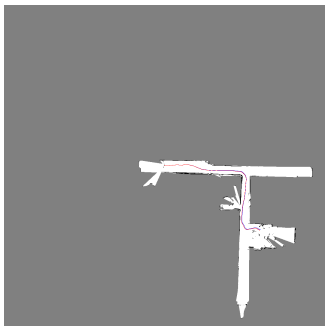

Fig. 6. Test set, 40% done, red - trajectory, blue - dead reckoning

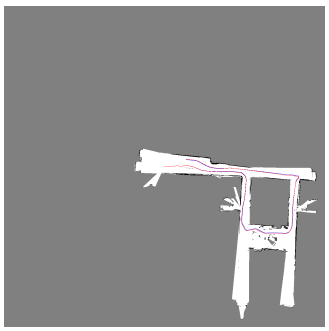

Fig. 7. Test set, 70% done, red - trajectory, blue - dead reckoning

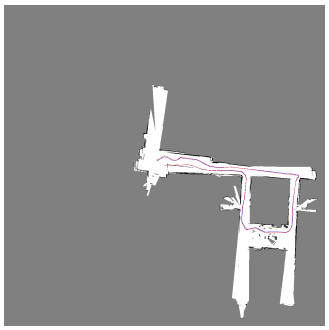

Fig. 8. Test set, 100% done, red - trajectory, blue - dead reckoning

V. CONCLUSIONS

While our SLAM algorithm performed well enough to provide a reinforced, consistent map across dataset 20 and 21, it should be noted that the test set encountered some distortion as evidenced by the widening hallways. This could be because of greater noise in encoder data which skews the yaw rate, or greater noise in the LiDAR scans. We can also see that dead reckoning is not much different than the best particle's trajectory, especially in the test set, which may reveal some flaws in our correlation method. Further progress should include modeling LiDAR noise, for example including a Gaussian p_{hit} for small measurement noise, an exponential p_{short} for unexpected objects, and uniform p_{rand} and p_{max} for unexplained noise and no objects being hit. Texture mapping is greatly dependent on the lighting conditions of the environment as seen from the robot's perspective, and therefore can benefit from some filtering on the RGB images to bring light to a consistent level.

ACKNOWLEDGMENT

Thanks to Professor Nikolay Atanasov and TAs Tianyu Wang and Ibrahim Akbar.

REFERENCES

[1] Barfoot, T. "State Estimation For Robotics." Cambridge, United Kingdom: Cambridge University Press, 2017