Simultaneous Localization and Mapping (SLAM)

1st Simon Fong
Electrical and Computer Engineering
University of California San Diego
La Jolla, United States
simonfong6@gmail.com

***Abstract*—In this paper, we discuss performing simultaneous localization and mapping or SLAM for short from sensor data read from the THOR robot.**

***Index Terms*—localization, mapping, SLAM**

I. Introduction

With the rise of autonomous vehicles from companies such as Waymo, Tesla, and Cruise Automation, the problem of being able to localize and map areas accurately is important so that these autonomous systems can know where they are and collect data about the environment to prevent such things as colliding with other vehicles. For our problem, we will be doing simultaneous localization and mapping using LIDAR and odometry data collected from the THOR robot traversing a University of Pennsylvania hallway.

II. Problem Formulation

In SLAM, we are after two main things: localization and mapping. Localization is our estimate of where the robot is. Mapping is knowing whether a region is space is unoccupied, occupied, or unknown.

A. Localization

In localization we are precisely interested in the pose of the robot, x_t, at a give time, t. Here the pose has two components, the position $p \in \mathbb{R}^3$, and the rotation $R \in \mathbb{R}^{3\times3}$ For our, problem formulation, we limit our position to two dimensions, keeping z component of p fixed to $z = 0.93m$ and only moving about the x and y coordinates. Additionally, the robot can only rotate its body about the z axis, meaning we only need to keep track of θ_t representing the yaw of the robot. To get this estimate, we will be combining the map m and observations $z_{0:t}$ to help us get an estimate of the pose the robot. In essence, we want to find the pose $\hat{x}_t = argmax_x P(x_t|m, z_{0:t})$ that maximizes this probability given the map and observations.

B. Mapping

In mapping, the values we are concerned about are whether a given pixel m_i in the map is unoccupied, occupied, or unknown. In my formulation, we will represent unoccupied with -1, occupied with 1, and unknown with 0. For each cell at position i, we want to know the probability that it is occupied or $P(m_i = 1|x_{0:t}, z_{0:t})$, where $x_{0:t}$ is the cumulation of all the estimated poses of the robot up to time t and $z_{0:t}$ is the cumulation of all the sensor observations up to time t. In our case, x_t is estimated using odometry readings and observations and z_t is the LIDAR scan at time t. We want to do this for all m_i with the true map M so that we have the $\hat{m} = argmin_m \Sigma_i Loss(m_i, M_i)$

III. Technical Approach

We have two approaches: dead reckoning and particle fitlering.

A. Dead Reckoning

In dead reckoning, we do not use the map to adjust the pose of the robot, we simply trust the odometry readings fully so that we absolutely know the pose of the robot at all times relative the origin (0,0). Now our robot pose x_t becomes an absolutely known value instead of a probability distribution. We calculate the pose by starting at $p = [0, 0, BodyHeight]$ and $R = I$ where I is the identity matrix in $\mathbb{R}^{3\times3}$ representing no rotation and facing in the positive x-axis and summing the change in pose given my our odometry readings: $\Sigma_{i=0}^{t} \Delta Pose$.

With the robot pose known, now all we have to do is incoporate our LIDAR scans z_t to create a map m. We can do this by initializing an empty occupancy grid with log-odds of 0 everywhere. Here the more negative the value of a cell m_i means the higher probability that it is unoccupied, the more positive the value of a cell means the higher the probability that it is occupied, when it is 0, it represents having equal probability that the cell is unoccupied and occupied meaning unknown.

Now with a given LIDAR scan, z_t, we can find which cells it thinks are unoccupied using the Bresenham's line rasterization algorithm implemented in $cv2.line$. We know the LIDAR thinks the end of the ray is occupied. So for each unoccupied cell, we decrease the cell's value by our defined log-odds, in our case $log(4)$ (meaning $\frac{80\% TruePositive}{20\% FalsePositive}$) and inversely, when the reading says the cell is occupied, we increase the cell's value by the log-odds.

However, since these cells are in the LIDAR's frame of reference, we will need to do a transformation of these cell coordinates to the world frame so that we can update the correct cells.

Finally, when all the LIDAR readings have been applied to occupancy grid, we can set a threshold of the value of each cell to figure out the estimated occupancy state of a cell m_i. For our purposes, we set the $UnoccupancyThreshold = -4*$

$logodds$, and the $OccupancyThreshold = 4 * logodds$, meaning all cells with values less than $UnoccupancyThreshold$ will be counted as unoccupied, all cells with values greater than $OccupancyThreshold$ will be counted as occupied, and all cells with values in between counted as unknown.

B. Particle Filtering

Using Dead Reckoning to localize is great when we have high trust in the odometry readings, but in our case we can see that odometry readings are not fully accurate when we take a look at the trajectory in Fig. 1. So, instead of thinking that we know our robot pose x_t with certainty, we use Particle Filtering to estimate the pose based using the map m and LIDAR readings z_t.

To do this, we will use N number of particles to represent our guesses of the pose of the robot x_t^i, the ith particle at time t. All particles are initialized with a uniform weight w which will be $w = \frac{1}{N}$. At the very beginning all particles will be initialized with location at $(x, y) = (0, 0)$ and orientation $\theta_0 = 0$.

Now, to incorporate the change in pose $\Delta Pose$, LIDAR readings z_t, and map m, we will follow the following procedure:

```
for lidar_reading in all_lidar_readings:
    for particle in all_particles:
        # Update the pose with noise.
        particle.pose += delta_pose + GAUSSIAN_NOISE

        lidar_filled = particle(lidar_reading)

        # Pose's lidar scan correlation with map.
        correlation = 0
        for lidar_i, map_i in lidar_filled, map:
            if lidar_i == map_i:
                correlation += 1
        particle.correlation = correlation

    # Update particle weights with correlation values.
    cors = [all_particle_correlations]
    soft_cors = softmax(cors)

    weights = [all_particle_weights]
    new_weights = weights * soft_cors
    new_weights = normalize(new_weights)

    assign_weights(particles, new_weights)

    particle = get_highest_weight(particles)

    lidar_scan= particle(lidar_reading)

    map.update(lidar_scan)

    if num_effective_particles < N_EFF:
        resample(particles)
```

IV. Results

A. Dead Reckoning Mapping

In Figures 1-5, we have the final localization and mapping results from dead reckoning peformed on the training datasets 0 thru 4 respectively. For each map, grey represents unknown or unexplored cells, white represents unoccupied cells, and red represents occupied cells. Additonally, blue represents the pose of the robot within the map.

The dead reckoning results were okay. The best result was that of Fig 1. on Train 0. It estimated the hallway on the left, the nook on the right, and the room behind it relatively okay. It is still far from perfect as you can see the hallway is not a rectangular shape but a more cone shaped. The room behind it was mapped relatively well, catching the square shape. The results from Fig 2. and Fig 3., Train 1-2 are hard to judge because there were no RGB images to compare to, but they seem relatively noisy and non rectangular, so it appears not so great. Fig 4-5 seem pretty off from the videos, but you can kinda make out the path they took and the maps resemble the room shapes vaguely, but overall very noisy.

These were performed with thresholds at -/+ 4 * log-odds, so maybe adjusting these numbers could help us only plot the cells with higher likelihood by increasing these thresholds.

Additionally, we have already thrown away scans, below $z = 0.3meters$, $distance > 30meters$, $distance < 0.1meters$ because these detect obstacles not relevant to the map, and are not valid LIDAR readings. However, furthing tweaking of the LIDAR filtering could help a lot with adjusting the scan.

One noticeable thing is that the LIDAR will scan through windows giving faulty readings occasionally, but increasing the occupancy thresholds should help with not keeping these in the final maps.

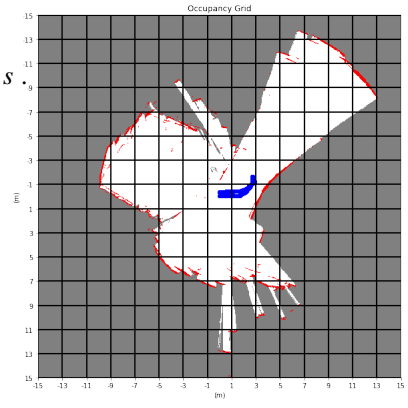

Fig. 1. Dead Reckoning: Train 0

B. Particle Filtering Mapping

In Figures 6-10, we have the final localization and mapping results from particle filtering peformed on the training datasets 0 thru 4 respectively. Here the colors represent the same

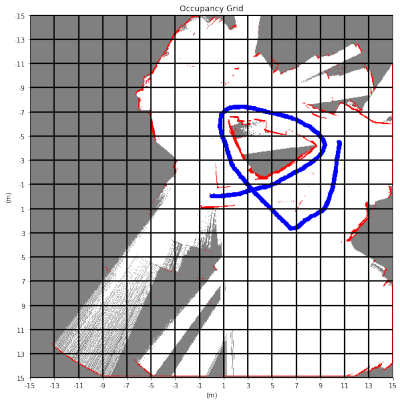

Fig. 2. Dead Reckoning: Train 1

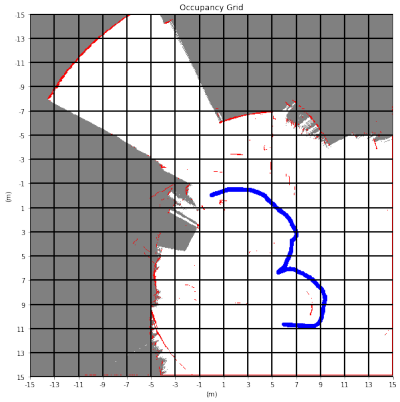

Fig. 3. Dead Reckoning: Train 2

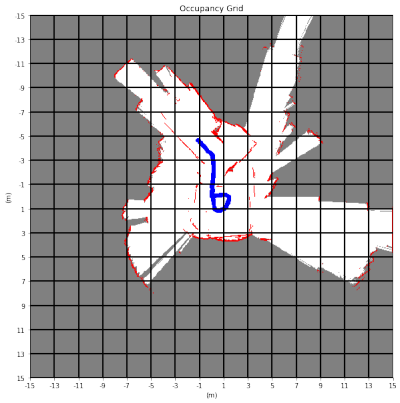

Fig. 4. Dead Reckoning: Train 3

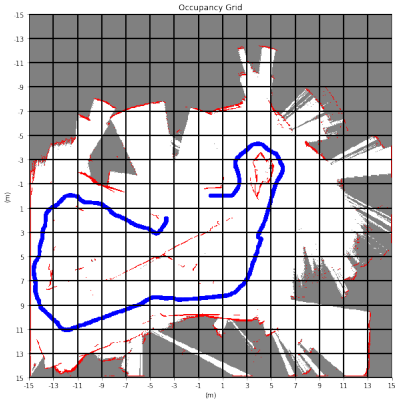

Fig. 5. Dead Reckoning: Train 4

elements as in dead reckoning, except magenta represents the pose of the robot within the map.

For all Figures 6-10 they appear to have much improved upon the dead reckoning maps in that the map shapes seem more rectangular which is closer to approximating the shapes of rooms and hallways. However, they are still far from perfect. Take for example Fig. 6 of Train 0. The hallway to the left was better approximated to a rectangle, but there is an entire second hallway slightly off angle from the hallway. This is not great.

For these results, we used 100 particles, and down sampled all the LIDAR by 100 readings. We used 100 particles to better guess where the pose is, but we needed to down sample by 100 readings so that we can run the particle filtering on the training sets in a reasonable amount of time. Ideally, we increase the particle count up to 1000 and perform no down sampling, but in the limited time that I allocated, this was not possible.

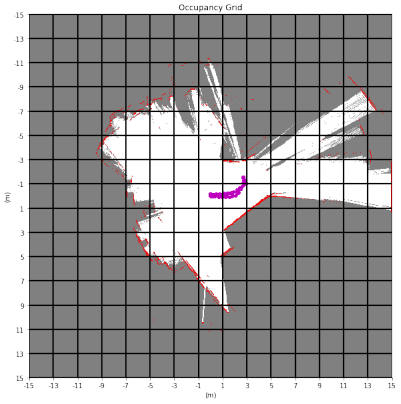

Fig. 6. Particle Filtering: Train 0

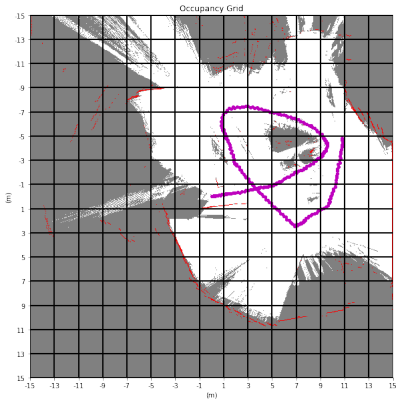

Fig. 7. Particle Filtering: Train 1

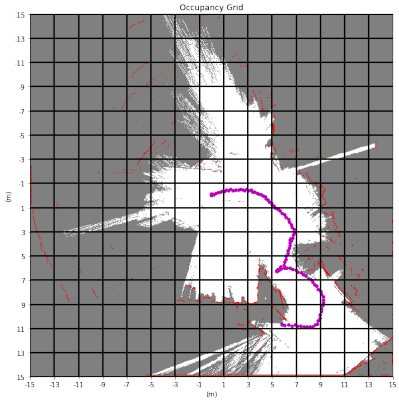

Fig. 8. Particle Filtering: Train 2

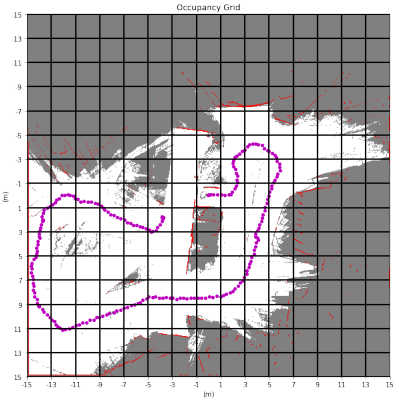

Fig. 10. Particle Filtering: Train 4

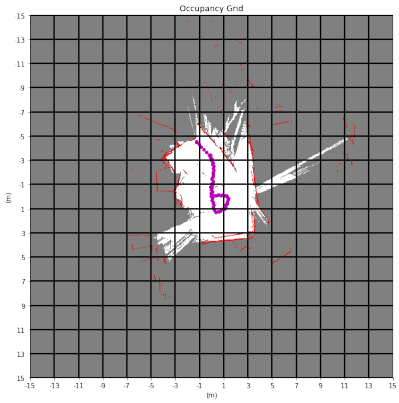

Fig. 9. Particle Filtering: Train 3