Simultaneous Localization and Mapping (SLAM)

Abhijeet Singh

***Abstract*— This report describes the process of estimating the surrounding map of a robot while simultaneously ascertaining the position of the robot in the same map. The map used in this project is an occupancy grid which tells whether a particular location in the map is filled or occupied. The robot localization is done with the help of particle filters that keep track of the robot's (x, y) position and the direction it is pointing, with respect to a fixed global frame. We use LIDAR, IMU and odometry data to perform SLAM.**

I. INTRODUCTION

Simultaneous Localization and Mapping (SLAM) is a process of creating and updating a map of the robot's initially unknown environment and locating the robot itself in this map. This is a core problem in robotics and there are several methods to solve it - particle filters, Kalman Filters, graph slam etc. Here we demonstrate the Particle Filter based SLAM and use it to draw the occupancy grid map of the robot's environment while keeping track of its pose - the (x, y) position and its yaw angle representing the direction it is pointing to, with reference to a fixed global frame.

The experiments to collect the data were performed with a humanoid robot (fig 1) with sensor data from LIDAR, IMU and global odometry measurements. With LIDAR, we get a good estimate of the distance of objects from the sensor. IMU gives us accelerometer and angular velocity measurements. The global odometry data enables us to estimate the relative position of the robot from its previous position. We combine this input sensor data and give a map of the environment and robot location as the output.

II. PROBLEM FORMULATION

Given the 2D laser range scanner (LIDAR) measurements, IMU body measurements and the global odometry data, implement the structure of mapping and localization in an indoor environment. Identify the places which are free and those that are occupied and simultaneously keep track of the robot trajectory in the map.

III. TECHNICAL APPROACH

The robot's pose (x, y, theta) is tracked by particles, which are basically samples from the pose distribution. Each such particle (sample) has a weight associated with it, which tells the likelihood of this particle at that time instant.

The particle filter based SLAM algorithm works in the following steps. These steps are repeated at every time step of our data.

1) Find the particle with the highest weight.

*This work is part of the Project 4 of the ESE 650 class Learning in Robotics at University of Pennsylvania, taught by Prof. Daniel Lee

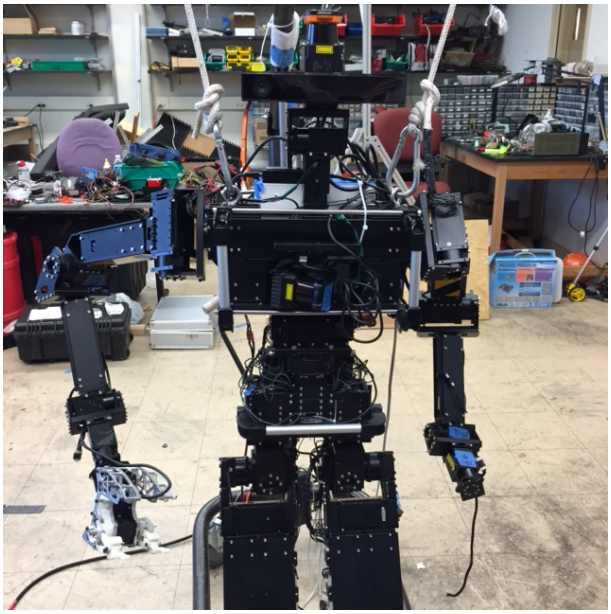

Fig. 1. Robot

2) Perform Mapping with the above particle
3) Do localization prediction to predict the next pose of the particle
4) Do localization update to update the weights of all the particles
5) Resample the particles, if required

Each of the above step is described in more detail in the subsequent sections.

A. Particle with highest weight

We choose the maximum likelihood particle to update our map. This corresponds to the particle having the maximum weight in our set up. There could be more sophisticated ways of updating the weight - for example each particle could keep its separate map, or choose more than one particle to update a map etc, but we keep it simple here.

B. Mapping

Mapping is done using the LIDAR measurements. The basic approach is to find the distance of the obstacles, from the current robot position, and use it to update the map. The map is an occupancy grid matrix of cells, where each element denotes whether the corresponding cell is free or not. Instead of a binary number, it will be useful to keep the probability of a cell being occupied. Hence, we use a log-odds ratio denoting the occupancy of the cell given whether the LIDAR ray returned from that cell or not. If the ray returns from the

cell (which we can get from the distance information from LIDAR data), then there is higher probability that the cell will be occupied, and vice-versa. Bayes rule gives us -

$$\log \frac{P(\text{Cell occupied}|z)}{P(\text{Cell free}|z)} = \log \frac{P(\text{Cell occupied})}{P(\text{Cell free})} + \log \frac{P(z|\text{Cell occupied})}{P(z|\text{Cell free})}$$

The term on the left hand side of the equation is our posterior belief of the map, for each cell. The first term on the right hand side denotes our prior belief, while the second term denotes the accuracy of the LIDAR. This term increases or decreases our current belief of the cell - if a ray passes through this cell, we decrease its log-odds by this term, if a ray returns from this cell, we increase the log-odds ratio of the cell by this term. The amount of increase and decrease is a constant and depends on the particular LIDAR and how much we trust its measurement.

The LIDAR is attached to the head of the robot, so the distances from this frame need to be transformed to the global frame. We can use three frame of references to describe the details - first is the head frame, where the LIDAR is kept. The x-y plane of this frame consists of all the LIDAR rays, around the perpendicular z-axis. This head frame is kept at an angle to the rest of the body. Rotation about these angles are further translation to the foot of the robot, gives us the body frame of reference with its origin at the feet of the robot (on the ground). This body frame is further at some orientation with respect to the global ground frame (say the point where the robot started, with the x-y plane matching with the ground), and the robot's IMU measurement can give us angles to transform from this body frame to the ground frame. After all these transformations, we get the global coordinates of the points that are reflecting the LIDAR rays. We remove the rays that are hitting the ground.

The continuous global frame can be discretized by dividing it into cells at some resolution. For each cell we maintain a log-odds ratio as described previously and we update the ratio whenever a ray passes or hits that cell. The high values of this ratio denote that this cell is occupied, and low values denote that this cell is free. This, in short, is our map.

C. Localization prediction

This is the step where we predict the next pose of the particle, based on the odometry data. We do this for all the particles.

Simply put, we update the pose of the particle by adding the displacement of the particle from its current position, in all the three dimensions - x-position, y-position, yaw angle (representing the direction the robot is pointing). So this is basically a dead-reckoning problem. Since the odometry measurements are in a global frame and we cannot directly trust the data, we convert it into the local frame of the particle, measure the displacement in this local frame and then add this to the previous position. This last step also involves rotation of the local frame into global frame. These transformations can be described by the following equations

$$p_{t+1} = p_t \oplus (o_{t+1} - \ominus o_t)$$

where,

$$x_{t+1} \oplus x_t = \begin{bmatrix} p_t + R(\theta_t)p_{t+1} \\ \theta_t + \theta_{t+1} \end{bmatrix}$$

$$x_{t+1} \ominus x_t = \begin{bmatrix} R(\theta_t)(p_{t+1} - p_t) \\ \theta_{t+1} - \theta_t \end{bmatrix}$$

We also add some random noise to the pose of these particles to denote the uncertainty of the measurements. Different particles have different noises added using the following equation (w represents a Gaussian noise vector)

$$p_{t+1} = p_t \oplus ((o_{t+1} - \ominus o_t) \oplus w)$$

D. Localization update

The previous step of prediction spreads the particles due to the noise added to them. This step selects the good particles by assigning a larger weight to the particle that better represents the current pose. This better representation is found out by observing the correlation between the pose of the particle (which tells how the map looks to this particle) and the actual map we have been tracking till now (last updated in the mapping step above). The larger the correlation, the larger the weight and vice-versa.

Map correlation can be found by examining the similarities in the global map being observed by this particle and the current map found out by the Mapping step. This similarity can simply be the number of cells that are free in both the maps and occupied in both the maps. Higher the number, more similar are the two maps. This correlation value can be used to update the current weights of the particle by simply adding the normalized correlation value to this particles current weight.

This step helps in finding the good particles - those that have been tracking the map adequately. If we have N particles keeping track of the current pose, their pose will be spread out after the previous step of localization prediction. This is primarily to keep track of the errors in measurement. This step then performs the opposite and folds the particles to a few good ones.

E. Resampling

The previous step of localization update re-weights the particles. If there are too many particles with low weight, chances are that they do not represent the current pose of the particle (and neither the uncertainty). Also, the few particles with good weight might become limited in numbers to track the subsequent uncertainty in measurements. Hence, whenever the number of particles with low weight increases too much (given by $N_{effective}$), we resample the particles according to their current weight. The particle with higher weight is more likely to be chosen in this resampling

step. Several methods can be used for sampling - Sample importance resampling, rejection sampling, low variance resampling etc.

$$N_{effective} = \frac{(\sum_{i=1}^{N} w_i)^2}{\sum_{i=1}^{N} w_i^2}$$

IV. Discussion

A. Mapping

While mapping (and also while localization update) we need to use the LIDAR scan data. This data has to be pruned by ignoring the laser points that are too near (less than 0.1m) or too far (greater than 30m). Further, we need to remove the ground points from the data, otherwise they will be treated as obstacles, when indeed those points comprise the free space where the robot can move. Fortunately this becomes very easy after the LIDAR scan points are transformed to the global frame - since the x-y plane of the global frame matches with the ground, we just remove the points that have value close to 0 in their z-coordinate.

The map represented physical space from -22m to +22m in both x and y directions (global frame). It was divided into cells at a resolution of 0.05m. Each cell was initialized with a log-odds value of 0, denoting equal probability of being free or occupied. To find out the free cells from the position of occupied cells, we used Bresenham line algorithms. Once we find the free cells and the occupied cells, we update the log odds map. If a LIDAR scan hits a cell, we increase its log-odds by 0.08, if it passes through (free cell) we decrease it by -0.008. Further, we bound the log-odds to a minimum of -5 and maximum of 5, so that we can incorporate some changes in the environment.

B. Localization prediction

In this step we had to add noise to account for the uncertainty in the odometry measurements. This noise plays a crucial role in determining how much we trust the measurements. We ended with a Gaussian noise model with a diagonal covariance matrix $[0.001, 0.001, 0.005]$. If we increase these noise covariance values, the results we got were quite blurry. In fact, in dataset 1, with increased noise values, there was a point at one time when probably incorrect particle was being chosen due to the higher noise value which resulted in a wrong trajectory. So we used small values.

C. Localization update

In this step we update the weight of the particles based on map correlation values. Since the correlation value is taken for many cells, this might become too large to exponentiate, and hence we operated in log-space here.

We also performed a simple optimization in calculating the map correlation. This was done in favour of time to quickly find a correlation value. Instead of finding the actual number of cells that were hit and not hit and compare them with the original map, we simply summed the log-odds value of the cells which were hit according to the current pose of the robot. Since the correlation values are finally normalized, this led to a huge increase in speed of the algorithm.

V. Results

The results on the train data and test data are shown in the figures 2, 3, 4, 5, 6. The yellow-white pixels/line represent the walls and various obstacles. The dark pixels/areas represent the free space. The blue line represents the actual trajectory followed by the particle estimated from the particle filter.

We see that these results are quite good - there are very few double lines and not so much blurring. The obstacle lines (walls primarily) are also quite sharp and clear.

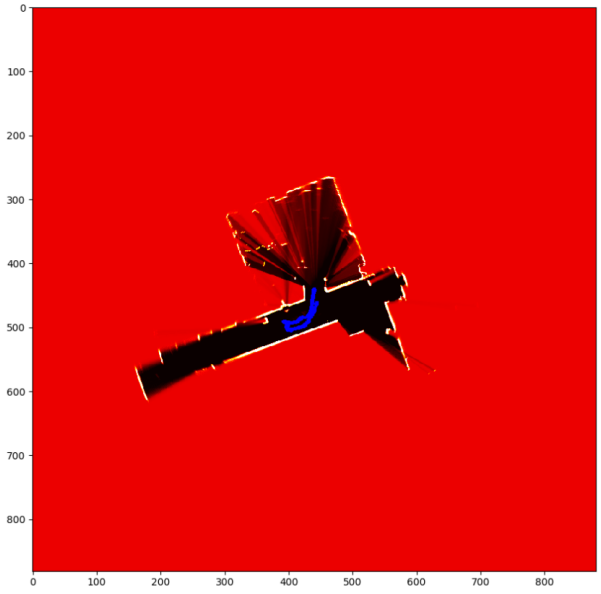

Fig. 2. Train dataset 0

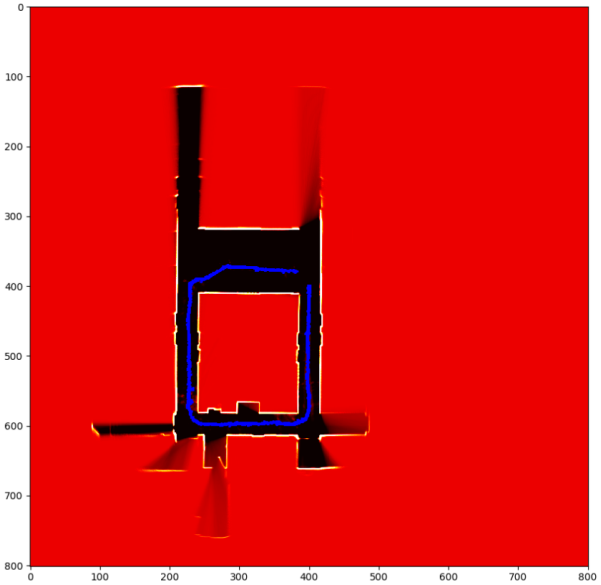

Fig. 3. Train dataset 1

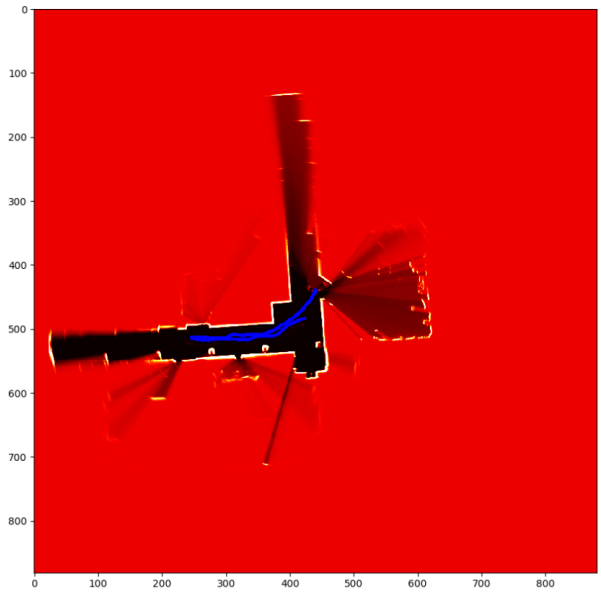

Fig. 4. Train dataset 2

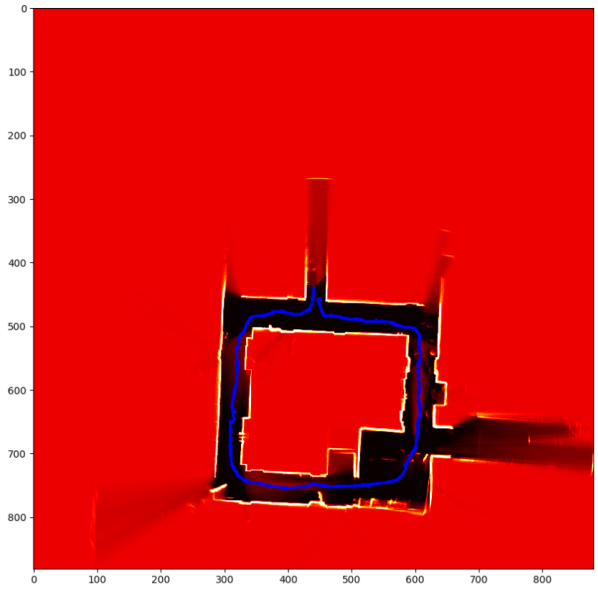

Fig. 6. Test dataset

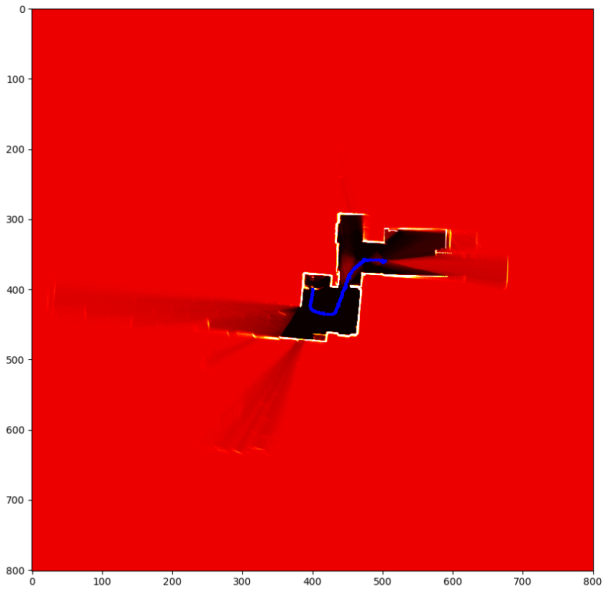

Fig. 5. Train dataset 3