Project 2: Particle Filter SLAM

Jaidev Shriram (University of California, San Diego)

I. Introduction

The simultaneous localisation and mapping of a moving robot is a classic problem in robotics, one that to date, lacks a perfect solution. The fundamental difficulty with the problem is how coupled the two tasks, localisation and mapping, are, which is then made more difficult by the fact that most sensors contain noise. As a result, trivially tracking the location of a robot and building a map will not suffice, as sensor noise can compound significantly and lead to drift. In this project, we tackle this problem by formulating SLAM probabilistically and repeatedly update our estimated location and map based on new information that is received.

While there are several formulations formulations for probability based SLAM, we will focus on particle-SLAM in the context of occupancy grid mapping. In this scenario, we build a 2D map of the world, indicating whether a grid in the real world is occupied or free. While this map is being built, we maintain a collection of pose estimates, also known as particles, which represents candidate positions for the robot. Using sensor observations, the likelihood of these particles being the candidate pose can be measured. At a high level, performing these iterative predictions of the pose using a robot's motion model, and updating the pose based on sensor observations is called Particle Filter SLAM.

Our results show that this filtering of the pose using LiDAR based observations is able to effectively estimate the pose of the robot, while building a high-quality occupancy map of the scene. For ease of visualising, we also map the RGB images captured by the robot to the occupancy grid.

II. Problem Formulation

In our scenario, we must localise a differentiable drive robot equipped with a Kinect camera and Lidar using its encoder and range measurements. Formally, we are tasked with estimating the state $\mathbf{x}_{0:t}$ of the robot till a time instant t, based on the corresponding control inputs $\mathbf{u}_{0:t-1}$ and observations $\mathbf{z}_{0:t}$ while building a map $\mathbf{m}$.

A. Occupancy Grid Mapping

The first part of the Particle filter SLAM is occupancy grid mapping. An occupancy grid is a 2D bird's eye view representation of the world, where the map $\mathbf{m}$ is initially unknown and needs to be estimated given the robot trajectory $\mathbf{x}_{0:t}$ and a sequence of observations $\mathbf{z}_{0:t}$. Each cell in this 2D grid corresponds to a fixed resolution or scale with respect to the world. However, the map is only an estimate and hence, we model the probability of the map's state instead:

$$p(\mathbf{m}|\mathbf{z}_{0:t}, \mathbf{x}_{0:t}) = \prod_{i=1}^{n} p(m_i|\mathbf{z}_{0:t}, \mathbf{u}_{0:t}), \tag{II.1}$$

where m_i is the ith cell of the map, which we assume is independent of other cells when conditioned on $\mathbf{z}$ and $\mathbf{u}$. We model m_i as a Bernoulli random variable, where each cell is either 1 (occupied) or 0 (free), with probability $\gamma_{i,t}$ and $1 - \gamma_{i,t}$ respectively. We then define the log odds ratio for the ith grid at time t as $\lambda_{i,t}$, where the log-odds ratio is estimated as:

$$\lambda_{i,t} = \lambda_{i,t-1} + (\Delta\lambda_{i,t} - \lambda_{i,0}). \tag{II.2}$$

These log odds can then be converted to a binary map by obtaining the probability mass function (representing probability of being occupied) as $\gamma_{i,t} = \frac{\exp(\lambda_{i,t})}{1+\exp(\lambda_{i,t})}$. Here, the log-odds represent the log-odds of the bernoulli random variable m_i, such that:

$$\lambda_{i,t} = \log o(m_i|\mathbf{z}_{0:t}, \mathbf{x}_{0:t}), \tag{II.3}$$

where the odds ratio for occupancy grid mapping is defined as:

$$o(m_i|\mathbf{z}_{0:t}, \mathbf{x}_{0:t}) = g_h(\mathbf{z}_t|m_i, \mathbf{x}_t) o(m_i|\mathbf{z}_{0:t-1}, \mathbf{x}_{0:t-1}), \tag{II.4}$$

Here, $g_h(\mathbf{z}_t|m_i, \mathbf{x}_t)$ is the observation model odds ratio, defined as the product of the inverse observation model odds ratio and map prior odds ratio.

$$g_h(\mathbf{z}_t|m_i, \mathbf{x}_t) = \frac{p(m_i = 1|\mathbf{z}_t, \mathbf{x}_t)}{p(m_i = -1|\mathbf{z}_t, \mathbf{x}_t)} \frac{p(m_i = -1)}{p(m_i = 1)} \tag{II.5}$$

B. Localisation

Localisation is the problem of estimating the trajectory of the robot $\mathbf{x}_{0:t}$ given a map $\mathbf{m}$, control inputs $\mathbf{u}_{0:t-1}$, and a sequence of measurements $\mathbf{z}_{0:t}$. Here, we make use of a particle filter to keep track of the probability density function $p(\mathbf{x}_t|\mathbf{z}_{0:t}, \mathbf{u}_{0:t-1}, m)$ of the robot over time. The particle filter maintains N hypothesis of the state $\mu_{t|t}$ (representing the state at t given information upto time t) with an associated confidence $\alpha_{t|t}$ per hypothesis. The pose of the robot is then considered to be the $\mu[k]$ where $\alpha[k] = \max\{\alpha[0] \dots \alpha[N]\}$.

The state of these particles is updated in two steps:

1. Prediction Step: In this step, for every particle $\mu_{t|t}[k]$, we compute:

$$\mu_{t|t}[k] = f(\mu_{t|t}[k], \mathbf{u}_t + \epsilon), \tag{II.6}$$

where $f(\mathbf{x}, \mathbf{u})$ is the differentiable drive motion model; $\mathbf{u}_t$ is comprised of the linear and angular velocity input, and ϵ is random Gaussian noise with variance σ.

2. Update Step: Here, we update the confidence estimate for the particle hypothesis using the LiDAR observation model $p_h(\mathbf{z}|\mathbf{x}, \mathbf{m})$, which measures the likelihood of obtaining the current sensor reading from a state $\mathbf{x}$ on the current map $\mathbf{m}$. The weights are then updated as follows:

$$\alpha_{t+1|t+1}[k] \propto p_h(\mathbf{z}_{t+1}|\mu_{t+1|t}[k], \mathbf{m})\alpha_{t+1|t}[k] \tag{II.7}$$

The LiDAR observation model is calculated as the number of cells in the map $\mathbf{m}$ that align with the sensor reading $\mathbf{z}_{t+1}$ when converted to the world frame.

C. Texture Mapping

Once the pose is estimates, we aim to colour the occupancy map with the RGB values of the floor captured by the Kinect camera onboard. The camera captures RGB and depth information, which we can reproject to 3D and map back to the occupancy grid.

III. Approach

A. Robot Configuration

In this project, we work with a differentiable drive robot equipped with a Kinect RGBD camera, 2D LiDAR, and IMU. Each sensor operates at a different frequency, and hence exact mappings between sensor readings are not available. As a result, we use the nearest available sensor reading for every operation. Hence, u_t will correspond to the control estimated from the nearest available input to t from the wheel encoder and IMU.

Wheel encoder: The wheel encoder provides the number of rotations of the four wheels at 40Hz. The wheel has a diameter of 0.254m, which corresponds to 0.0022 meters per tic. Given the readings for each wheel, the left and right velocity are calculated as $FL + RL/2 * 0.0022$ and $FR + RR/2 * 0.0022$ respectively, where F/R correspond to front/rear and L/R correspond to left/right.

IMU: The IMU measures the linear and angular velocity of the data. Since the robot is rotating around the Z axis, we only consider the yaw rate and use this as the angular velocity. We also do a low-pass fiilter of this data to smooth out the data, with a bandwith of 10Hz.

LiDAR: The LiDAR sensor outputs a vector of length 1076 per time step, representing the furthest point hit by the laser at 1076 angles, in a range from $-135°$ to $135°$, or $270°$ total. The LiDAR has a maximum range of 30 meters.

Kinect Camera: An RGBD camera is placed on top of the differentiable drive robot, at $(0.18, 0.005, 0.36)$ m with respect to the robot. The camera faced down with a pitch of 0.36 radians and yaw of 0.021 radians. The camera is precalibrated and the intrinsics are provided with the specifications sheet. Due to an offset between the RGB and depth camera, the two are not perfectly aligned, and a transformation must be applied to align the two, as given in the specification sheet.

B. Motion Model

We represent the state $\mathbf{x}$ of the robot as $(x, y, \theta) \in \mathbb{R}^3$, where (x, y) represent the position and θ represents the rotation/heading of the robot. We consider the robot's origin at the center of the robot, and hence, the linear velocity v_t is $\frac{v_l+v_r}{2}$, where v_l and v_r are the average left and right wheel velocities. Let ω_t be the angular velocity around the Z axis, derived from the IMU sensor. Then, the motion model for the differentiable robot states that:

$$\mathbf{x}_{t+1} = \begin{bmatrix} x_t \\ y_t \\ \theta_t \end{bmatrix} + \begin{bmatrix} v_t * \cos(\theta_t) \\ v_t * \sin(\theta_t) \\ w_t \end{bmatrix} * \tau, \tag{III.1}$$

where τ is the time difference between successive measurements. Due to noise in the sensors, this motion model produces a noisy trajectory, but still proves useful for the predict step of the filtering process. The motion model is verified by plotting the dead-reckoning trajectory, as shown in figure 6.

C. Mapping

Updating the map: We assume that the Lidar is accurate 80% of the time or $\Delta\lambda_{i,t} = \log 4$, when the cell is occupied. We assume that there is a uniform prior on the map - each cell is equally likely to be occupied or empty, making $\lambda_{i,0} = 0$. Hence, we update each grid as:

$$\lambda_{i,t+1} = \lambda_{i,t} + / - \log 4 \tag{III.2}$$

We assume that the robot begins with the identity pose. To avoid overconfident estimates, we limit the values of $\lambda_{i,t}$ to be within $(-50, 50)$.

Converting LiDAR Readings to Map: First, we convert the LiDAR reading L to 3D coordinates as $z = [L\cos(\theta), L\sin(\theta), 0]$, where L is the vector of range measurements and θ is the $270°$ range of the LiDAR split into 1076 bins. The LiDAR scan is then transformed to the world frame using the current frame's pose estimate.

These points only give us the occupied cells of the map, as the LiDAR does not detect free space explicitly. Hence, we mark the free cells of the map using a 2D line drawing tool (Bresnham) which marks all points between the robot's position and occupied cells detected as free. In both these cases, we follow the update rule defined earlier.

D. Particle Filter

We maintain a list of 100 particles for the robot $\mu_i \in \mathbb{R}^3$ representing (x, y, θ). We initialise all particles to zero at the start. We set the weight of each particle to be equal initially.

1) Prediction Step: In the prediction step, we apply the differentiable drive kinematics to the particles with the addition of some noise. Formally,

$$\mu_{t+1} = f(\mu_t, \mathbf{u}_t + \epsilon), \tag{III.3}$$

where $\epsilon \in \mathbb{R}^2$ is added to the linear and angular velocity and is drawn from a 2D gaussian $N(0, \gamma)$. The weights of each particle are untouched in this step.

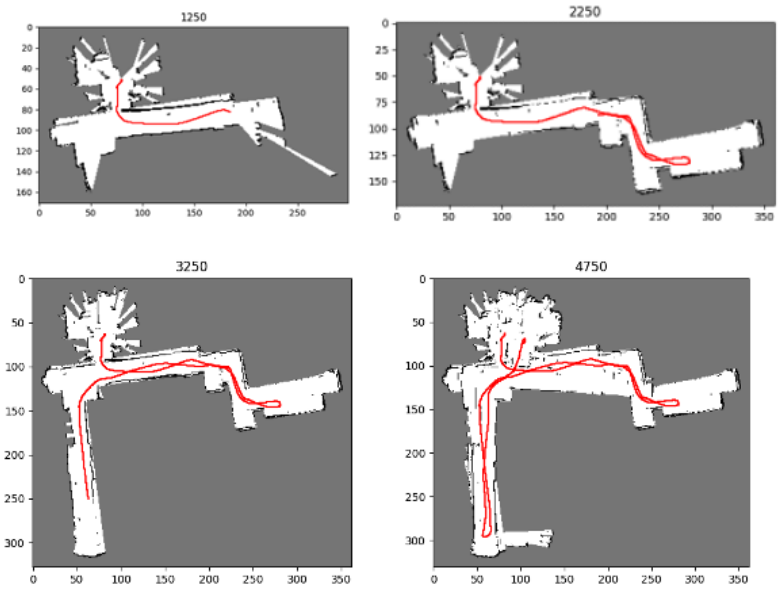

Fig. 1: The Trajectory and Occupancy Grid Map Over Time - Sequence 20

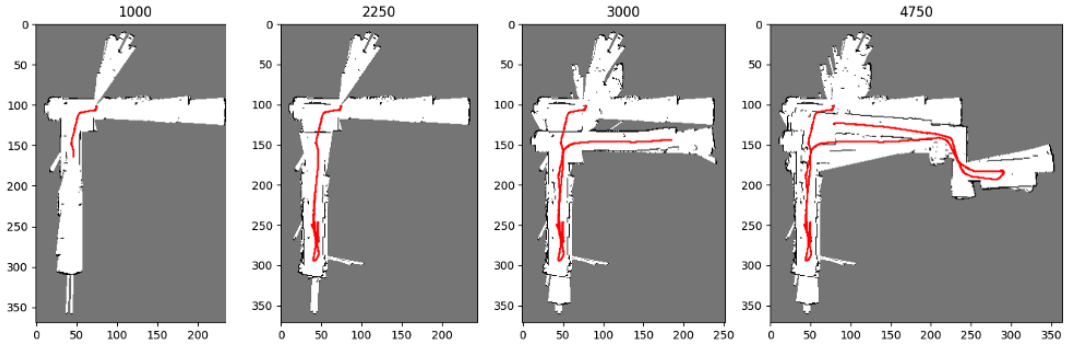

Fig. 2: The Trajectory and Occupancy Grid Map Over Time - Sequence 21

2) Update Step: In the update step, we update the particle weights, not the poses, based on the observation model $p_h(\mathbf{z}_{t+1}|\mu_{t+1|t}[i])$ for the ith particle. Specifically, we scale each particle weight with the correlation between the LiDAR reading and the world map, before normalising it to ensure that $\sum \alpha_i = 1$. The correlation function computes the number of cells in the map that agree with the transformed LiDAR scan. In practice, we transform the LiDAR scan from every particle, to obtain a correlation value for each.

We also go a step further and perform a local search in the neighbourhood of each particle to find the optical correlation values. We do this by perturbing each particle by some $\delta x, \delta y$ and $\delta\theta$ and take the cartesian product of these, to obtain all possible combinations. In practice, we perturb the positions by 0.1m and θ by 0.1 radians to form a $5\times5\times5$ grid of values around the current particle's position. The particle's state is then updated to include the perturbation which yielded the maximum correlation value for that particle.

3) Resampling: In the process of updating particle weights, we experience particle depletion, where particle weights quickly go to zero. To ensure diversity in the pose hypothesis, we resample the particles, where the probability of resampling is based on the particle weights. This ensure's that we have more guesses in the vicinity of the most likely pose. The number of effective particles is calculated as:

$$N_{eff} = \frac{1}{\sum_{i=1}^{N} \alpha_i}. \tag{III.4}$$

We set a lower bound on N_{eff} to be $N/2$ and resample when N_{eff} drops below this.

E. Texture Mapping

In texture mapping, we first attempt to create a 3D point cloud from the Kinect data. This point cloud can then be orthographically projected to the map. First, we align the RGB image and the depth map using the transformation provided to us. After this is aligned, we convert the pixel

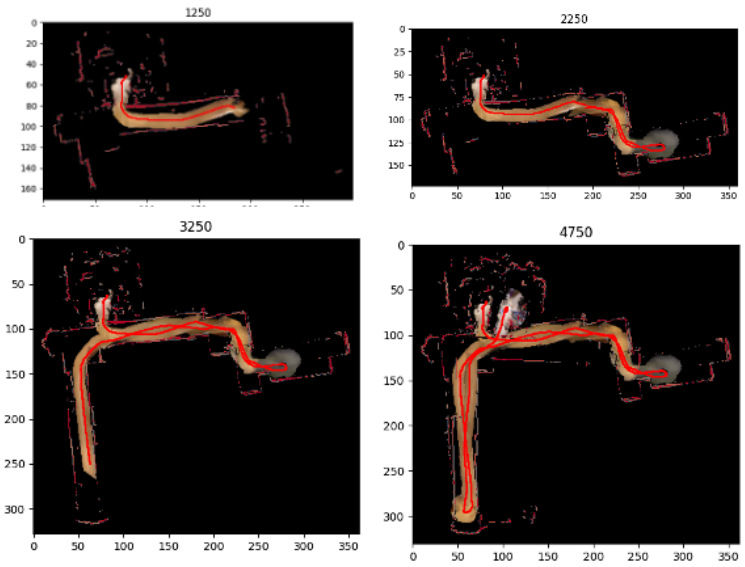

Fig. 3: The Textured Map Over Time - Sequence 20

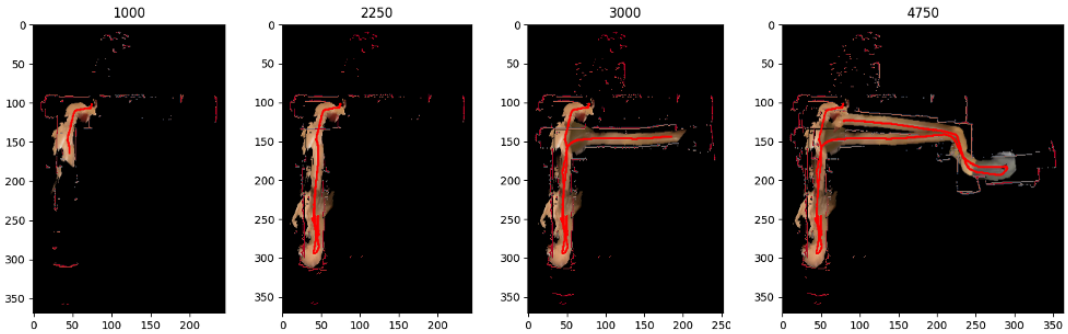

Fig. 4: The Textured Map Over Time - Sequence 21

coordinates to 3D coordinates by multiplying it with the inverse of the intrinsic matrix, after which we scale it by its corresponding depth.

$$x = (u - c_x)/f_x * z \tag{III.5}$$

$$y = (v - c_y)/f_y * z, \tag{III.6}$$

where u, v are image coordinates and c_x, c_y are the coordinates of the principal point, and f_x, f_y are the focal length of the camera. These points are now in the optical frame. Before transforming them to the world frame, we represent it with respect to the body frame. The transformation from this aligned optical frame to world frame is:

$$\hat{X_W} = \begin{bmatrix} 0 & 0 & -1 \\ -1 & 0 & 0 \\ 0 & -1 & 0 \end{bmatrix} X_O \tag{III.7}$$

The Kinect camera is not level with the floor, hence we reverse this rotation, computed using the pitch and yaw of the kinect sensor, as:

$$X_W = R^{-1}_{pitch,yaw} \hat{X_W} \tag{III.8}$$

We subsequently filter the points to only include those close to the floor ($z < 0.1$). To ensure that the map produced is smooth and free of sharp irregularities, we also do colour averaging per pixel.

IV. Results

In this section, we present some results for SLAM from a differentiable drive robot. The figures 1, 3 represent the trajectory, occupancy map, and texture map for sequence 20 at various stages. The figures 2, 4 show the same information for sequence 21.

V. Discussion

Particle Filtering Helps Avoid Drift. This is especially clear in sequence 21, where the dead-reckoning trajectory (fig. 5), produces multiple copies of the corridor. However, at a similar timestep, in the particle filter SLAM map (fig. 2), the corridor is very compact and well estimated - this is

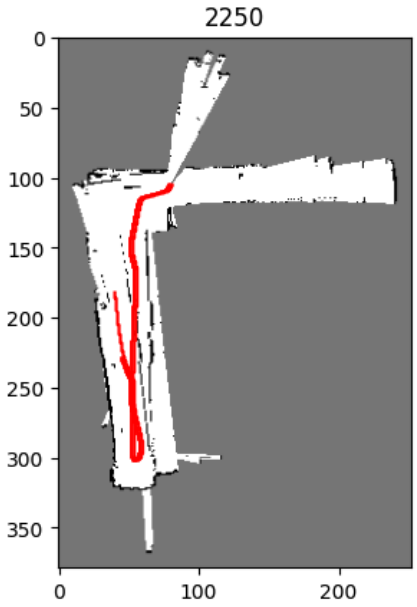

Fig. 5: Dead-reckoning trajectory for Sequence 21 - intermediate result

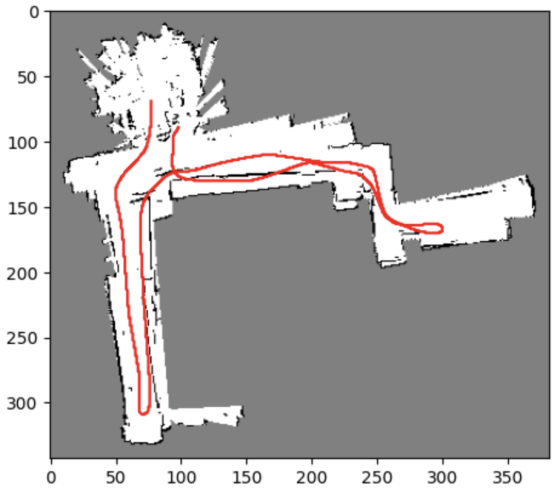

Fig. 6: Dead-reckoning trajectory for Sequence 20

likely due to the observation model. While sequence 21 does diverge later, the benefits of the technique are still visible.

High Sensitivity to θ. Changing the variance of the noise added to the motion model yielded drastically differing results, with θ having the most impact. It should be noted when θ is perturbed, points far away, move along a larger radius. As a result, it is expected that changing the yaw would have a large impact on the observation model. This can be seen in figure 7, where the map looks very incorrect in earlier stages as well. While the final maps produced from particle filtering in this project aren't perfect, the intermediate are impressive and capture the shape of the indoor scene very well.

IMU Filtering Inconsequential In my experiments, I found

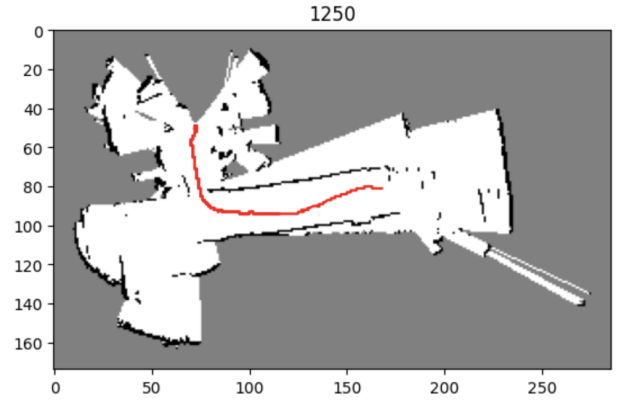

Fig. 7: Poor calibration of perturbation in yaw can lead to a bad map at initial stages itself.

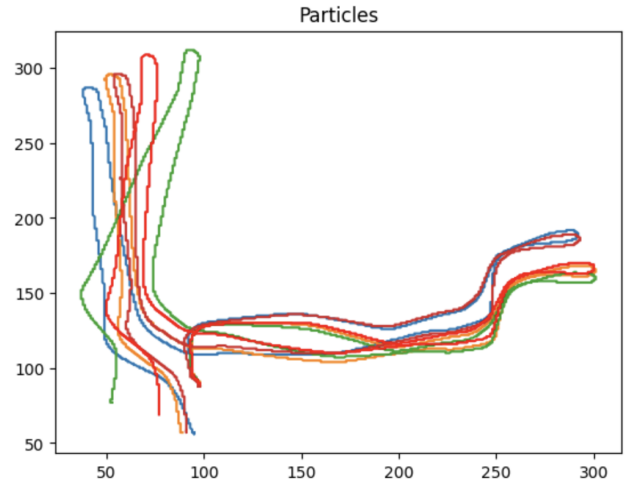

Fig. 8: The trajectory of multiple particles

that doing a low-pass filter using the IMU did not produce any drastic changes in the dead-reckoning trajectory.

Colour Agnostic Correlation Computation Can Fail In the case of sequence 21, we can see the robot make a right turn before the actual corridor and create an incorrect map. This is likely because different walls can be aligned with a parallel offset. This is a common problem in point cloud registration and using visual features can potentially improve results.

Particle Trajectories Diverge Due to Noise In figure 8, we can see how the addition of noise causes the particles to follow different trajectories. The map is not displayed here to make the visualisation clearer.