ECE276A HW3 Report
Samuel Sunarjo
A12340230

Introduction

Simultaneous Localization And Mapping (SLAM) is a problem of constructing a map of an unknown environment while keeping track of the agent's location and pose simultaneously. SLAM is a very important problem to address because environment perception and mapping is crucial for a robot to operate autonomously and make correct decisions or actions. Furthermore, location and mapping are the essence and prerequisite of motion planning. There are many real life applications that use SLAM or similar approach, such as self-driving cars and unmanned vehicles/robots in the air, ground, under water, and so on. SLAM is certainly one of the important aspects in autonomous robots. My **approach** to the SLAM problem in this assignment is by using a particle filter on the lidar sensor data, which consists of lidar scans and pose of the sensor, to construct the 2-D map and estimate robot pose simultaneously.

Texture mapping is the problem of constructing a map of the environment using images taken by a RGBD camera. Texture mapping adds more information, such as color of the floor, on the map constructed through SLAM. This is practical in real life applications because more information about the environment will certainly be useful for solving certain problems or understanding the environment better. In this assignment particularly, images from the RGBD camera are projected onto the 2-D map from SLAM to color the floor. My **approach** to texture mapping is to project image points that belong to the ground on a copy of the SLAM 2-D map using the estimated pose while running the SLAM algorithm.

Problem Formulation

Simultaneous Localization And Mapping (SLAM)

Given dataset of an inertial measurement unit (IMU), Lidar, and Kinect sensor on the robot, develop and run the SLAM algorithm. The position (in meter) of the center of mass of the robot and the positions (in centimeters) of the sensors relative to the robot are also given. The angles of the head and neck, $\phi_{head}\ and\ \phi_{neck}$ of the robot are also given. IMU measurements, roll pitch yaw angles of the robot joints are given, along with the timestamp for each measurement. The Lidar sensor provides the absolute odometry $o \in R^3$, where $o = (x, y, yaw)$, scan resolution in radians, IMU roll pitch yaw angles, scanned distances $l \in \{R^1\}^N$, where N is the number of lidar scans, range from $-135^\circ\ to\ 135^\circ$, along with the timestamp for each measurement. The documentation for the robot THOR contains some of the necessary parameter values mentioned above about the robot.

Texture mapping
Given a set of camera images with RGBD data, along with timestamp each image is sampled at, the goal for this part of the project is to construct a texture map of the floor for the constructed 2-D map in the world frame based on the estimated pose of the robot from the SLAM algorithm. The images are taken in the robot frame. The images are formatted as $M = \{P_v, P_h, C\}^N$, where $P_v \in R^1$ is the pixel position on the vertical axis and $P_h \in R^1$ is the horizontal axis, and $C \in R^4$ corresponds to the RGBD channels of the pixel, and N is the number of samples. The depth channel (in millimeters) is provided from the IR camera and the RGB channels are provided from the RGB camera.

Technical Approach

Simultaneous Localization And Mapping (SLAM)

- First, the states s of all the particles are initialized to $s = (0, 0, 0)$, which will be used as the origin (or center) of the map that is being constructed.
- The first lidar scan is used to initialize the occupancy map and the log-odd map, with the state $s = (0, 0, 0)$.
- The **log-odd map** contains the log of the probability λ that a particular location m_i on the occupancy map is occupied (oppose to being free). The log-odd map will be updated in each iteration of the SLAM particle filter algorithm. The log-odd $\lambda(m_i)$ is updated by accumulating the value $log(g)$ if the location is identified as occupied, and the value $log(\frac{1}{g})$ if the location is identified as free. The state of each location m_i in the **occupancy map** is determined by the current log-odd map via a probability transformation and thresholding. The log-odds will be transformed to probabilities $P(m_i = 1)$ which indicates the likelihood of the location being occupied and compared to thresholds to set the state of the map to "occupied" (indicated with value of 1), "free" (indicated with the value of -1), and "unknown" (indicated with the value of 0).
    - $P(m_i = 1) = 1 - \frac{1}{1+exp(\lambda(m_i))}$
    - $m_i = 1$, if $P(m_i = 1) > 0.6$
    - $m_i = -1$, if $P(m_i = 1) < 0.4$
    - $m_i = 0$, if $0.4 < P(m_i = 1) < 0.6$
- The lidar scan distances l are first filtered to only contain valid ranges from 0.1 to 30 meters. Then they need to be transformed to cartesian coordinates x, y, z in the lidar frame using basic trigonometry sine and cosine function, as each lidar scan corresponds to a specific angle in the range of $-135° \; to \; 135°$:
    - $x = l * cos(\theta_{lidar\ scan})$
    - $y = l * sin(\theta_{lidar\ scan})$
    - $z = 0$ (lidar scans form a plane at z = 0 in the lidar frame)

- These lidar scan coordinates x, y, z are then transformed to the world frame via a $SE(3)$ transformation T_{wl}, which is constructed based on the current robot pose x, y, yaw.
    - $T_{wl} = \begin{bmatrix} R_z(yaw) & p \\ 0 & 1 \end{bmatrix}$
    - $p = [x, y, 1.41]^T$
- The **particle filter** is used to better estimate the pose of the robot given odometry data. The number of particles N_p used for the filter is certainly an important parameter to pick. The state of these particles is initialized to $s = (x, y, yaw) = (0, 0, 0)$, and the weights for each particle are initialized uniformly to $\frac{1}{N_p}$. At each iteration, the **prediction step** will be applied to each particle by altering the state of each particle by the motion model, which is determined by the difference in the lidar's odometry data of the pose s_{odom} from two timestamps and additive Gaussian noise w_t in x, y, yaw.
    - $\Delta s_{odom} = s_{odom}(t+1) - s_{odom}(t)$
    - $w_t \sim N(0,\ diag(\sigma_x^2,\ \sigma_y^2,\ \sigma_{yaw}^{\ 2}\ )$
    - $s\,(t+1) = s(t) + \Delta s_{odom} + w_t$
- Next step would be **selecting the best particle.** The current lidar scan will be transformed using the different estimated pose from each particle. The transformed lidar scan will then be compared to the current occupancy map for **correlation**. The map correlation function will slightly deviate the lidar scans in a 9x9x9 grid with deviations in x, y, yaw because in some situation the original pose of the particle results in low correlation because it's slightly off from the true state. This hacky method will improve the performance of the particle filter. Note that the dimension of the grid (which is 9 above) is actually a parameter that can be adjusted by the programmer. The state of the particle will also be adjusted to the new state (by adding the corresponding deviation to the state) that results in the highest correlation. Each particle will then obtain a correlation (max correlation picked from the deviation grid). Then the particle with the maximum correlation, aka the best particle, will be used as the estimated pose of the robot.
- The next step would be the **map update** using the best particle. The log-odd map will be updated by the lidar scans transformed by this particle's state. Points along the line will be constructed through the Bresenham algorithm in 2-D. Then the updated log-odd map will update the occupancy map. The update procedures of the two maps were described in the 3rd bullet point earlier in this section for technical approach. I came up with a hacky approach that the map update step is skipped if the max correlation is negative, because I don't believe it is reasonable to update the maps with a particle that disagrees with the map very much.
- After the update steps for the two maps, next step of the algorithm would be **particle weights update**. The weights of the N_p particles $\alpha_t^{\{N_p\}}$ are updated with the computed correlation c via the following:

  - $\alpha_{t+1}{}^{\{N_p\}} = \frac{1}{\gamma}[\alpha_t{}^{\{N_p\}} * exp(c - c_{max})]$
  - $\gamma = \sum_{N_p}[\alpha_t{}^{\{N_p\}} * exp(c - c_{max})]$
  - γ is the normalization factor for the particle weights
  - c_{max} is the correlation from the best particle
- In certain circumstances, the particles need **resampling**. In short, a new set of particles will be created, which contain mostly the particles with the highest correlations. There are many methods for resampling. I used the stratified resampling method, which produces the sample set containing high correlation particles at least once and low correlation particles at most once. After resampling, all the weights are The condition for resampling is defined as follow:
  - $\frac{1}{\sum_{N_p}(\alpha_{t+1}{}^{\{N_p\}})^2} < N_{eff}$
  - N_{eff} is a threshold parameter, indicating the number of effective particles

- Complete the algorithm by Iterating through the steps above with all lidar scan data! Every 50 lidar scan data is used to decrease algorithm run-time.

Results and Discussion

Simultaneous Localization And Mapping (SLAM)

The results from the particle SLAM algorithm certainly seems to produce better results than simple integration (see plots for each dataset below). Some **parameter choices** I made are **summarized** below:

- Number of particles: 100
- Lidar Ssan step size: 50
- Sigma of x, y, yaw: 0.6, 0.6, 0.105
- Deviation grid for map correlation search:
  - x_range = np.arange(-0.2, 0.2+0.05, 0.05), in meters
  - y_range = np.arange(-0.2, 0.2+0.05, 0.05), in meters
  - yaw_range = np.arange(-2, 2+0.25, 0.25)*np.pi/180, in rad
- Map size
  - x: -30 to 30. In meters
  - y: -30 to 30. In meters
  - Resolution of 0.05 meters
- Map correlation threshold
  - Occupied: $m_i = 1$, if $P(m_i = 1) > 0.6$
  - Free: $m_i = -1$, if $P(m_i = 1) < 0.4$
  - Unknown: $m_i = 0$, if $0.4 < P(m_i = 1) < 0.6$
- Log-odd value
  - $log(g)$, where $g = 4$
- Valid lidar scan range:

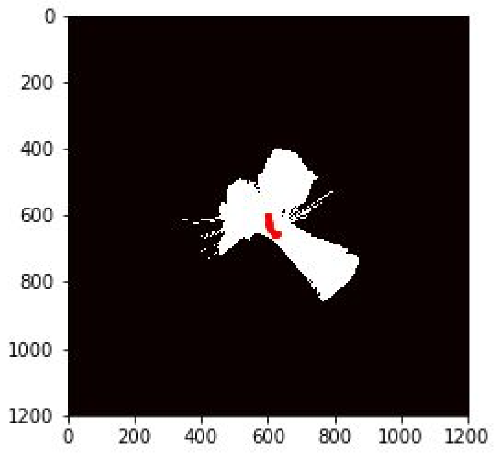
    - 0.1 to 30, in meters
- Resampling threshold:
    - $N_{eff} = 0.2 * Number\ of\ particles\ =\ 20$

Some **observations** I was able to make throughout the assignment is that the values for sigma of x, y, yaw changes the performance of the filter significantly. These sigma values determine how far apart the particles would spread out after the motion model. I believe that high sigma values, with enough number of particles, will lead to better result, because the particles are more spread out and more likely to find the correct state that fits the occupancy map better. Obviously if the sigma values are too high, this would result in the particles jumping around in the map, which is not a desired behavior. Fine-tuning these sigma values will certainly improve the performance of the filter, but it might overfit the parameters to our training set and not perform as well on the test sets.

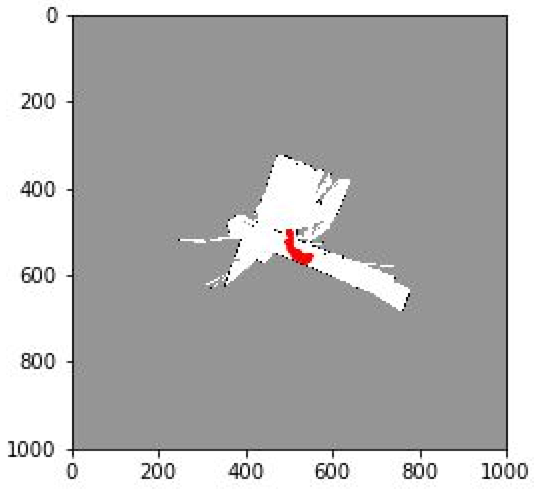
It is also tricky to balance the parameters for filter performance vs. algorithm duration. These parameters include the number of particles, lidar scan step size, and deviation grid for map correlation search. Some datasets just contain too many data points for the algorithm to run in a tractable duration with optimal results. Thus, the tradeoff between performance and time duration must be balanced. Normally this would depend on how much processing power and time the machine has.

The results from the training and test datasets are shown in the next few pages:

Training set 0

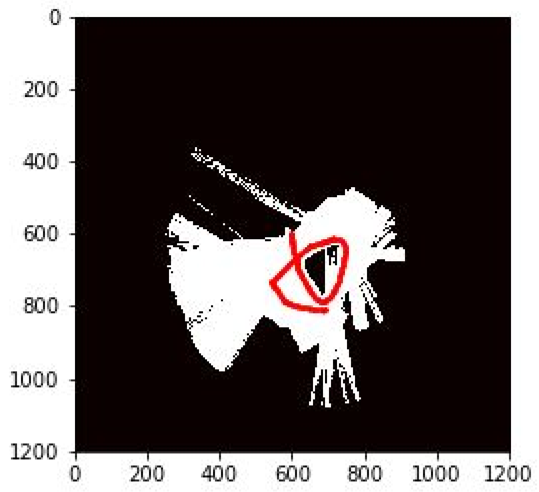


-
    - (above) Simple integration (without filtering)

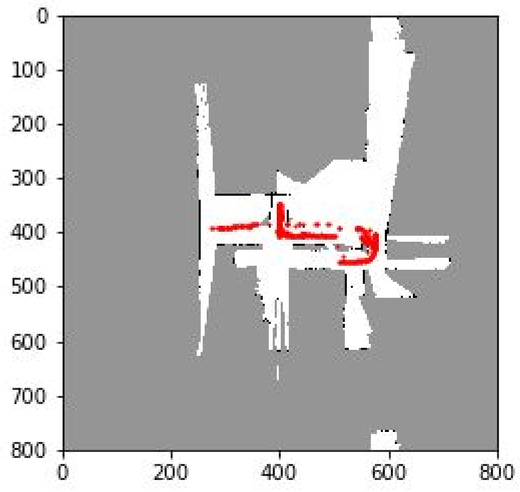


-
    - (above) SLAM w/ particle filter

Training set 1

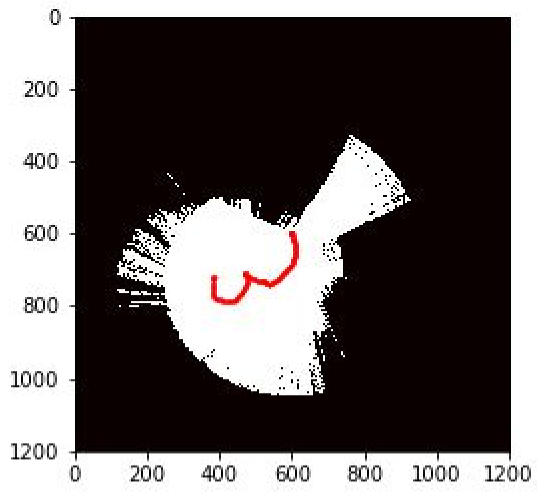


-
    - (above) Simple integration (without filtering)

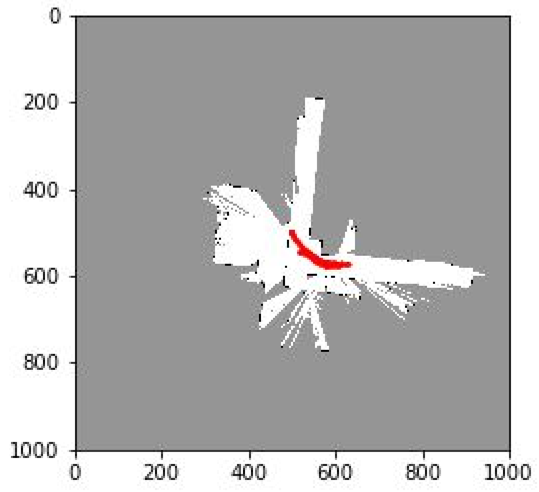


-
    - (above) SLAM w/ particle filter

Training set 2

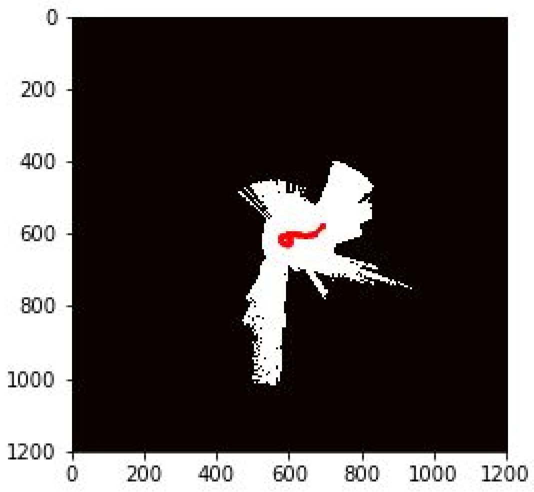


-
    - (above) Simple integration (without filtering)

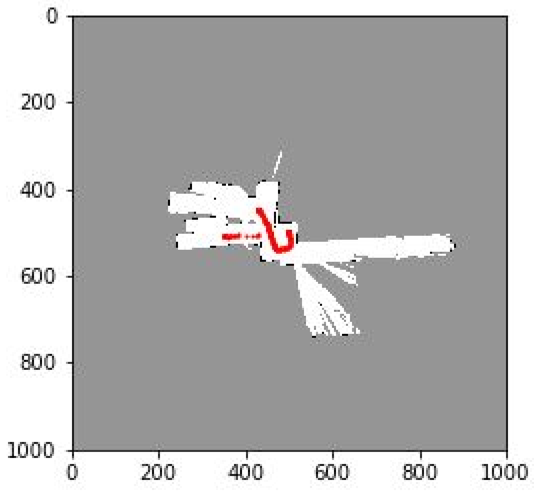


-
    - (above) SLAM w/ particle filter

Training set 3

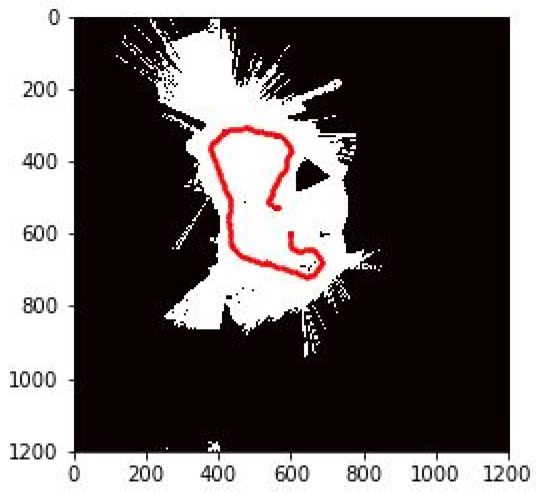


-
    - (above) Simple integration (without filtering)

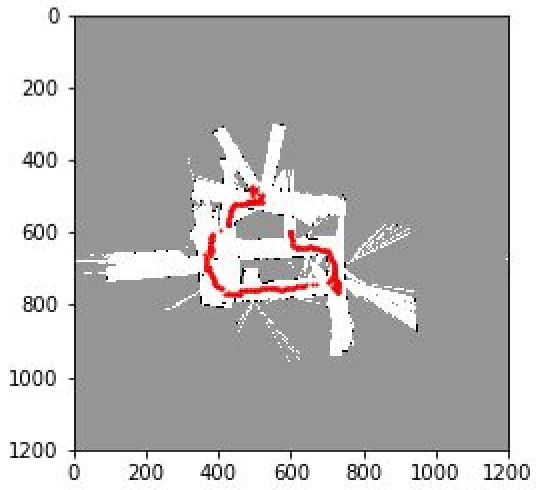


-
    - (above) SLAM w/ particle filter

Test set



-
    - (above) Simple integration (without filtering)



-
    - (above) SLAM w/ particle filter