Project 4: Indoor SLAM for Mobile Robots
ESE 650 Learning in Robotics, 2018 Spring

Dian Chen

March 23, 2018

Contents

1 Introduction

Localization and mapping are two common tasks for most mobile robots. They solve the problem of where the robot is, and how the environment is like, respectively. Simultaneous localization and mapping, on the other hand, tries to solve these two problem simultaneously, in which a robot carries some sensors to perceive the world, localize itself, and immediately to map the environment. There are various approaches to achieve SLAM (e.g., ORB SLAM), and among them the most well-studied, basic one is Fast SLAM.

In this project, we're given LIDAR scans, IMU readings, odometry readings and physical configurations, all coming from a humanoid robot roaming around in the Engineering buildings. We're expected to build a 2-D grid map using these sensor information, as well as localize the robot more accurately instead of trusting the raw odometry readings alone. To achieve that, we should also build a particle filter to estimate the robot's nonlinear motion, instead of EKF or UKF. After that, we are expected to texture the ground on the map, using the RGBD sensor information from a Kinect mounted on the robot's head. This project leads through the basic pipeline of Fast SLAM, without loop closure detection.

2 Particle-based SLAM

2.1 SLAM

Localization solves the problem "Where am I?", given a known map and the robot's perception of the world. The intuition is that the estimated location should be one that can most explain what the robot have seen, i.e., a place that would give measurements that best match the actual measurements. This is usually done using maximum likelihood estimation (MLE), maximizing the probability of a certain set of measurements over possible locations. Mapping is then given the robot's known position, describing the world using what have been seen. In SLAM these two procedures are performed iteratively such that it might seem to be a "chicken or egg first" problem.

2.2 Particle Filter

Particle filter is commonly used in SLAM, with several advantages. First, it gives good enough approximation of the actual distribution, using sufficient number of particles, i.e., samples of the

state space. This is a huge relief if the distribution is highly nonlinear (especially for multinomial distributions where KF and its variants may not work well), or has very complicated analytical form. Second, it doesn't require the models for motion and measurement to be linear either. Motion can be applied to particles naturally, and the measurement comes in such that particles with higher correlation are more likely to "survive", thus the particle set is always composed of particles that explain well the measurements. Particle filter doesn't have a analytical form of representation; it represent the belief using only samples of the state space.

3 Technical Details

3.1 Transformation

Since the sensor information acquired is all with respect to the robot frame, we should perform proper transformation to bring the measurements back to world frame to build maps correctly. This section goes over several kinds of transformation involved in SLAM.

3.1.1 2-D Planar Robot Model

In this 2-D planar SLAM task, we should always take a bird view of the space and consider only the horizontal projections of points from the 3-D real world. That said, our robot has 3 DoF in this 2-D world, namely the position x, y and heading θ.

3.1.2 3-D LIDAR Transformation

In the LIDAR frame, all rays are assumed to: 1) spread evenly from -135 degrees to 135 degrees; 2) lie in the xy plane, radiating from the center and form a fan. However, we still need to transform the points hit by LIDAR rays from LIDAR frame to the world frame. The series of transformations goes like follow:

$$T_b^w = \begin{bmatrix} \mathbf{R_b^w} & \mathbf{t_b^w} \\ \mathbf{0} & 1 \end{bmatrix}, \quad T_h^b = \begin{bmatrix} \mathbf{R_h^b} & \mathbf{t_h^b} \\ \mathbf{0} & 1 \end{bmatrix}, \quad T_l^h = \begin{bmatrix} \mathbf{R_l^h} & \mathbf{t_l^h} \\ \mathbf{0} & 1 \end{bmatrix}$$

$$T = T_b^w T_h^b T_l^h$$

Here, $\mathbf{R_b^w}$ results from the roll, pitch, yaw angle of the robot body with respect to world, and $\mathbf{t_b^w}$ results from xy translation and the height of center of mass. $\mathbf{R_h^b}$ comes from head yaw, and $\mathbf{t_h^b}$ comes from neck length. Finally, $\mathbf{R_l^h}$ comes from head pitch and $\mathbf{t_l^h}$ comes from the distance from LIDAR to head. Also, don't forget that in LIDAR frame each hit point has coordinates:

$$x = d\cos\alpha, \quad y = d\sin\alpha$$

in which d is the distance from LIDAR reading, and α is the bearing of each ray, ranging from -135 degrees to 135 degrees.

3.1.3 Physical Coordinates to Grids

Besides transformations between physical coordinates, we should also take care of the transformation from physical world coordinates to grid indices of our map. For this project we're using a

fixed map, namely a map with fixed range from "xmin" to "xmax" and "ymin" to "ymax, and fixed resolution "res", to avoid extra work involved with dynamically growing a map. Given these properties, the transformation goes like follow:

$$X = \frac{\lfloor x - xmin \rfloor}{res}, \quad Y = \frac{\lfloor ymin - y \rfloor}{res}$$

This takes into account both discretization and flipping the y coordinates which grow in the opposite direction of vertical indices.

3.2 Update Map

3.2.1 Maximum Likelihood Estimation

Particle filter represents the belief using a set of samples of the real state space, and each particle survives based on how well it explains the seen measurements. To quantify this match we assign a weight to each particle based on its correlation with the map. Finally, we choose the particle with the largest weight as the "best particle", i.e., the result of localization.

3.2.2 Update Log-odds

In this project we're using a 2-D occupancy grid map to represent the world. We discretize the world into grids, and for each grid we assign a value to represent our belief of it occupancy, i.e., the probability of being occupied. This is a softer representation compared to the binary map where only 0 is used for free grids and 1 is used for obstacles.

To use Bayes rule to update the probability for each grid, usually we would chain the probabilities up and take normalization. This is computationally infeasible for our map consisting of massive number of grids. A clever, and also intuitive way is to use log-odds for each grid instead of plain probability. The log-odds is defined as the log of the probability of being occupied over the probability of being free. The update rule can be formulated as follow:

$$g_{x,y} = \log \frac{p(m_{x,y} = 1|z)}{p(m_{x,y} = 0|z)} = \log \frac{p(z|m_{x,y} = 1)p(m_{x,y} = 1)}{p(z|m_{x,y} = 0)p(m_{x,y} = 0)}$$

$$= \log \frac{p(z|m_{x,y} = 1)}{p(z|m_{x,y} = 0)} + \log \frac{p(m_{x,y} = 1)}{p(m_{x,y} = 0)}$$

Here, the first term is one of our hyper-parameters, which represents how we trust our LIDAR data in a way that a higher value means that points hit by LIDAR are more likely to be obstacles indeed and a lower value otherwise. The second term is our current log-odds belief and what we actually store in that grid. A reasonable initialization value is 0, which means that we believe the cell to be equally likely free or occupied.

Another thing worth mentioning is that no matter how we trust our sensors, we should never be too certain about the estimated states of the world. We should maintain the ability to recover from false estimations. To to that we should set upper and lower bounds to the log-odds that effectively clip the values stored in the map.

3.3 Localization Prediction

3.3.1 Apply Motion and Noise

The motion information comes from the odometry which is overall drifting badly. However, the differentiation of odometry is relatively accurate and thus can be used to update our particles. That said, we should first differentiate the odometry to get delta:

$$\Delta = o_{t+1} \ominus o_t$$

in which "$\ominus$" means smart minus that take the change in world frame into local frame. After that, we apply the local motion to each of the particle:

$$p_{t+1} = p_t \oplus \Delta$$

To introduce variation into the particle set, which is essential in particle filtering, we finally apply a local Gaussian noise to each particle:

$$p_{t+1} = p_t \oplus \Delta \oplus \eta_t$$

in which η_t is independently drawn from a Gaussian distribution for each particle.

3.4 Localization Update

3.4.1 Improved Correlation

Correlation essentially describes how well a particle explains the current measurements. High correlation should then increase a particle's chance to survive, whereas low correlation should do the opposite. There are various ways of computing correlation, and theoretically those conforms with this intuition would work. A simple choice is to count the number of overlapped obstacles between what the particle postulated and what there actually are. A more sophisticated way is to sum up the log-odds of the grids "hit" by the particle. However, the former one takes no difference in "strong" obstacles and "weak" obstacles, namely obstacles with very large or relatively small log-odds, while the latter one takes too much account of these such that some false identified obstacles may also justify a particle's existence.

After some try-outs, I found an improved way of computing the correlation that takes advantage of both methods. First, I apply a threshold to the log-odds map to filter out the weaker obstacles, leaving only ones that have cumulated strong enough log-odds belief. After the thresholding the log-odds of hit grids are summed up as usual. In implementation, this procedure can be simply done using a binary mask coming from thresholding, and multiply the mask with the log-odds map and finally take the sum.

$$corr = \sum_{x_{hit}, y_{hit}} \text{np.multiply}(g, g[g > thresh])$$

3.4.2 Improved Weight Update

The equations for updating the weights proceed as follow:

1. $a = \log w$
2. $a = a + corr$
3. $a = a - \max a - \log \sum e^{(a - \max a)}$
4. $w = e^a$

However, from careful inspection of the execution, I found that the correlation can easily become very big (since there are about a thousand hit points and each may encounter big log-odds) and overwhelms the term on which it's added to. What happens then is that the weights for all particles immediately collapse into a pulse-like distribution, in which one particle has close-to-one weights while the others have near zero. This greatly reduces the effective number of particles and affects the quality of our particle filter.

To prevent that, I convert the correlation to a predetermined range before adding it to the log weight a in line 2:

$$corr = \frac{corr * scale}{\max corr + 1}$$

After applying this trick, the particle set had achieved better quality.

3.4.3 Particle Resampling

Resampling is another crucial part in particle filtering. It adapts the distribution of particles to one that conforms with the measurements according to the weights computed from the previous step. The ideas is simply assign higher probability to particles with higher weights and re-draw particles from the same set. As for implementation, there are a lot of ways of implementing sampling. The most frequently used ones are systematic sampling, stratified sampling, etc. They are simple to implement, and I chose to use stratified sampling in this project.

3.5 Texture Mapping

After completing the SLAM, we should have acquired a much better estimate of the trajectory of the robot than the raw odometry. We could then use this trajectory, together with synchronized RGBD images to build a textured map. There are basically two ways of achieving this.

1. The simpler way as suggested, is to find points in the RGBD images that correspond to floor, which can be simply done by checking if the height value is close enough to zero, and texture its position on the map with its color.
2. The more exciting, and harder way is to build a 3-D visualization of the entire space, projecting the points back to 3-D world to form a point cloud.

I chose the second option and was experimenting with Point Cloud Library (PCL). However, due to limited time, I wasn't able to finish this part by the completion of this report.

4 Results

4.1 Train Results

The SLAM results of 4 training datasets are as follow Here training set 0 and 2 have achieved

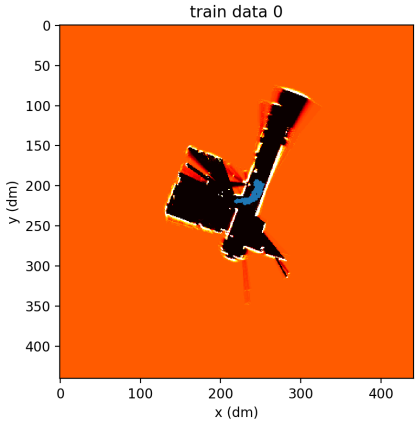

Figure 1: SLAM for train data 0

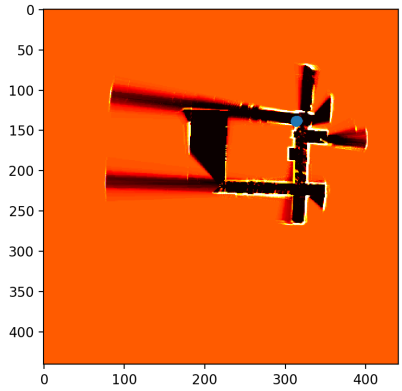

Figure 2: SLAM for train data 1

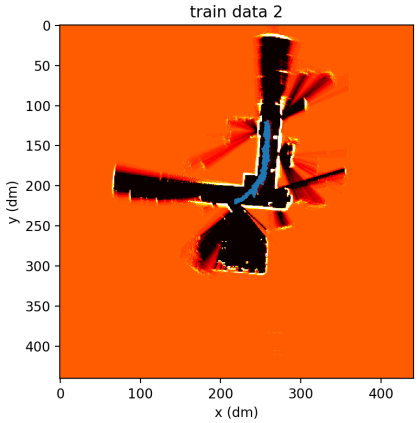

Figure 3: SLAM for train data 2

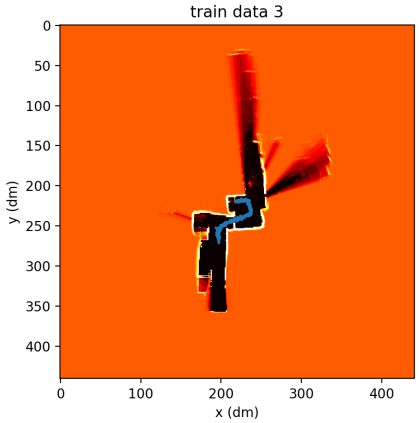

Figure 4: SLAM for train data 3

satisfactory SLAM results, while the corridor in training set 3 is slightly skew. The result for training set 1 is not as good, since it involved a much longer journey in long corridors, which will be discussed in the following sections.

4.2 Test Results

The SLAM result of test set is as follow. In the first half of the journey the map seemed ok. However when the robot had reached the lower right corner of the space, the map started to go off askew. The final trajectory didn't quite close as expected, but the first half of the map came out reasonable. Detailed analysis and discussion will come in the following section.

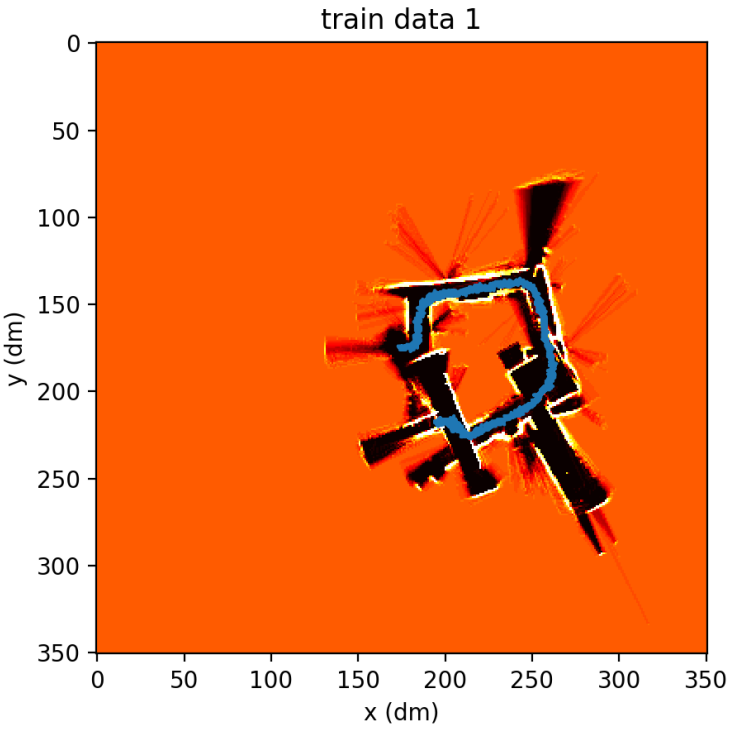

Figure 5: SLAM for test set

5 Discussion

5.1 Recovering from False Mapping

As described above, there are times when the robot would build up "false" obstacles. These falsely identified obstacles may in turn justify some particles that are actually incorrect. Sometimes the built up values may be cleared out by penetrating rays, but most of the times they don't. This is exactly the main reason that led to the failure of my test set. As shown in the map, the lower left corner is off direction badly and the following route is building up on the skew direction. After a discussion with some classmates (Chenhao Liu and Ziyun Wang), we concluded that there's a major slip in yaw odometry when the robot reached that square space at the corner, and our particle filter failed to find a particle that corresponds well to the straight direction. What follows then is that the log-odds for wrong obstacles started cumulating and the shape went wrong.

One possible solution to this is to increase the magnitude of noise, generating particles that can escape the wrong odometry. However I found that too big noises may lead to other problems (will be discussed later), and the problem of false mapping hasn't been solved completely.

5.2 Passing a Long Corridor

The main difficulty in SLAM I've seen in all data sets is that, the localization for the robot becomes much harder when it's passing through a long corridor. This is because the surroundings are so

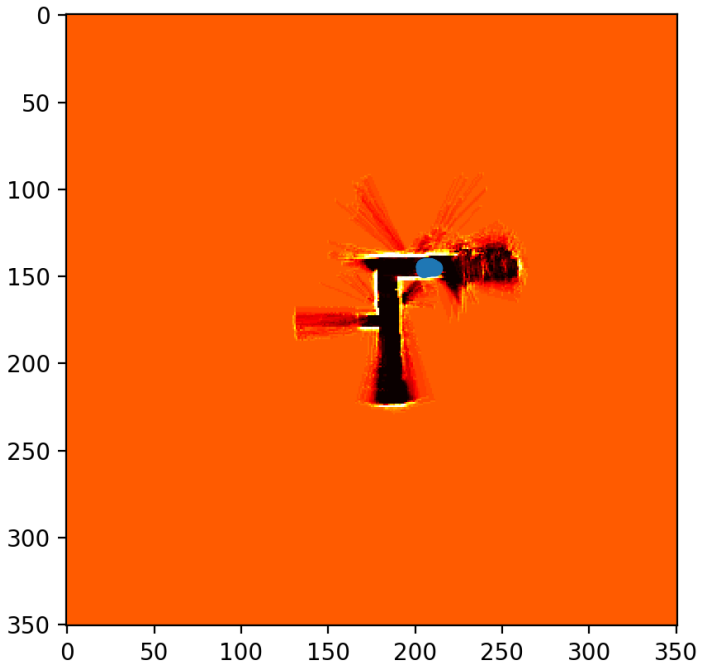

Figure 6: Failure due to too weak anchor

similar in a long corridor, that the main source that would yields most valuable correlation is the wall at the end of the corridor, which effectively serves as an "anchor" that pulls the robot along the direction. If the robot has built up some false "anchor", as described in the previous section, then it is likely to get stuck at a wrong place in the corridor and trapped forever. An illustration of such situation is:

This is again a hard problem I've encountered in this project. A trick that mitigated the problem is to set a higher threshold in the correlation computation step, which basically allows for fewer obstacles to be taken into accounts and at some extent leave out more false anchors. Another attempt is to increase the noise, again increasing the chance of finding particles that can escape the trap.

5.3 Tuning Noise

As said the the previous two subsections, noise plays an important role in the entire process and needs careful treatment. Smaller noise means that we introduce less variance to the particle set and trust more the odometry, whereas larger noise may introduce variance large enough to bring the particles out of some trap. However, noise can either too small or too large. If we use too small noise, the particles would basically follow the odometry which has significant cumulated drift over time. If we use too large noise the particle set may be overly spread out that few has coincided with the actual robot pose.

5.4 Other Comments

In this project I also found other parameters that can affect the performance of SLAM, the most effective one being the resolution of the map. I've tried originally very high resolution, discretizing the physical world into very fine grids, however it seemed that fine grid map doesn't really lead to good performance. When I lowered the resolution to 0.1 meter per grid, the quality of the map becomes much more satisfactory.

Reference

1. S. Thrun, W. Burgard, D. Fox "Probabilistic Robotics"
2. `http://ais.informatik.uni-freiburg.de/teaching/ws13/mapping/pdf/slam12-fastslam.pdf`, Fast SLAM Lecture by Cyrill Stachniss