2D SLAM Report

The report contains six parts, introduction, problem formulation, technical approach, failure discussion, tests results and brief summary.

The key points in the project includes understanding provided data set; proper transformation between global frame, robot frame, head frame and lidar frame; try alternative methods to speed up code which is good for debugging; particle filter resampling; proper noise selection to improve the mapping performance.

I. Introduction

This project is aimed to implement the structure of mapping and localization in an indoor environment using information from an IMU and range sensors.

The basic idea is to integrate the IMU orientation and odometry information from a walking humanoid with a 2D laser range scanner (LIDAR) in order to build a 2D occupancy grid map of the walls and obstacles in the environment.

After this, trying to integrate additional camera and depth imagery from a Kinect One sensor to build a textured map.

II. Problem formulation

A. *Frame Transfer*

The frame transfer is pretty important for this project. The scan data we obtain is from lidar which means that is respected to the lidar frame and also only contains the distance information. Based on the grid map structure, we need to convert the distance information into proper cell position in the global world frame.

Only when converting scan data correctly can we have the opportunity to get good look and reasonable map.

What's more, in the prediction step, transferring frame properly can guarantee the higher accuracy in the pose change of robot. Even just little error will cause total different mapping result since during the whole process, small error will accumulate more and more.

B. Particle Filer Construction and Resampling

In this project, we use particles to represent possible pose of the moving robot. Since we have no idea on the initial state of robot, it's necessary to guess the first pose of robot and then use lidar data to implement the prediction and observation model, just like UKF. There are three main issues in this part.

First, in the prediction step, besides the control input obtained from lidar pose, it's necessary to add random noise to simulate the actual trajectory of robot, therefore, the value and selection of noise matters a lot.

Second, in the observation step which is used to update particles as well as corresponding weights based on the correlation value, trusting the current single particle may not perform well. Enlarging the estimation region and dimension is recommended. For example, apply a 3 by 3 window around certain particle to get more measurements in x and y dimension, and also can add some offset on the yaw angle. Remember to modify the particle information after each time observation if necessary.

Third, as the iteration times increase, some particles with small weights will play little role in the filter process which may cause the number of effective particles decreases. In order to keep a certain number of effective particles that can make a better result, resampling is quite important. Several possible approaches are provided for resampling.

C. *Time Complexity of Program*

The speed of code might be the biggest problem in this project. When your code can run a single map within 10 minutes, it will do a lot of favor to debug.

In the code, we need at least 100 particles for good result, and more particles will bring better performance. What's more, when computing the correlation value of each particle, creating a nearby region is recommended, for example, 3 by 3 matrix for each particle. That means we need at least run 900 loops in just one single iteration time. Consider the length of timeline is about 10 thousand, it will take quite a long time for just a single map.

However, long period running time makes debugging more difficult. Therefore, figuring out approaches to speed up the code is necessary. Besides trying best to vectorization, choose some alternative method to replace the provided function can also work well, for example, when updating log – odds in the map, rather than loop each position along the beam, we can create a contour or mask with certain value to update.

D. *Texture Mapping*

Given the images from IR camera and RGB camera, in order to get a good ground texture map, frame transferring plays an important role again.

In the IR camera, we can extract depth information, with the instinct parameters and correct homography transformation, we map 2D position into 3D world frame, then map that physical position into RGB camera to extract RGB information. Another important issue is filtering points on the ground to do the texture.

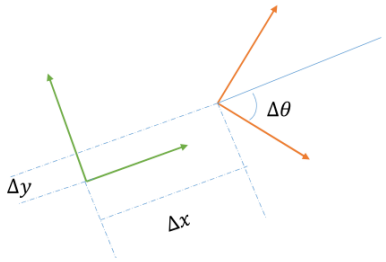
III. Technical Approach

This section shows some several technical approaches in my code to deal with mentioned problems above.

A. *Use Data Correctly*

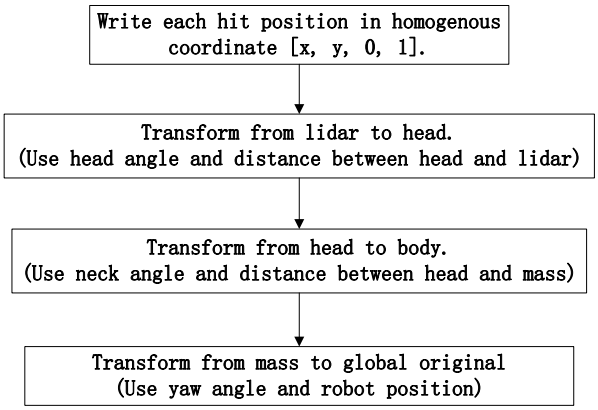
For such a kind of project, including object recognition and UKF, the data truly plays an important role. If we cannot handle data correctly, such as data gathering and transformation, the result must be wrong even with the perfect algorithm. For this SLAM project, we need to handle provided data properly in below several parts.

First, in the prediction step, in order to simulate robot's pose correctly, we cannot use the lidar pose directly which don't show the true physical pose of the robot. What we can trust is the absolute difference of pose between two timestamps. And note that the odometry itself is measured in the local frame, which requires us to interpret the difference within the local frame. Then, we should use estimated particles' states to convert the corrected (local) odometry into the estimated global frame. Below figure show the process roughly.



Second, in the map estimation step, it's pretty important to transfer the scan data into global frame correctly. The confusion is, scan data only shows the distance information between lidar and hit object but we need to find its corresponding position information in the estimated grid map. Converting distance information to position data, we need the scan angles to do this transformation.

What's more, the obtained scan data is respected to the lidar frame while the grid map and robot pose is measured in the global frame, and the robot also has its head and pose orientation, so combining several transformation together to project scan data into the global frame is important. Below diagram shows the procedure to map the scan data.



B. *Mapping*

For 2D SLAM, we use occupancy grid to simulate the actual map, just like an image which is looked from top to bottom, each cell represents single position within the environment. What we need to do is to figure out the occupied and empty cell in the environment via the scan data.

The trick is, in order to make the result more reasonable, we choose the log – odds map instead of the binary map to store our detection.

The basic idea is, for each detected hit position, we add log – odds ratio which represents the probability of occupied. For example, we add positive value of a cell, e.g. log(9) to show this cell is highly likely to be occupied, while for empty position where we add some negative number like log(1/9) to represent the confidence of empty.

$$g(1)=\frac{p_k(z=1|m=1)}{p_k(z=1|m=0)}=\frac{90\%}{10\%}=9$$

$$g(0)=\frac{1}{9}$$

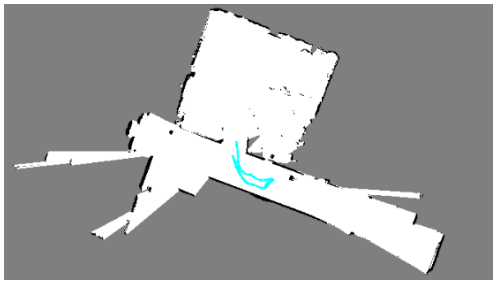
And we need to accumulate log – odds ratio over time and generate the log – odd map.

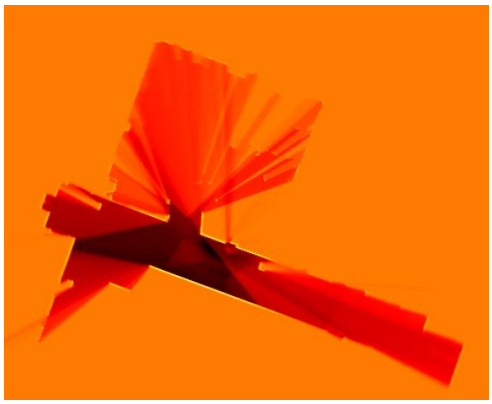
$$\lambda(m|z_{0:t}):=\log o(m|z_{0:t})=\lambda(m)+\sum_{s=0}^{t}\log g(z_s|m)$$

Finally, we can choose a threshold to determine the occupied and empty position in the map.

$$p(m=1|z_{0:t})=1-\frac{1}{1+\exp(\lambda(m|z_{0:t}))}$$

Below are binary map and log- odds map of data set, you may find the log - odds map can show the state of map more accurate.





The log – odds map shows the probability of occupied state more clearly, the deeper the color is, the more likely to be an empty one.

C. Noise Selection

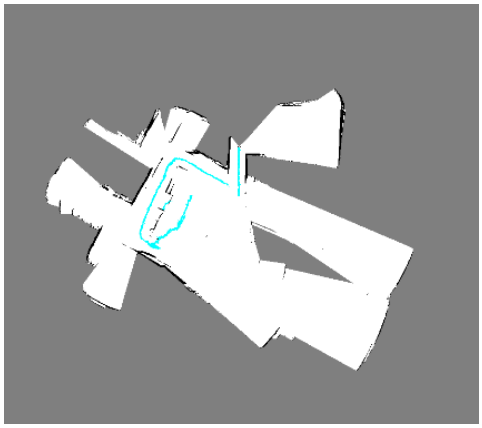
Another key point in this project is the noise selection. Since the pose information obtained from lidar is not accurate without noise. Adding noise is simulating the actual process when the robot is moving and rotating in certain environment.

However, the value of noise is pretty important.

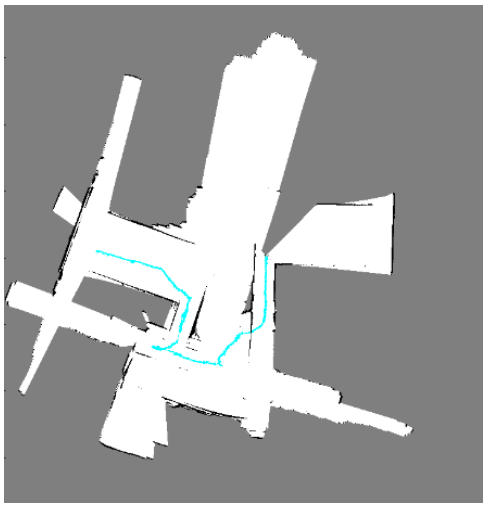
If we choose small noise, that means we trust the lidar data a lot and assume the robot is moving smoothly both in directions and pose orientation;

If we use the noise with larger noise, we may assume that robot's moving is fast and we cannot trust the lidar data very much. Below are two results with different noise.

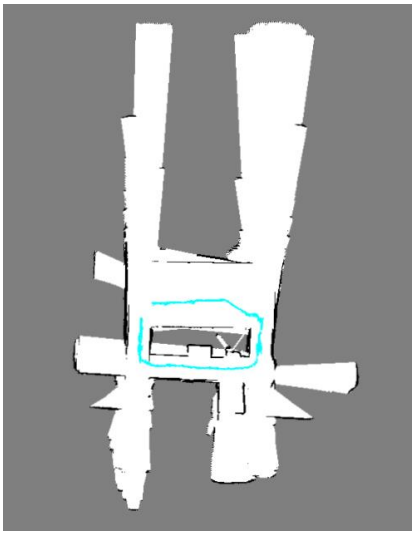
When I add a larger noise, you may find the trajectory has some jump state and even unreasonable gap, because the large noise totally change the previous state in some case.



But when the noise is too small, in some corner position, its route will change to other direction which makes the final map totally wrong.



Selecting suitable noise will truly improve the result.



In fact, noise selection should be a trade – off issue, since the robot may have different states when moving in a straight line and changing its direction in the corner. That is, when the robot traverses in a straight route, its noise should be small for the high possibility to being low speed. But when meet a corner and need to change its direction, it's quite reasonable to add more noise. Therefore, it's better to add noise based on the continuous state of robot instead of choosing a certain value of noise.

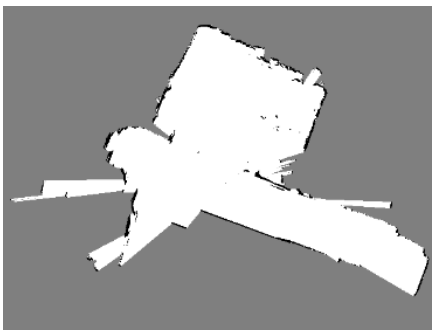
Of course, it might be a little complicated to take continuous states into account when choosing the noise, in order to reduce error as much as possible, we can use the random normal noise and add different value at each iteration.

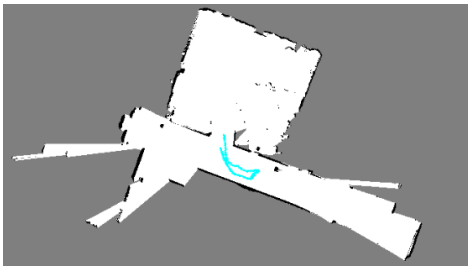
D. Variation in Particles

When computing the correlation value to get the best particle for map estimation, it's pretty important to add some variations for each particle. That means, we should allow some error for each particle and then find the optimal one which is similar to the estimated particle in order to get a better result.

For example, we can add a 3 by 3 or 9 by 9 position matrix for each estimated particle, compute correlation value for all of them and pick up the particle with the max value as updated one. It will truly increase the accuracy of mapping. In addition, we can also give small offset to yaw value and try to estimate yaw angle better.

Below two figures show the function of variation adding.



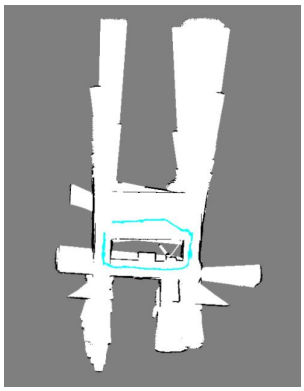


The upper one is the result before variation adding, and the bottom one is obtained after adding 3 by 3 variation in x and y dimension. Although two results look similarly, you can still find that the second one is better than the first map, which means the variation truly improves the result.

IV. Discussion on Failures

In this project, we are provided four training datasets and one test set. Three of them performs pretty well except the training data 1 and test set.

Although I have tried to use different value of noise and also add more noise in yaw angle, the best result of train data 1 shows below,



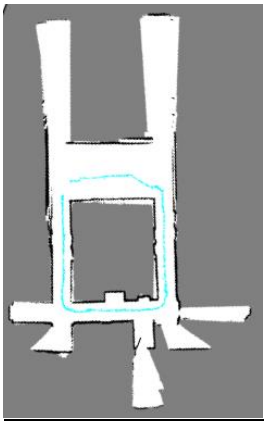
It looks like reasonable as well as the trajectory, but in some parts, the mapping result is not good. You can find there is an oscillation in the right corridor.

The reason might be the computation of correlation value. Since I pick the particle with the largest correlation value, in some cases with two similar or even equal max number, the function will return the one with smaller index, it might be fine in just one single step, but when this error accumulates, the final result will be totally different.

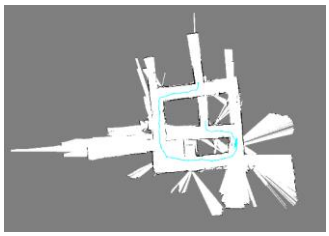
The alternative approach to deal with this problem is using log – odds map to estimating correlation value and increase the variation dimensions in all x, y and yaw angle.

Of course, the noise is another issue, just as what I pointed out above, it's better to choose the proper noise based on the continuous state of robot, smooth movement should be given less noise while adding more noise when changing robot's direction.

Below figure shows the improved one after implementing above approaches, the result is obvious better than previous one.

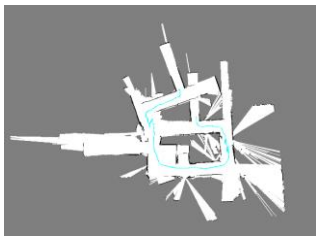


For the test data set, it looks not good at first time. The result shows below.



At the beginning, the trajectory is reasonable as well as the scan result. But when the robot moves to the right corridor, it meets the same problem as the second training data, unreasonable oscillation and followed by wrong direction change. That might due to the large noise happening during the smoothing moving.

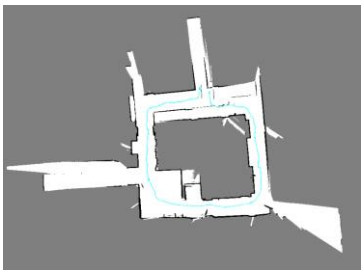
Then I reduce the noise in order to get a better performance. However, the updated one is also not good.



Just as what I pointed out before, small noise might perform worse when robot changes its direction at some corner position. According to the result above, it happens trajectory drifting at the left – upper corner, but the good news is, reducing noise can truly improve the performance at the right corridor part.

I have tried different combination of noise but any of them cannot improve a lot. Now I focus on other issues to improve the result.

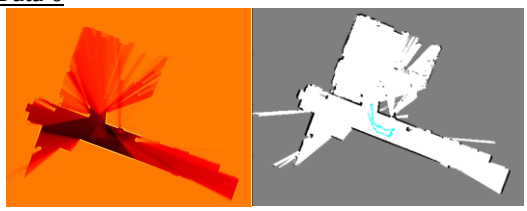
First, I take the roll and pitch angle of robot into consideration when transferring scan data into grid cell. What's more, I apply log – odds map instead of binary map into correlation function which can increase the accuracy. Finally, I increase the number of particles when filtering. And above operation truly improves the result.



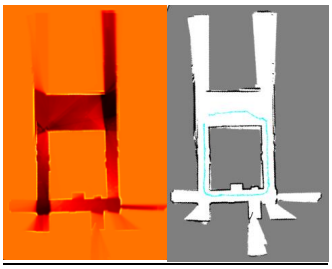
V. TEST RESULTS

Total 4 training data and 1 test data, I plot both the log – odds map and 2D SLAM map, and you can find the trajectory (blue curve) in the 2D SLAM map to check the accuracy of my algorithm.
(The first one is log – odds map while the second one is 2D map). You can also find corresponding videos that show mapping procedure via the below link.
https://drive.google.com/open?id=0B-YfsvV6PlJRMU56U2h0S2VaV0U

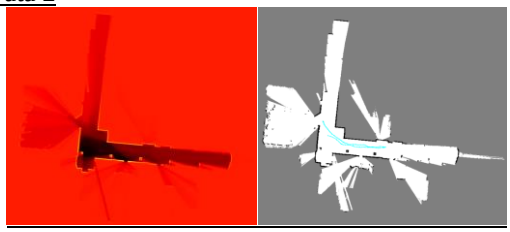
Train Data 0



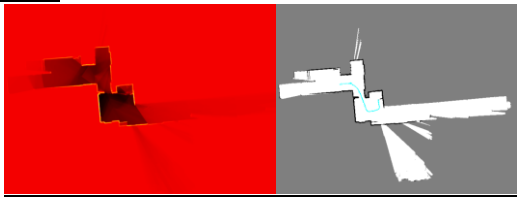
Train Data 1



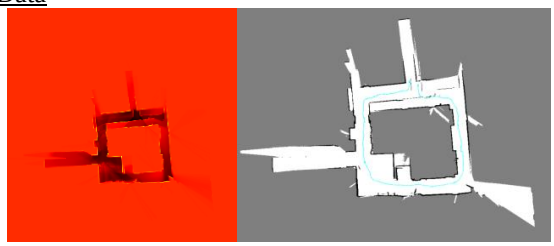
Train Data 2



Train Data 3



Test Data



Log – odds map looks similar to the binary map while the log one is much clearer. Since the log map stores the probability or confidence value of occupied and empty state, so it's reasonable to set a suitable threshold based on the log – odds map to generate a better binary map finally.

VI. BRIEF SUMMARY

After completing three robotics projects, I get to know the importance of data. Since all trained data is obtained from actual robot measurements which is raw, converting data into correct digital becomes quite important for the project. Even with a perfect algorithm, you cannot get the correct result without proper transformed data. For this SLAM project, transforming the difference of odometry and scan data into the estimated global frame is crucial.

What's more, a recommended approach for this project is testing each function individually to increase the probability to get the final result correct. I truly support the idea above since we cannot get right with any wrong code part.

Below are suggested procedures,

1. Try mapping from the first scan and plot the map to make sure the scan transferring is reasonable.
2. Try dead-reckoning and plot the robot trajectory to check the accuracy of prediction step.
3. Try prediction only and plot the robot trajectories (100 for N=100 particles), due to the random noise, 100 curves should be drifted with time accumulates, but the first several moving should be reasonable.
4. Try the update step with only 3-4 particles and see if the weight update makes sense.
5. Only when make sure all four steps correct that can you combine them all together to implement SLAM.

For me, it might not be difficult to implement the basic function of SLAM, but debugging is a pretty tough process, especially with quite long time to run a single map. Visualization works fine to me, but the problem is, if happens some error in one single step, you cannot figure that out well, but with the iteration times accumulating, the small error will be larger and more significant. Therefore, my suggestion is, trying to speed up code first and then set some label point where happens drifting.