ESE 650, Learning in Robotics, Spring 2016: Project4

Yu-Cheng Lin

Introduction

In this project, we try to accomplish SLAM(simultaneous localization and mapping) based on the lidar scan of a humanoid robot. This problem is difficult because a map is required to recover the pose of a mobile robot, and the pose of the robot is required to construct the map. One approach to solve this problem is particle filter. This approach is better in most cases compared to a Kalman Filter. First, the particle filter can model any type of distribution instead of just gaussian. Secondly, as the number of landmarks grow, the number of states that the Kalman filter needs to keep track of grows as well. Since the inversion of matrix at the update step is $O(n^3)$, the Kalman Filter can become quite inefficient. The particle filter, although better than the Kalman filter for localization purposes, still has its weaknesses. As the odometry gets worse, the noise one must inject into the odomentry update. The more noise there is, the more particles are required to localize effectively. Also, the number of particle required at higher dimensions also grows rapidly. Lastly, the noise in all dimensions are not independent given a fixed number of particles. Since particle filter uses Monte Carlo sampling, if I choose to search a larger space in one dimension, I will get fewer samples in other dimensions too. This makes tuning difficult.

Implementation

Grid-Based Particle Filter

For a robot that uses Lidar as its perceptual sensor, a grid-based particle filter would work better than for example FastSLAM because all scans are indistinguishable. A grid-based approach has a predifined probability of $P(O|S)$, this is the distribution of the grid being occupied given that the scanner returns true. This distribution is used to update the map. For localization, the prediction step uses odometry with an injected gaussian noise. $P(x^i|odometry) = N(x^i, C)$, where the C stands for covariance. In this implementation, C is just a simple spherical covariance. Then, once prediction is complete, the correction step follows. The correction step aims to maximize the joint probability of $P(x|odometry)$ and $P(scan|x)$, which is proportional to the posterior $P(x|odometry, scan)$.
The particle filter uses a Monte Carlo method and importance sampling to model this distribution. $P(scan|x)$ is evaluated by the correlation of the existing local map and the hypothesis scans of each particle. The best particle is then picked and the weights are updated based on the posterior of each particle. Resampling is performed when a number of particles have weights so low that they are basically ineffective.

Hacks

To improve the integrity of mapping, I figured that I can update the map only when I am sure the robot is localized properly. In this case, the correlation data can be used assuming that the scans between time steps vary slowly if we have the true pose of the robot. To accomplish this, I implemented a simple box filter on the moving average of the past 20 steps of correlation values, and the map is only updated when the new scan has at least, say 90 percent of the filtered value. This makes my mapping more stable, which improves localization as well.
In addition, at every update, I apply a sharpening kernel to the cells I am updating to discourage one-cell

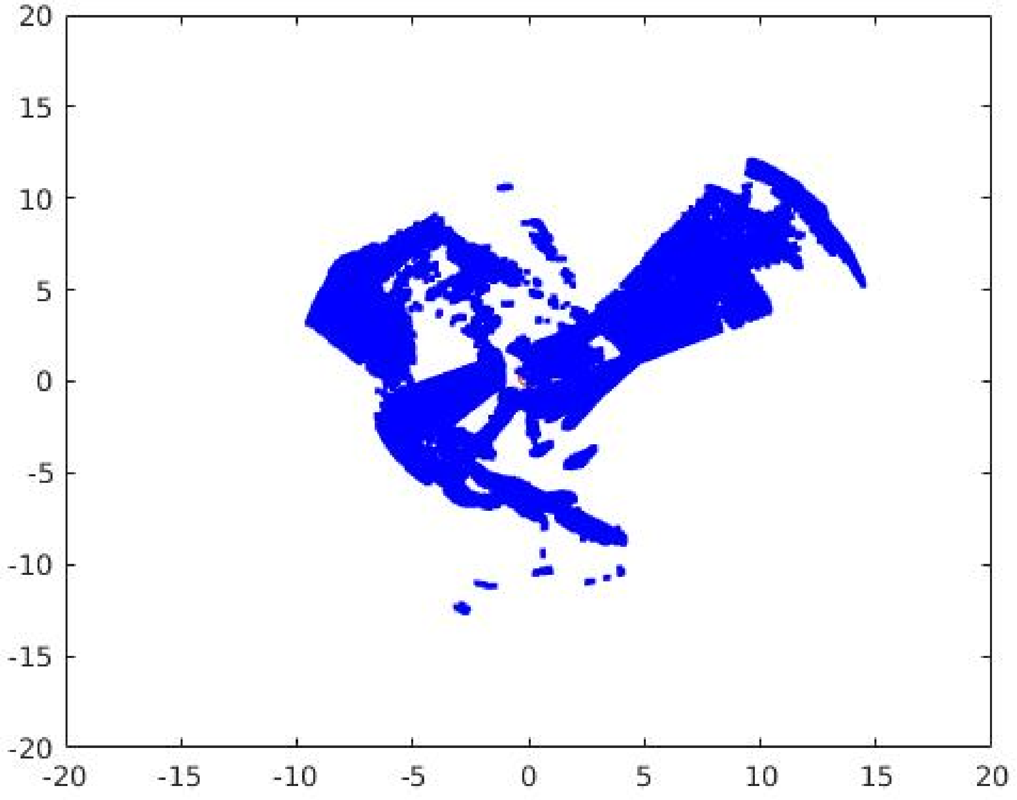
shifts of the map. The downside for this method is that I can clear out occupied cells if I have sparse readings, normally on the cells that are further away from the robot. The orders are map 0 through map3.

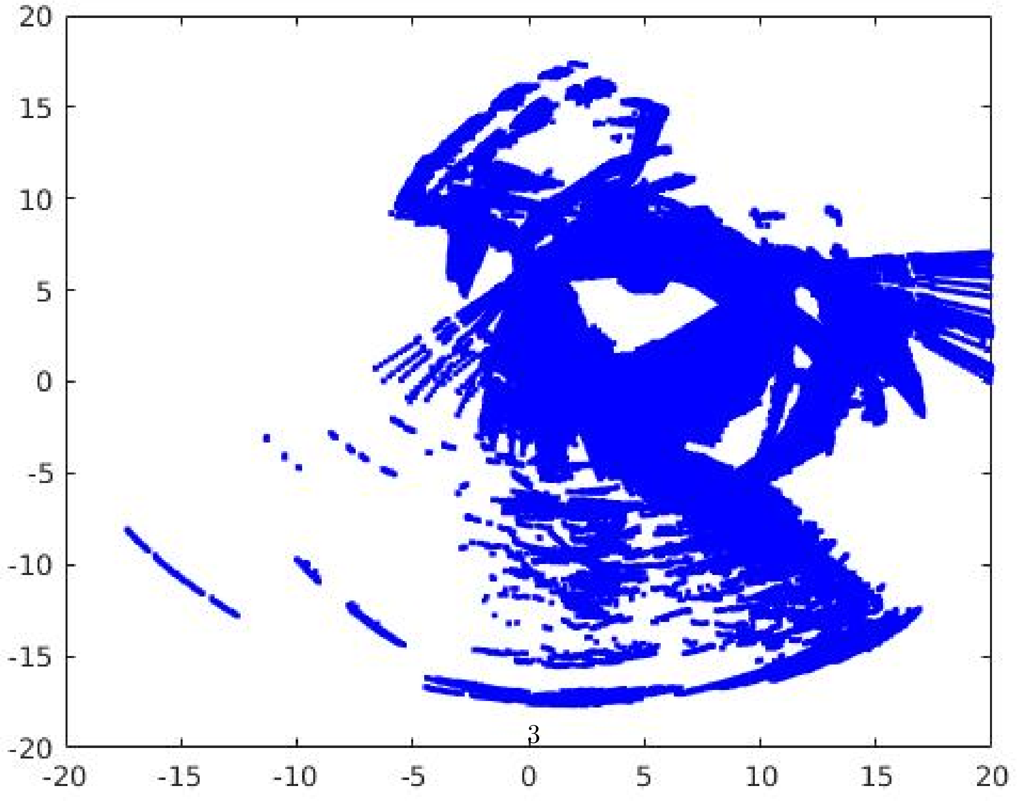
Ground Detection

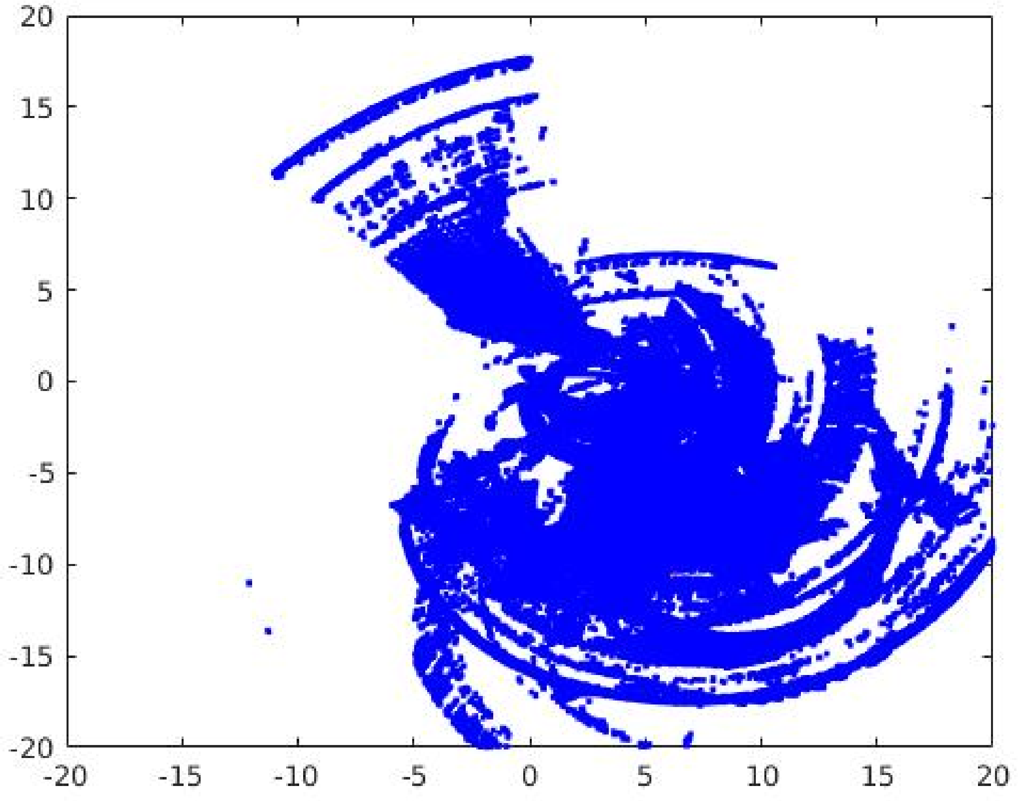
In order to detect the ground, I incorporated the RGBD data into the SLAM. I first get a prior on the expected vector of the ground plane using the pitch of the head angle. Then, I projected the depth readings into point clouds in the camera frame given the depth camera calibration. Given the expected vector, I can do a selective RANSAC on the point cloud to find the best points on the ground plane. Lastly, the points are transformed into both the world frame and the RGB camera frame, and the pixel values in the RGB camera values are pasted onto the grid map in the world frame.
Interestingly, this method ends up picking up the ceiling on dataset 0 because the ground plane takes up way fewer points in the depth point cloud. This can be easily solved by constraining the y coordinates of the points in the camera frame. The results are quite interesting.

Trainin Set Results

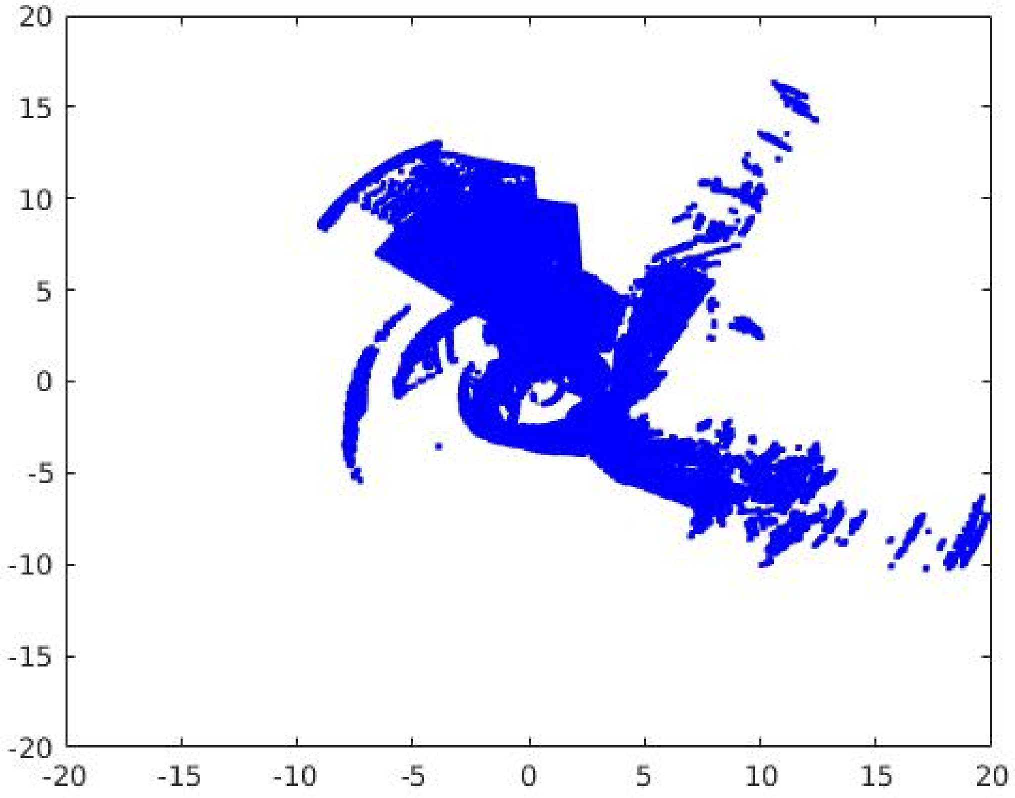
Dead Reckoning



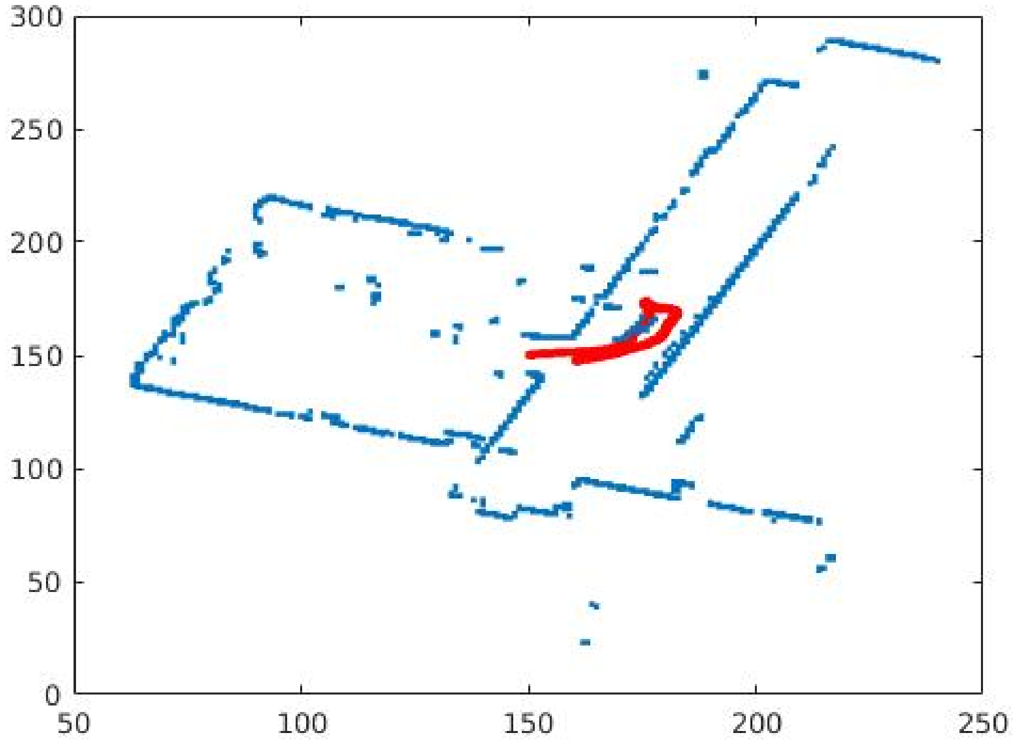




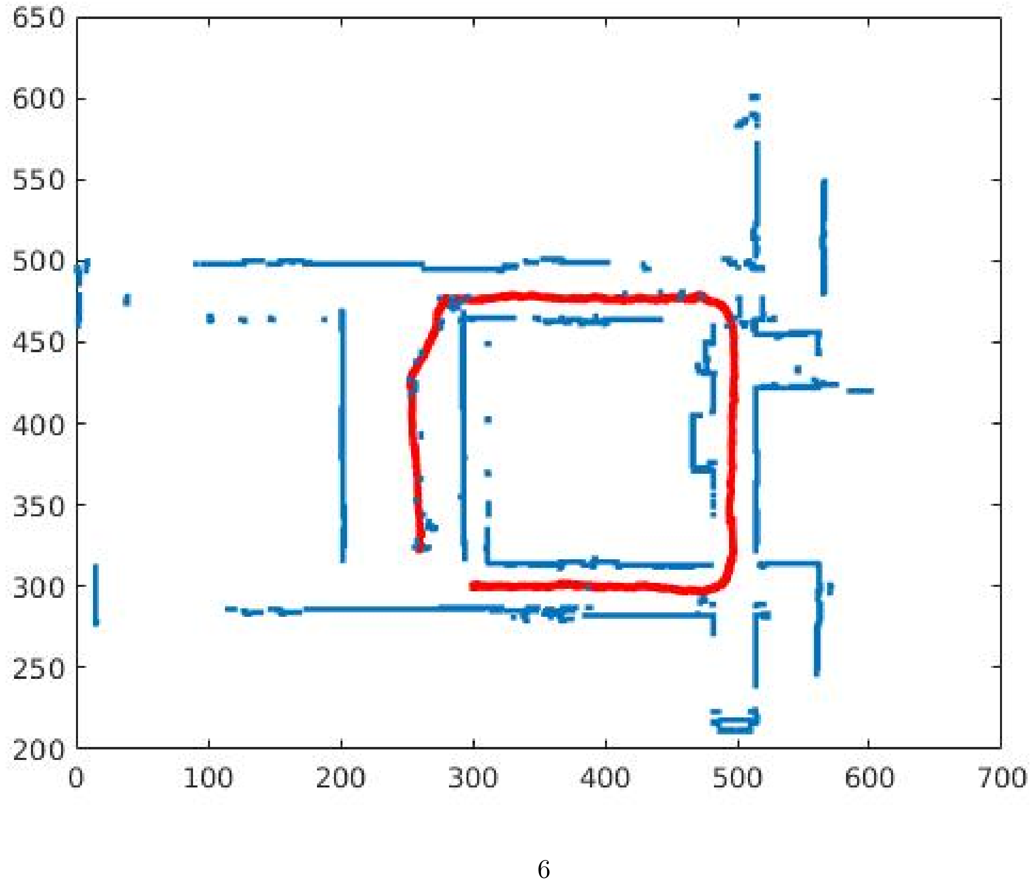
20
15
10
5
0
-5
-10
-15
-20
-20
-15
-10
-5
0
5
10
15
20


20
15
10
5
0
-5
-10
-15
-20
-20
-15
-10
-5
0
5
10
15
20

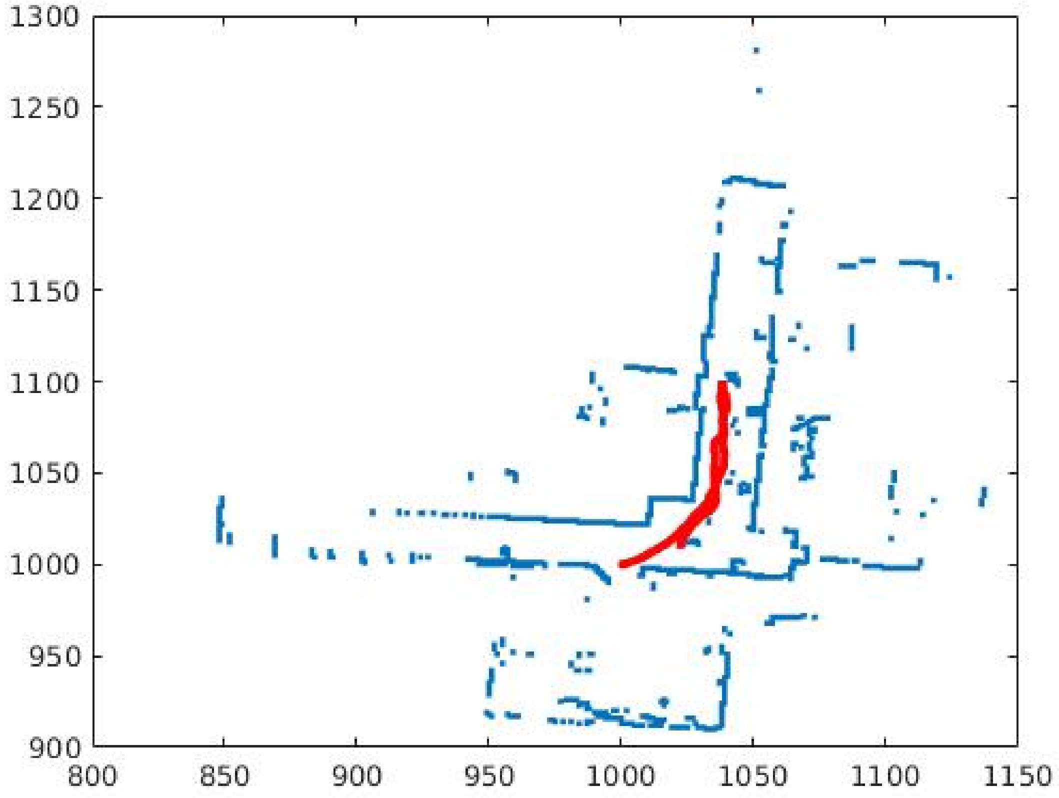
SLAM

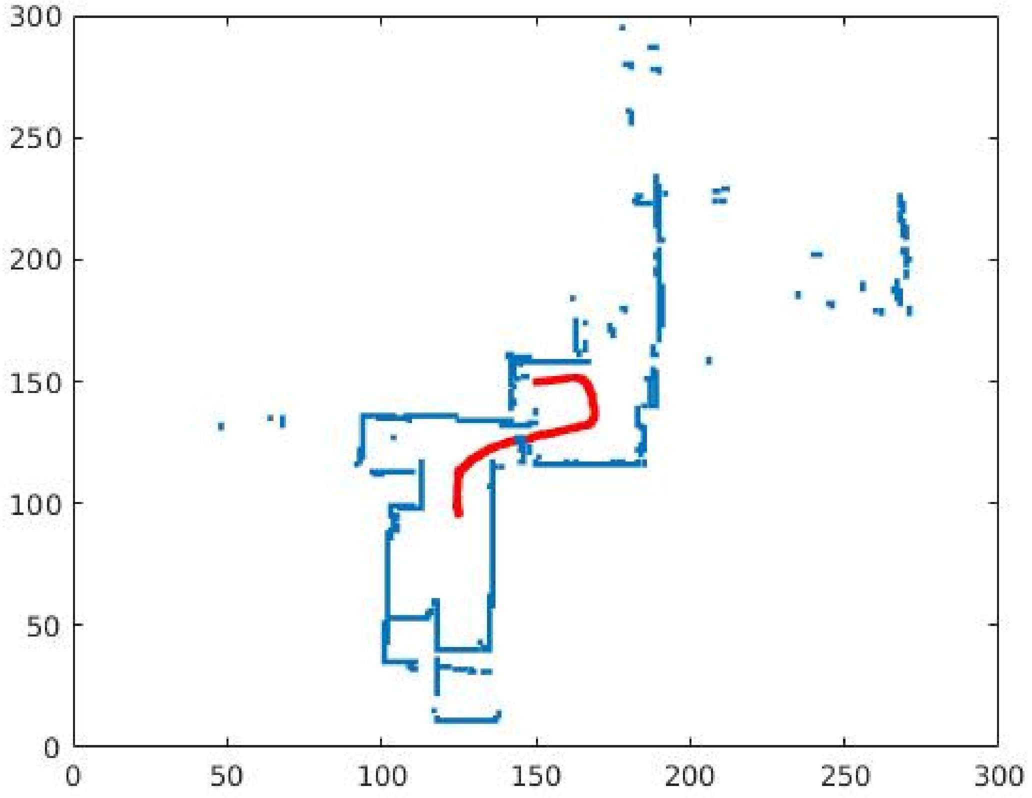


300
250
200
150
100
50
0
50
100
150
200
250

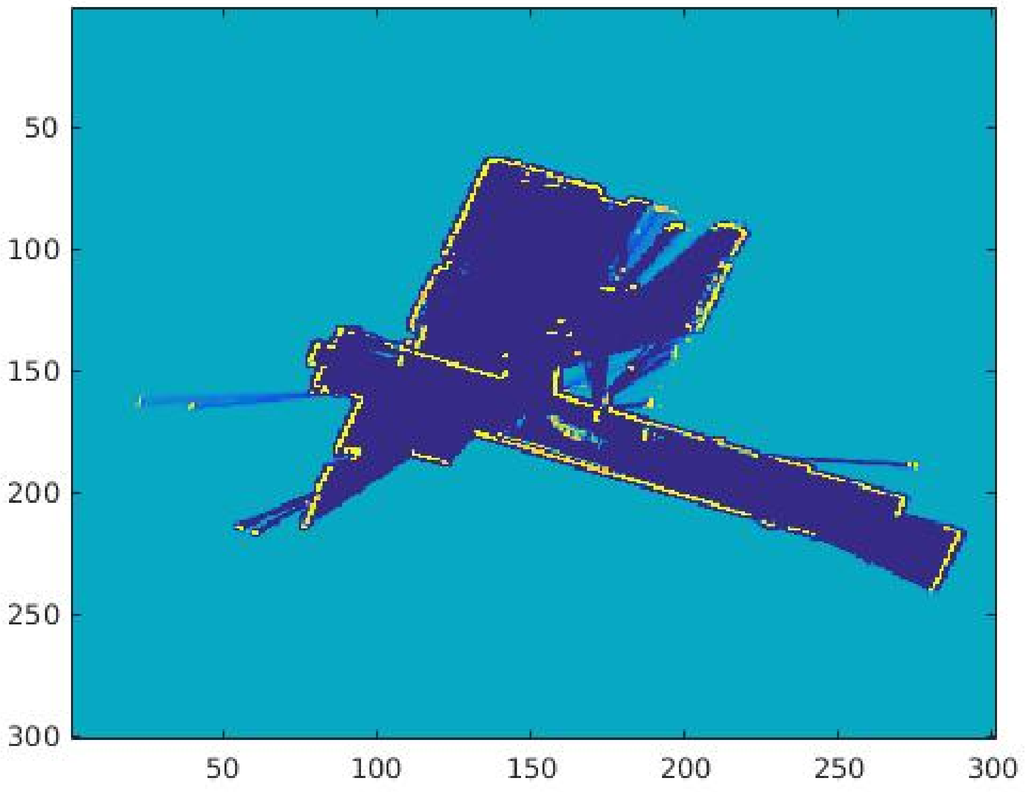


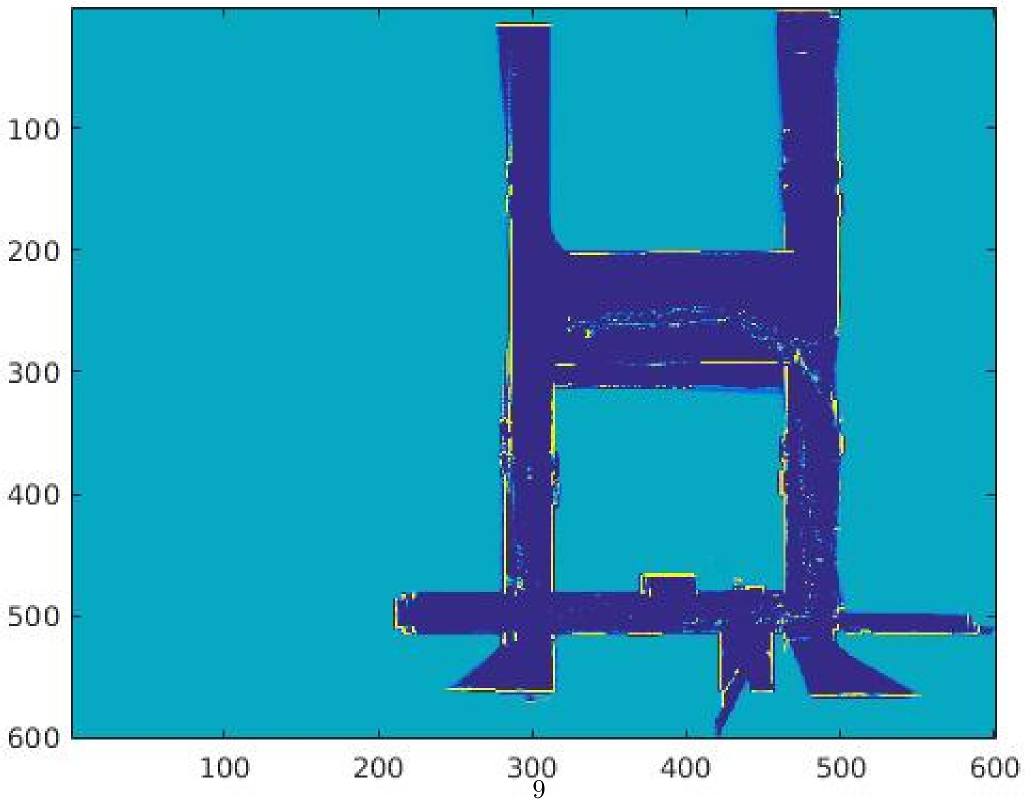
650
600
550
500
450
400
350
300
250
200
0
100
200
300
400
500
600
700

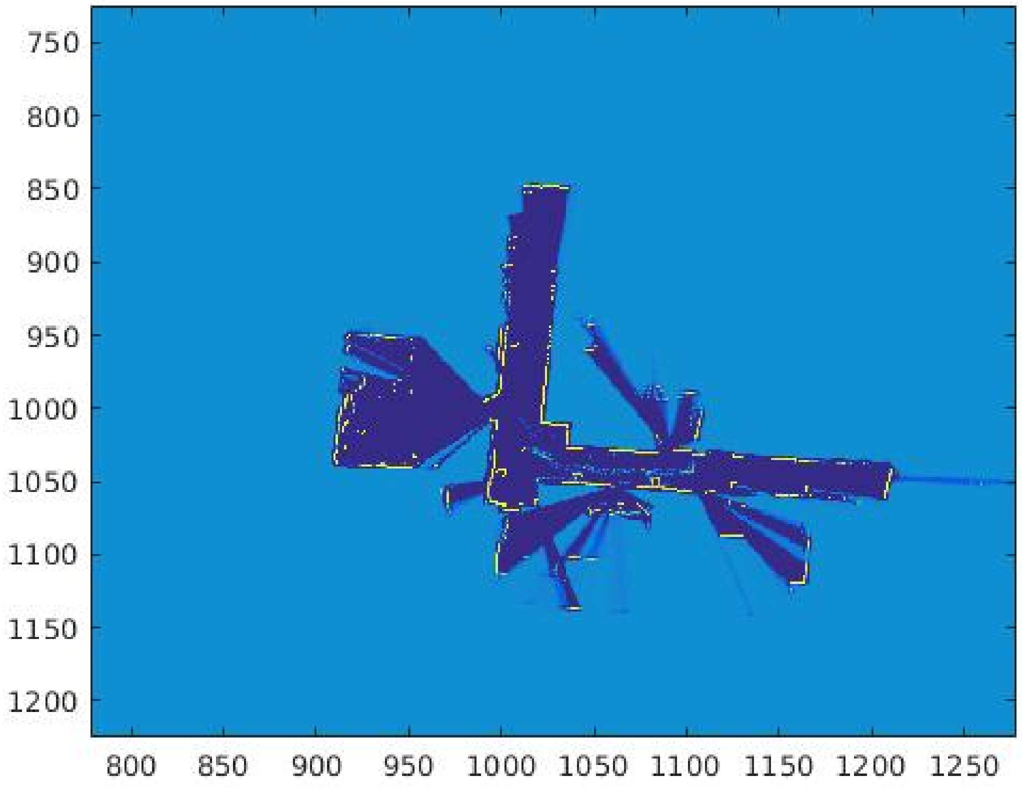

1300
1250
1200
1150
1100
1050
1000
950
900
800
850
900
950
1000
1050
1100
1150


300
250
200
150
100
50
0
0
50
100
150
200
250
300

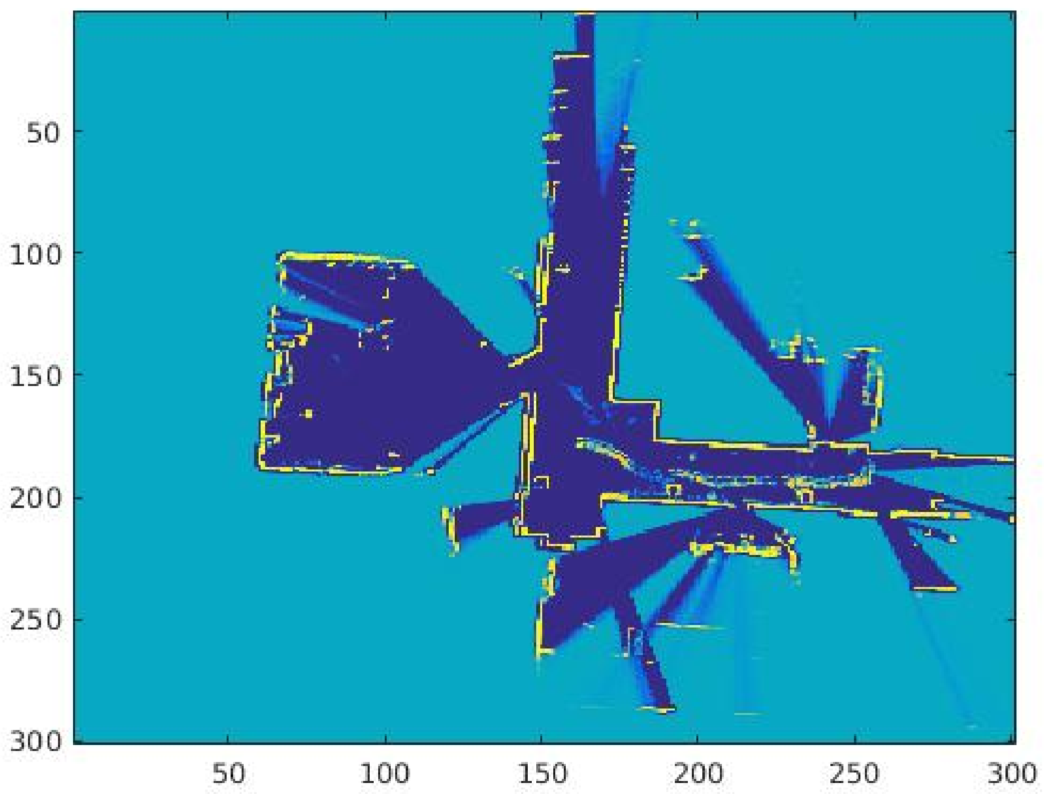
Log Odds Map


50
100
150
200
250
300
50
100
150
200
250
300

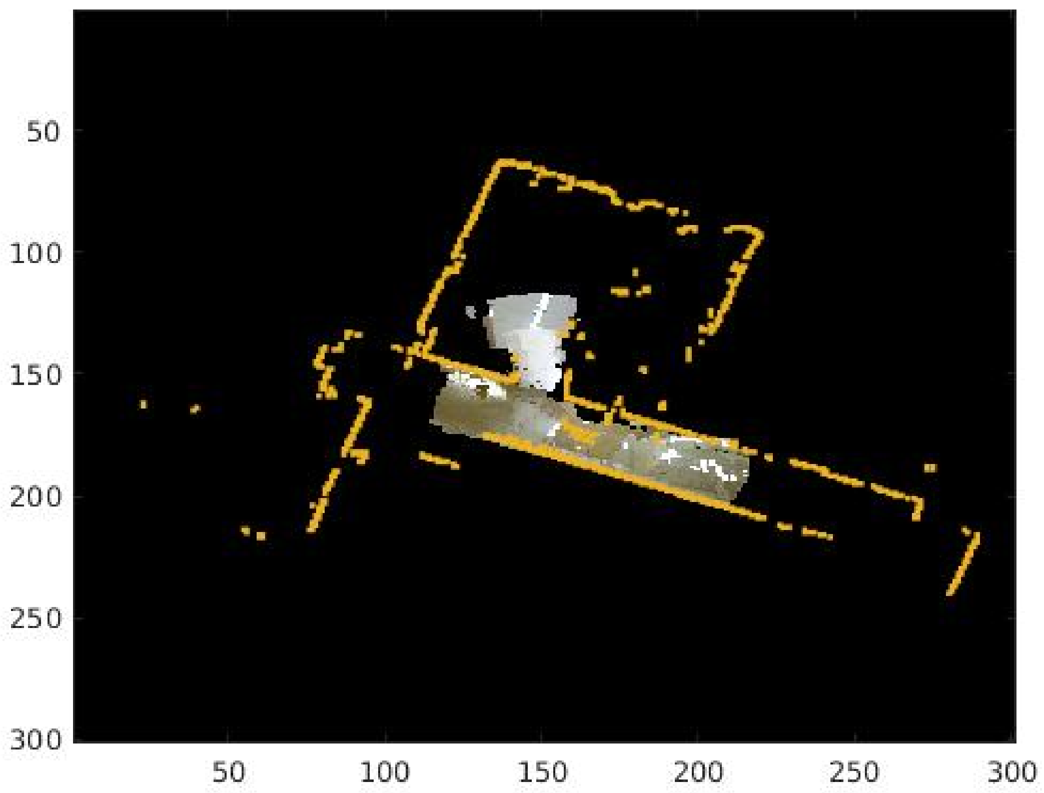

100
200
300
400
500
600
100
200
300
400
500
600

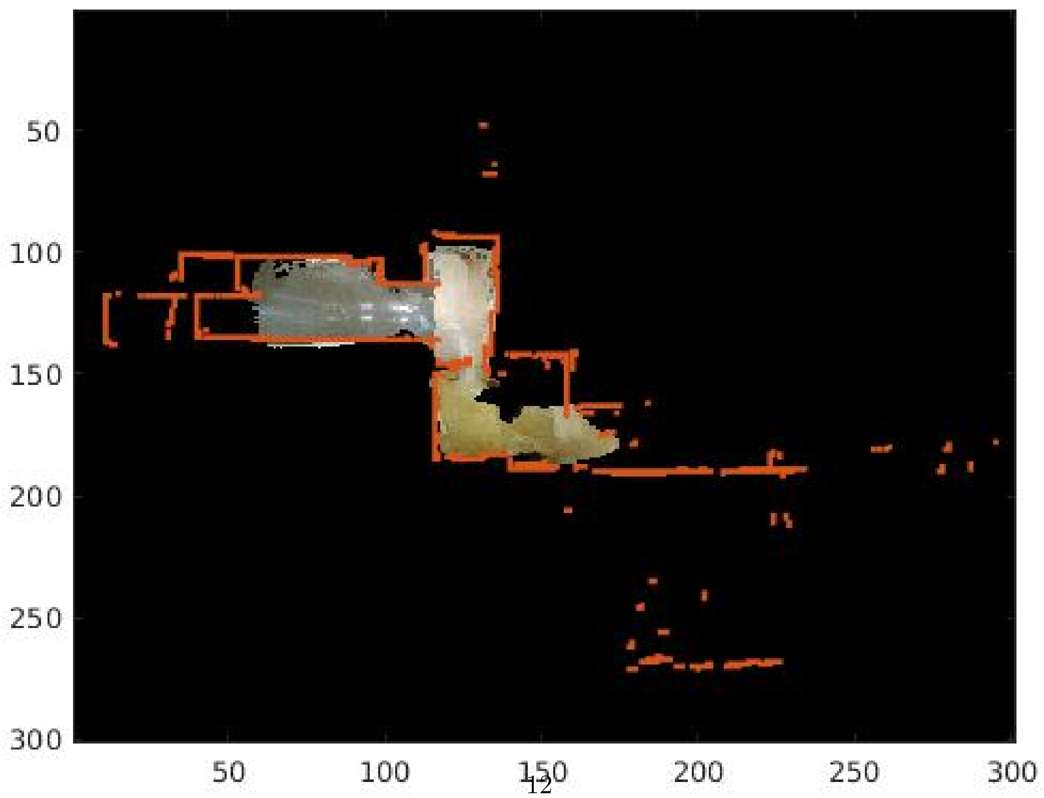


750
800
850
900
950
1000
1050
1100
1150
1200
800
850
900
950
1000
1050
1100
1150
1200
1250


50
100
150
200
250
300
50
100
150
200
250
300

Textured Map


50
100
150
200
250
300
50
100
150
200
250
300


50
100
150
200
250
300
50
100
150
200
250
300