ESE650 Project 4: Localization and Mapping

Mar 23, 2015

Guan Sun
guansun@seas.upenn.edu

Phase 1: Localization and Mapping using odometry data

Algorithm Description:

1. Read the location and orientation (x, y, θ) of the robot from the odometry data.
2. Read the lidar range data at this time, convert detected obstacles from the polar coordinate to x-y coordinate.
3. Use the robot orientation and location to transform the obstacles from the liar frame to world frame.
4. Draw the obstacles in a 2D grid map.
5. Repeat the above steps till the end of data.

Results:

The maps built are shown below, it is obvious that the map is far from the actual environment. The reason is it is built only based on the noisy odometry data.

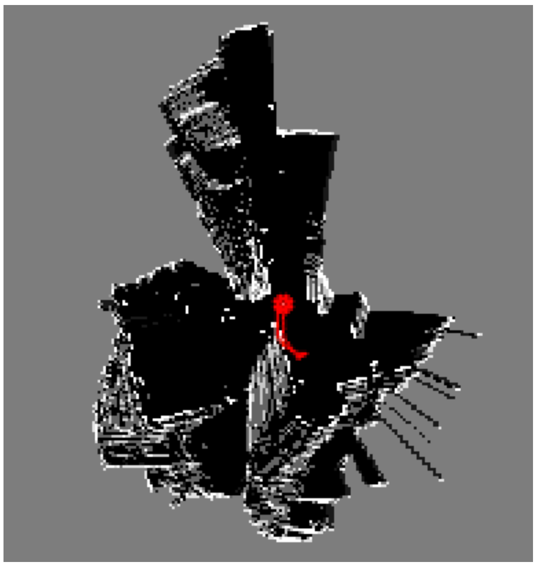
Fig1. Result of dataset0

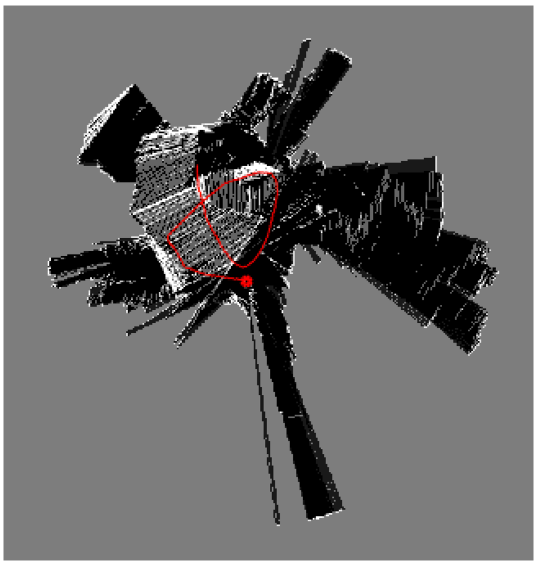
Fig2. Result of dataset1

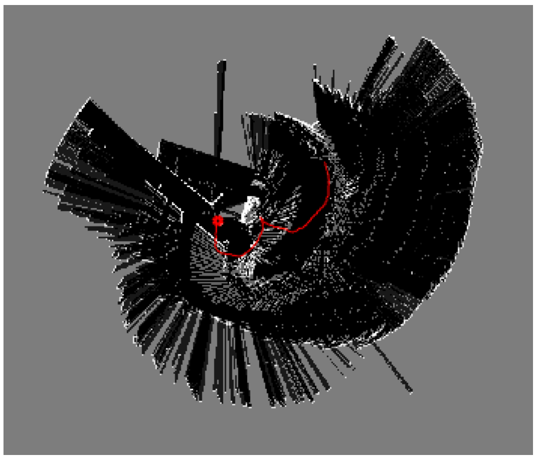
Fig3. Result of dataset2

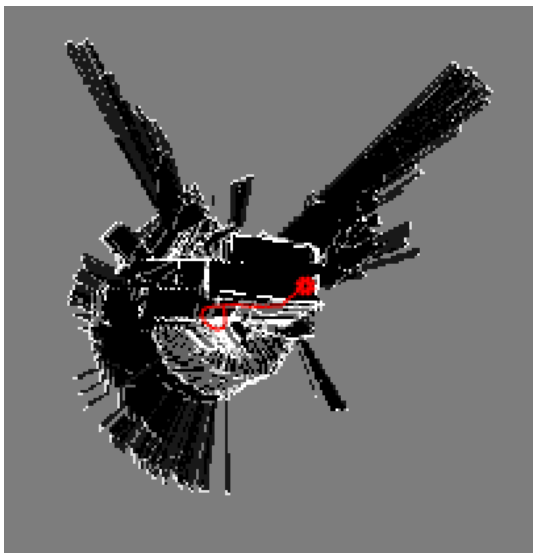
Fig4. Result of dataset3

Phase 2: SLAM using particle filer

Algorithm Description:

Particle filter is implemented to achieve high accuracy localization and mapping simultaneously. In my algorithm, I used 1000 particles and set the variance of the pose change to $(\delta x, \delta y, \theta\delta) = (0.05, 0.05, 0.02)$.

1. Initial 1000 particles with same state (read from odometry data) and evenly distributed weights.
2. Particle Dynamics: move the particles with the odometry increasement plus some random noise to get 1000 new particles.
3. Measurment: For each new particle, generate a obstacle map using the lidar data at this moment.
4. Correlation: Compute correlation of the original map and the 1000 maps generated from the particles.
5. Resample: Use the correlation result to update the weights of particles. Then use the weights to resample and get 1000 new particles.
6. Update the original map with the new map form the most weighted particle.
7. Repeat the above 2-6 steps till the end of data.

Results:

The maps built are shown below. Paticle Filter has dramatically improved the localization and mapping accuracy.

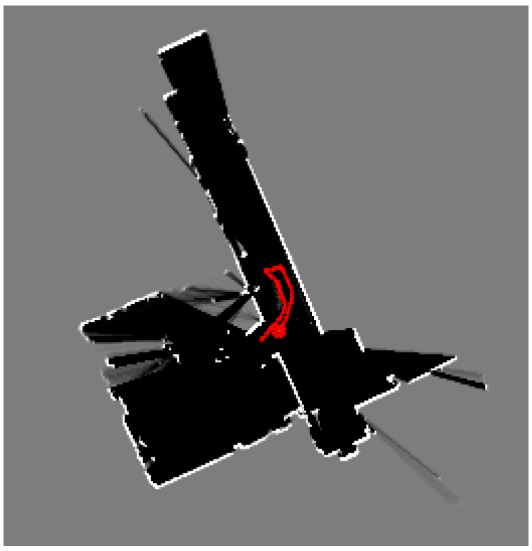
Fig5. Result of dataset0

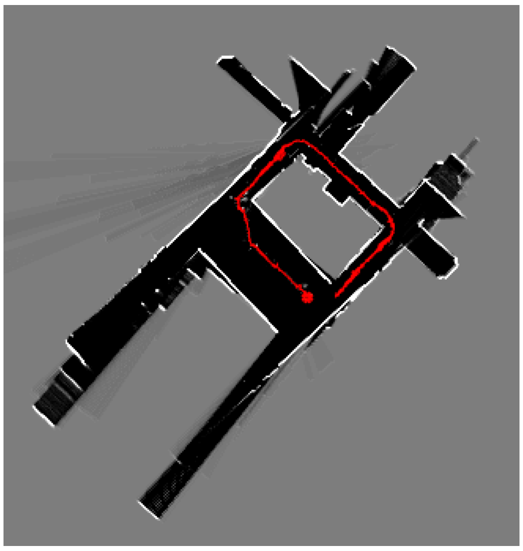
Fig6. Result of dataset1

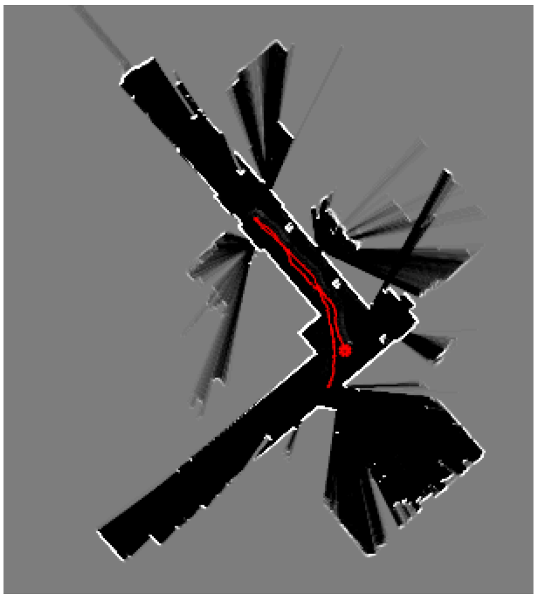
Fig7. Result of dataset2

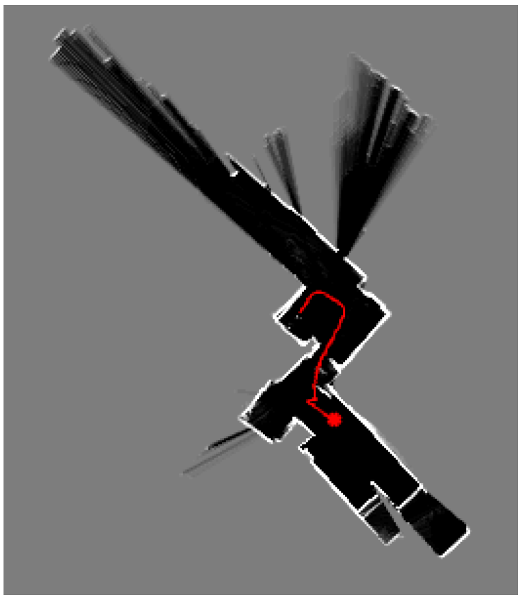
Fig8. Result of dataset2

Phase 3: 3D reconstruction using Kinect data

Algorithm Description:

1. Pre-processing: The RGB camera and depth camera of Kinect have different image resolutions and different FOV. So the first step is to resize the images and align the RGB image and the depth image pixel by pixel based on the same FOV.
2. Compute the (x,y,z) in Kinect frame: transform the (x,y,z) from pixel values to meters using the camera parameters(focal length and principle point).
3. Transform to world frame: get the rotation matrix R form the robot orientation, the robot's head angles. Use R to transform the coordinates from Kinect frame to world frame. Then add a translation based on the position of Kinect in real world.

4. Project the points in RGB image to the world frame. That is a 3D point cloud.
5. Ground Detection: set a threshold of z, regard all the points with less z values are on the ground.
6. Detect the ground plane from the point cloud of each moment. Plot all the points in x-y plane. That's the result map.

Results:

3D construction results:

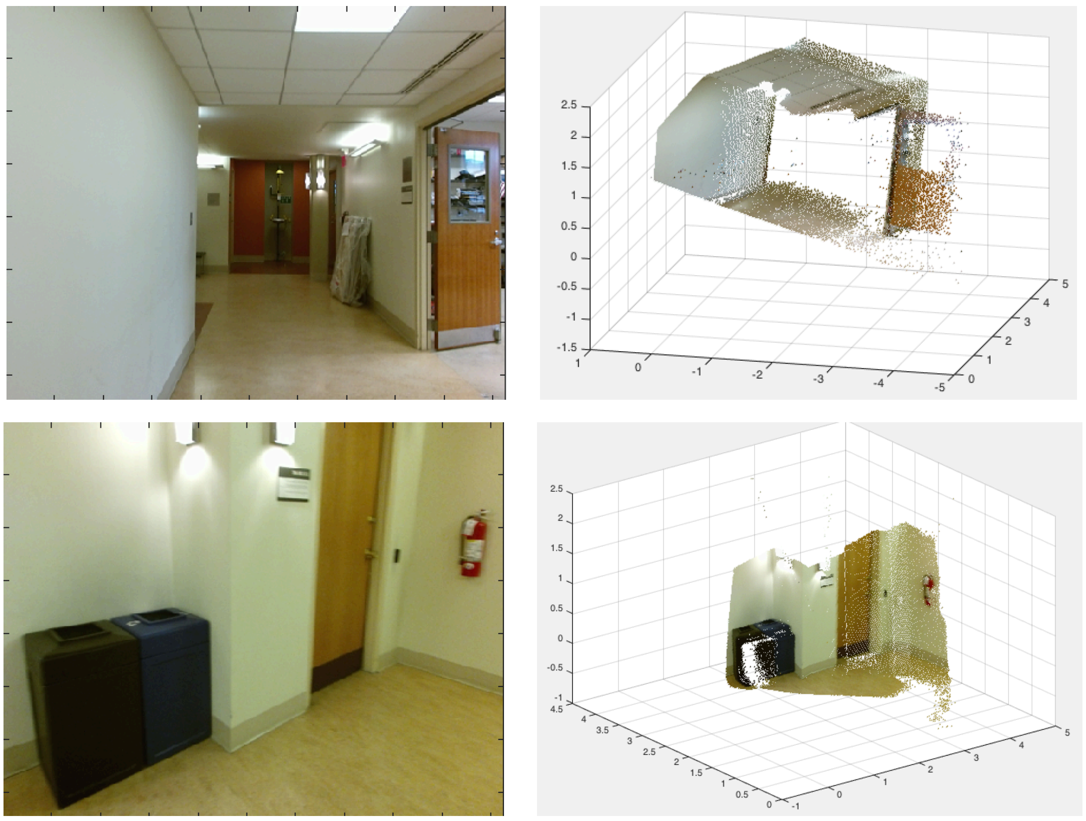

Fig.10 3D construction result

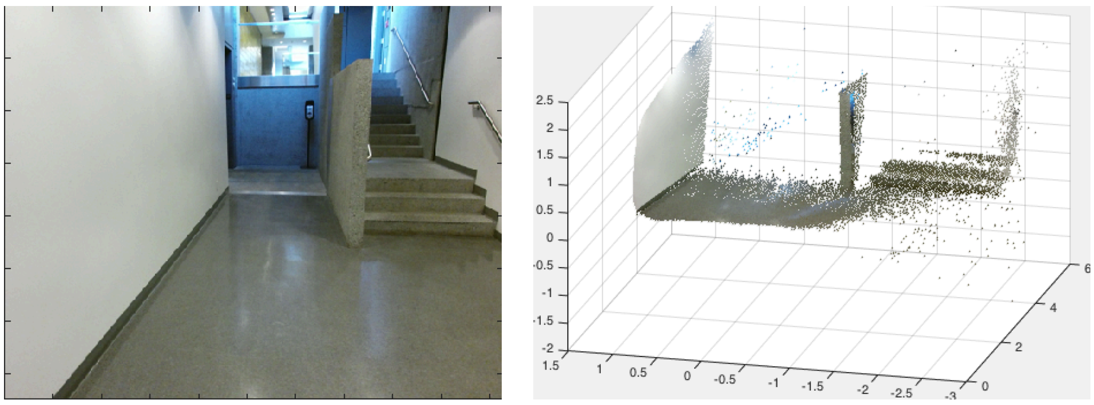

Fig.11 3D construction result

Mapping result:

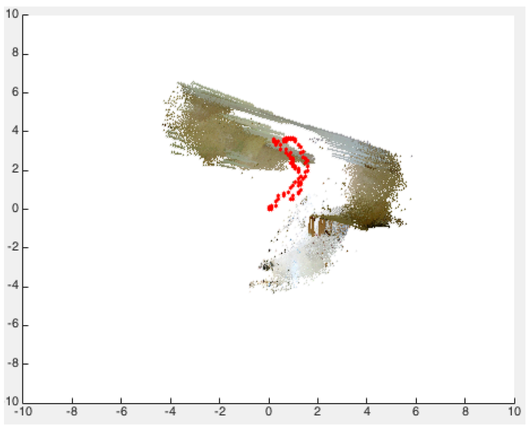

Fig.12 Mapping result of dataset0

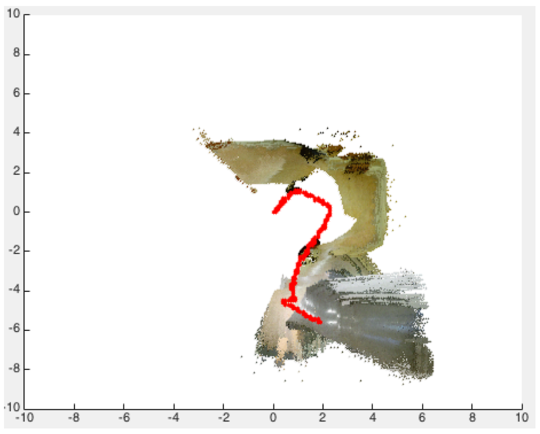

Fig.13 Mapping result of dataset3

Running Instructions:

1. Make sure the ‘code’ folder is in the Matlab’s ‘current folder’.
2. Run the ‘main_phase1.m’ or ‘main_phase2.m’ or ‘main_phase3.m’ file.
3. Make sure to change the data_no variable when loading data to get the corresponding results.
4. Please note the SLAM could be very slow.