ESE 650 – Simultaneous Localization and Mapping (SLAM)

Vishnu Purushothaman Sreenivasan

Introduction

Problem Statement

The objective of the project is to use data from the magic robot (a mobile robot), with odometers on each of the 4 wheels, a LIDAR, gyroscope (3 axes), accelerometer (3 axes) and a kinect sensor to localize the robot and simultaneously map the environment using Particle Filters (PF) (SLAM). We are required to color the ground using the kinect rgbd data.

Description of approach

I removed the bias and multiplied the sensitivities to the IMU data (accelerometer and gyroscope) after smoothing. Then I ran Unscented Kalman Filter to get the yaw, pitch and roll of the robot at every time instant. This data is only used to remove the LIDAR ray points hitting the ceiling or the floor when the robot is on a ramp. I first do dead reckoning using only odometery and then get an approximate map size. I use 100 particles in my particle filter. I perform motion based on the odometery and then randomly jitter the particles along x, y and the orientation (will be called φ in this report) with a Gaussian centered at 0 and a standard deviation of 3 times the change in x, y and φ. Then I compute weight as the sum of the map values at the hit positions of the LIDAR rays where the map value is the log odd value of it being an obstacle versus an empty space. I choose the particle with the largest weight and update the map with the LIDAR points. I increment the log odds of the obstacles by 30, I use Bresenham’s algorithm to decrement the points which are empty by only 1. Finally I resample every time based on the weights of the particles. I have not colored the ground with the kinect data yet.

Methodology

The methodology can be split into preprocessing and the particle filtering (SLAM).

Preprocessing

a) I first took the accelerometer and the gyroscope readings and removed the bias and multiplied by the corresponding sensitivities. I smoothened the accelerometer readings to remove the high frequency noise. I ran Unscented Kalman Filter to get the yaw, pitch and roll angle at every time step.

b) I interpolate the yaw, pitch and roll angles, imu data and the encoder counts with the time stamps of the LIDAR.

c) I had to change the robot base got from the data sheet by multiplying it with 1.9 as the skid type robot needs computation of an effective base.

d) Now I use the values from the encoder for dead reckoning and get a rough estimate of the map. I first average out the wheel encoder counts from the left and the right meter, The following equations are used for the same

   1) $D = \frac{2\pi RN}{Nr}$
      R – radius of the wheels
      N – counts
      Nr – Ticks per revolution.
   2) $Dc = (Dl+Dr)/2$
      Dc- Distance travelled by center
      Dr- Distance travelled by right wheel
      Dl- Distance travelled left wheel.
   3) $\varphi = (Dr-Dl)/base$
      base- effective base of robot.

e) Now a large enough map is built with a resolution of 0.05 m. And a crude map just using the encoder and the LIDAR is built.

Particle Filter

a) I generate 100 particles which are all centered on (0,0,0) being (x,y,φ).
b) I apply motion model for these particles using only the encoder according to the equations given above. Now using the change in x, y and φ, I induce jitter in the system with a Gaussian noise centered on zero and having a standard deviation of 3 times the change in x,y and φ for each particle.

c) Next I compute the weight of each particle. For this I first transform the LIDAR scans in the sensor coordinate to the robot's body coordinates and then to the world coordinates.

   Tsensor = trans([0.15, 0.2, 0.5]) * rotz(0) * roty(0) * rotx(0)
   Timu = rotz(φ)*roty(0)*rotx(0);
   Tpose   = trans([x y 0])
   T = Tpose * Timu * Tsensor;

   where trans corresponds to the homogenous transformation matrix only possessing translation

   $$\text{trans}(x, y, z) = \begin{pmatrix} 1 & 0 & 0 & x \\ 0 & 1 & 0 & y \\ 0 & 0 & 1 & z \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

   rotx(ϴ), roty(ϴ), rotz(ϴ) are homogenous transformation matrices corresponding to pure rotation along x, y and z directions.
   Tsensor is the transformation from the sensor to the body frame.
   Timu is the transformation from the body to the world frame for orientation and
   Tpose corresponds to the transformation from the body to the world frame for translation.

   Then I transform the LIDAR points to the world frame by the following equations
   xs0 = range x cos(ϴ);
   ys0 = range x sin(ϴ);
   where ϴ is the angle of the LIDAR respect to the sensor frame.

   $$X = \begin{pmatrix} xs0 \\ ys0 \\ 0 \\ 1 \end{pmatrix}$$

   $$Y = T \times X$$

   Now Y is the homogenous representation of the LIDAR point in the world frame.

d) To compute the weights I sum up the LIDAR hits with the map log odds value and I only choose the positive values. I normalize the weights of the particles though it may not be necessary.

e) I choose the particle with the most weight and use it to update the map. I increment the points which the LIDAR hit with a log odds update of 30 and decrement the regions the rays passed through by 1. The cells which the rays passed through are computed using the Bresenham's algorithm. I limit the log odds between [5000, -3000].

f) I resample every time the particles. I resample with replacement and pick a particle with a probability proportional to its weight.

g) I repeat the above steps for all the data.

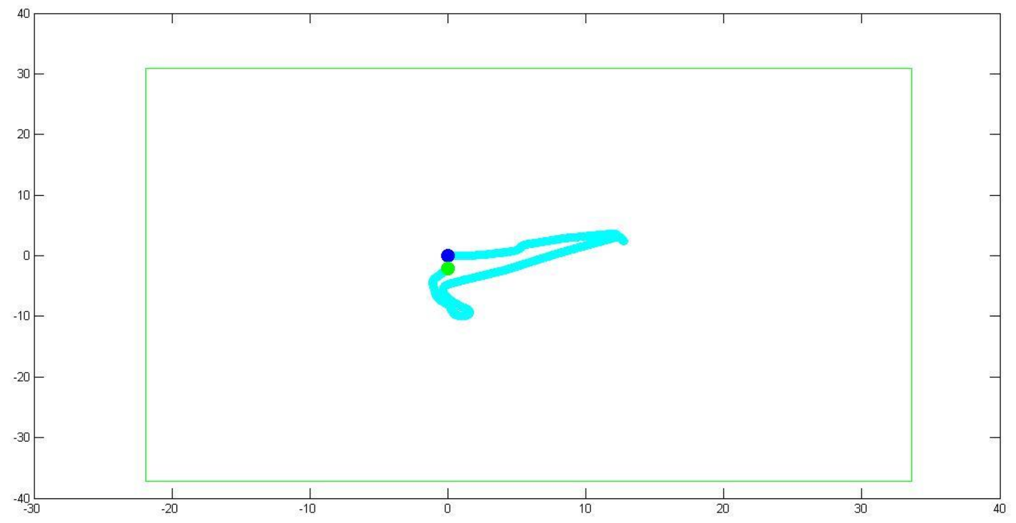
Results

The algorithm works very well both on the training and the test set except the third example in the test set. In the third example the robot slightly deviates but quickly stabilizes back. But the algorithm is computationally expensive and takes about 5-10 minutes for every data set. But it is comparatively faster corresponding to the crude weighting to the particle not using map_correlation which shifts around the particle to find the region of maximum correspondence of the LIDAR hits with previous map. Some of the results on the training and the test set are given below.

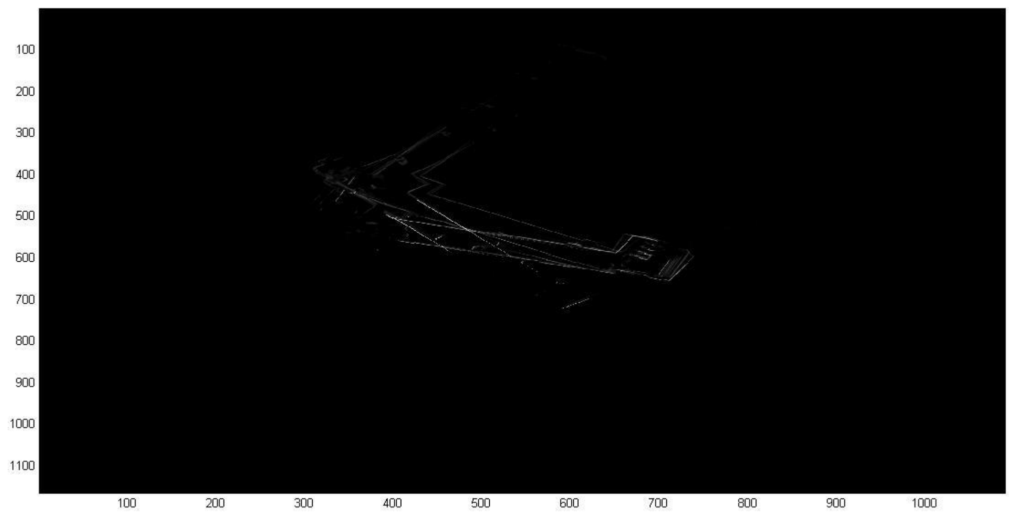
Test Results

Test1

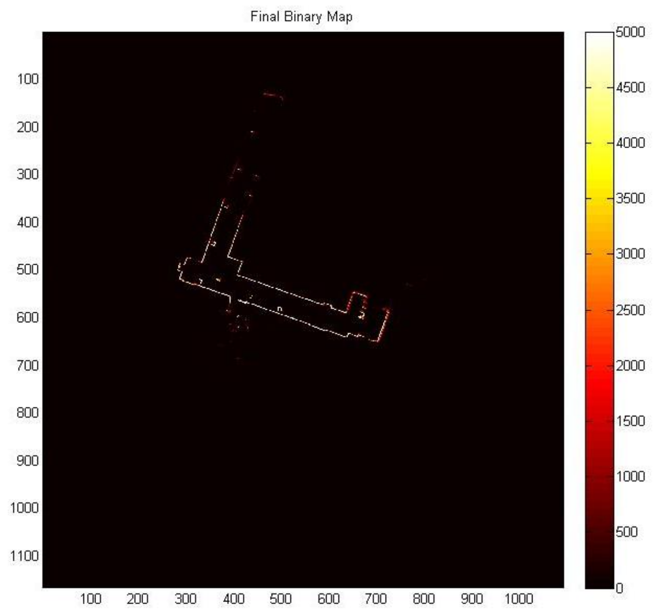
Dead reckoning

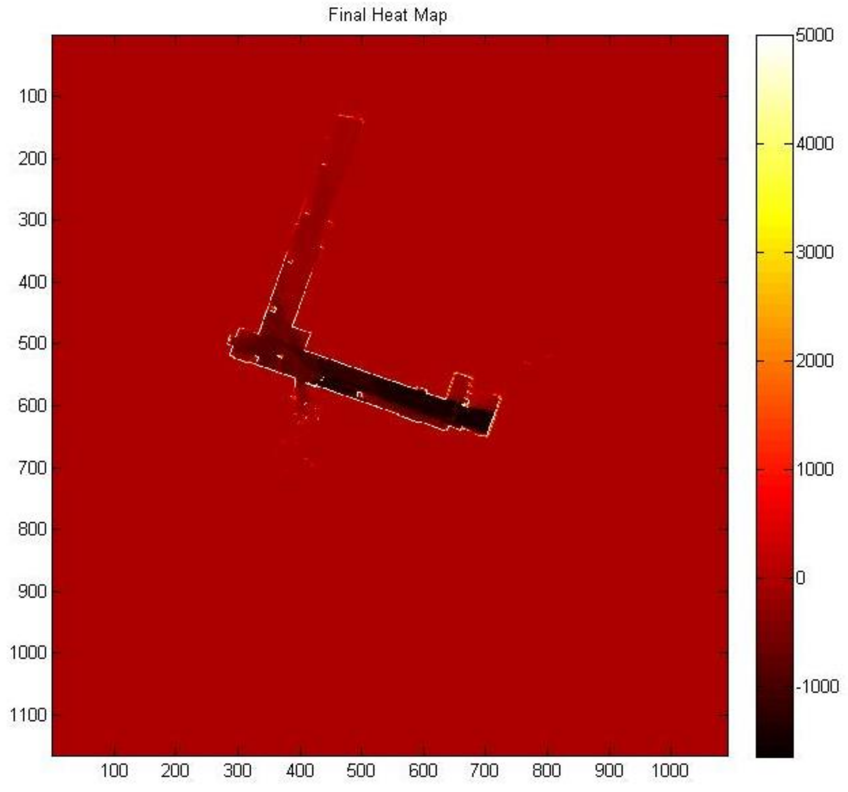


Dead reckoning map



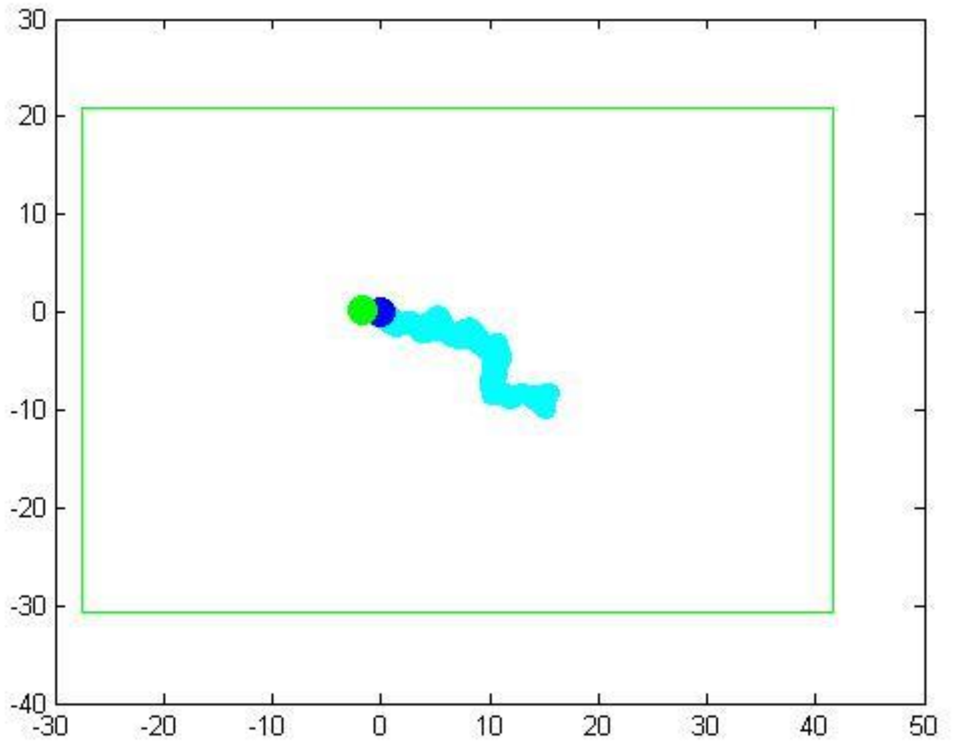
Final Maps

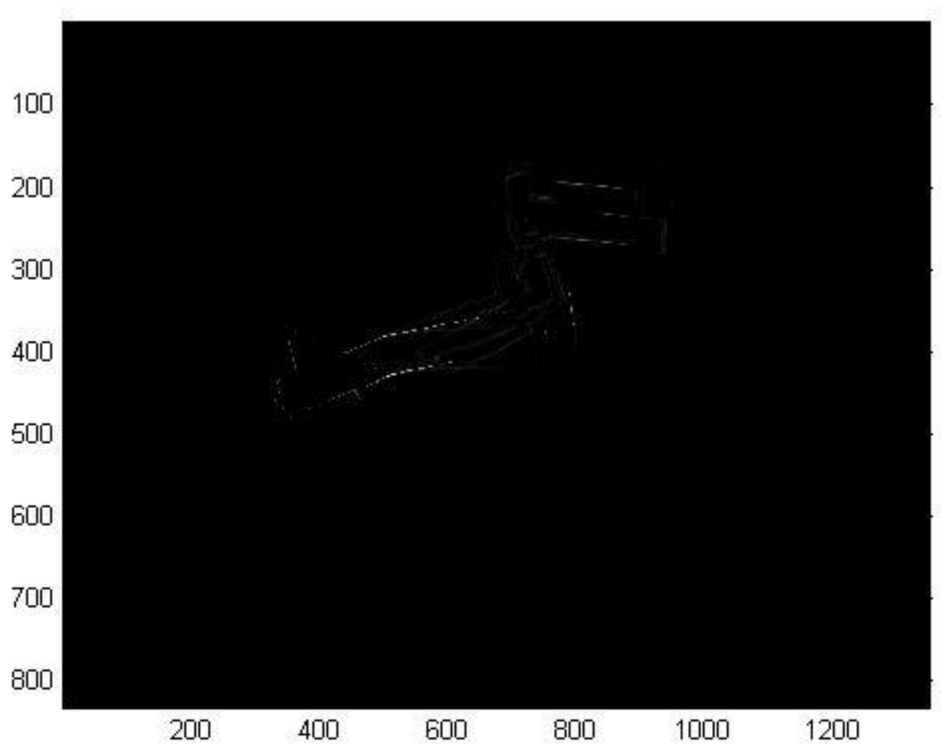

Final Binary Map
100
200
300
400
500
600
700
800
900
1000
1100
100 200 300 400 500 600 700 800 900 1000
5000
4500
4000
3500
3000
2500
2000
1500
1000
500
0


Final Heat Map
100
200
300
400
500
600
700
800
900
1000
1100
100 200 300 400 500 600 700 800 900 1000
5000
4000
3000
2000
1000
0
-1000

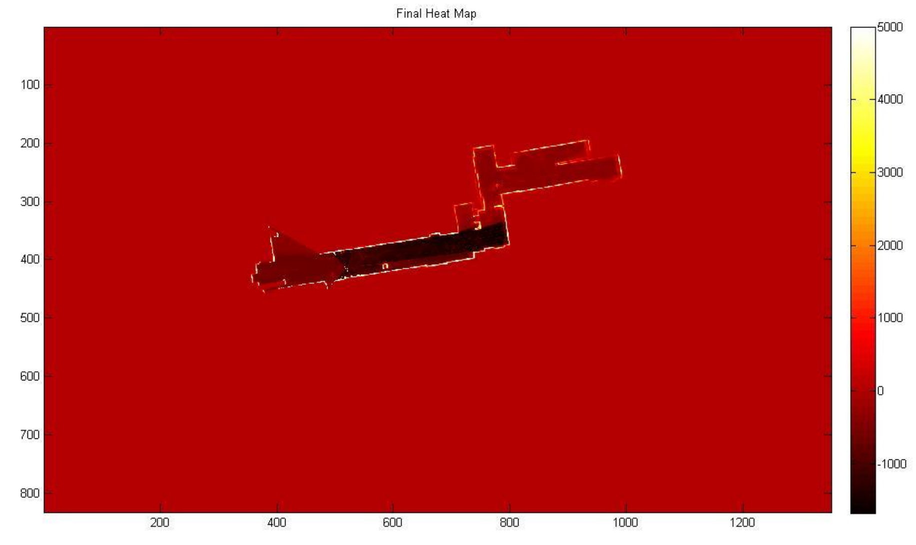
Test 2

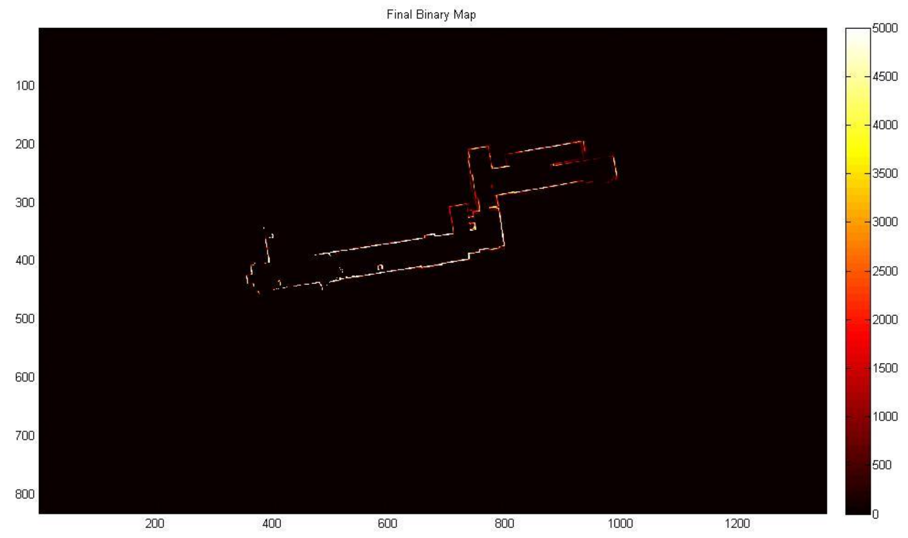
Dead reckoning




100
200
300
400
500
600
700
800
200
400
600
800
1000
1200

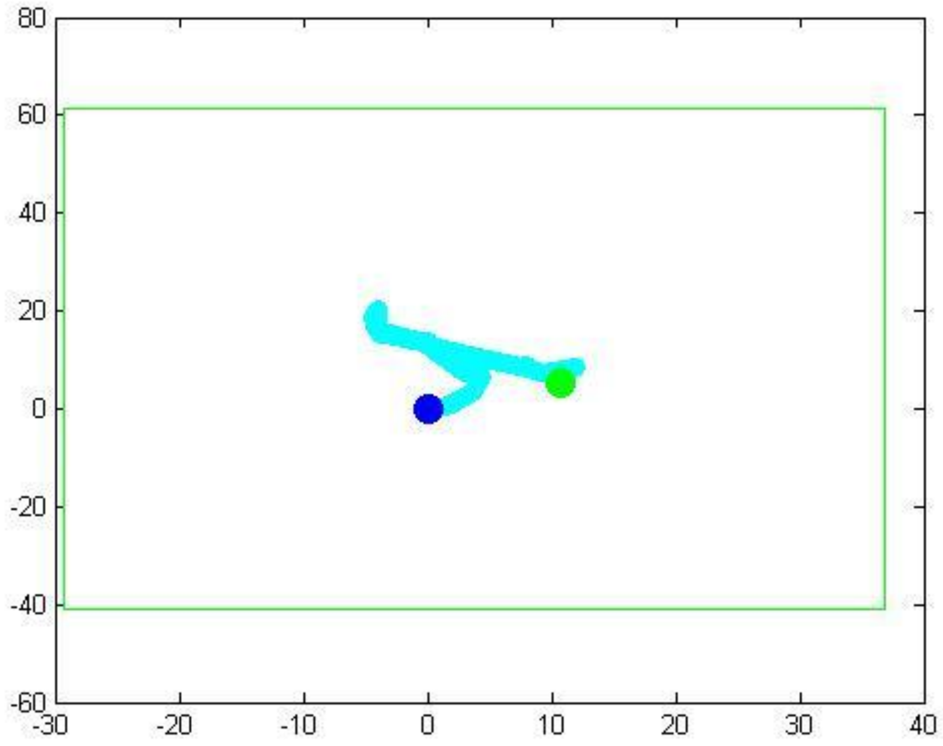
Final Map

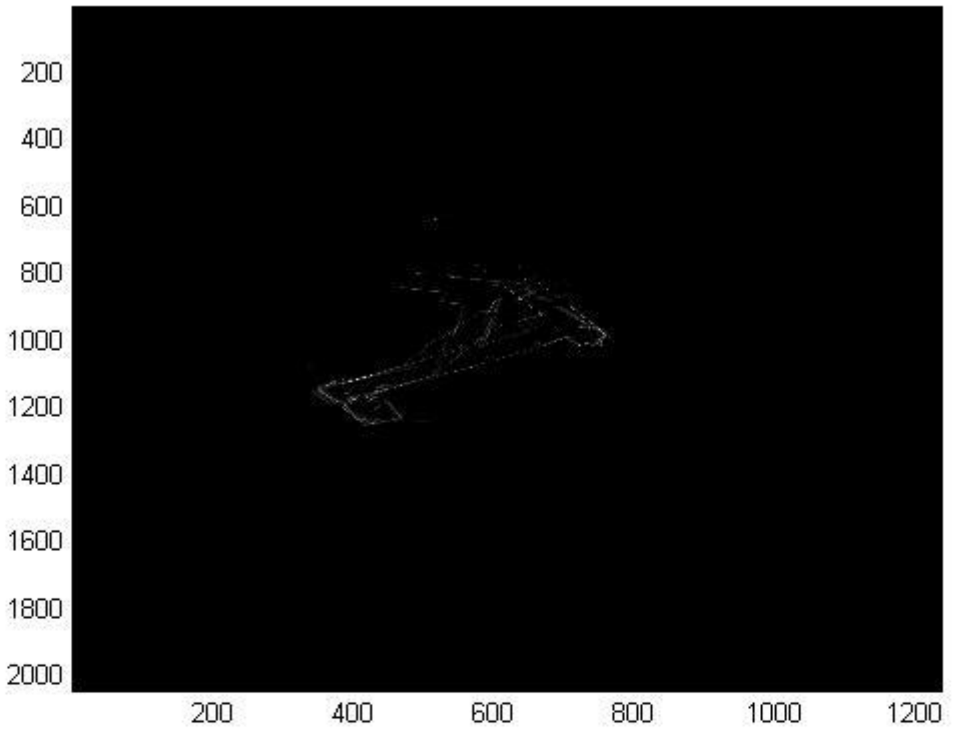

Final Heat Map
100
200
300
400
500
600
700
800
200
400
600
800
1000
1200
5000
4000
3000
2000
1000
0
-1000



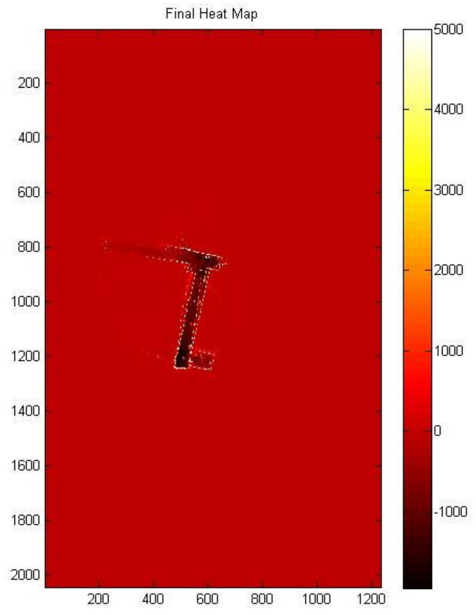
Test 3

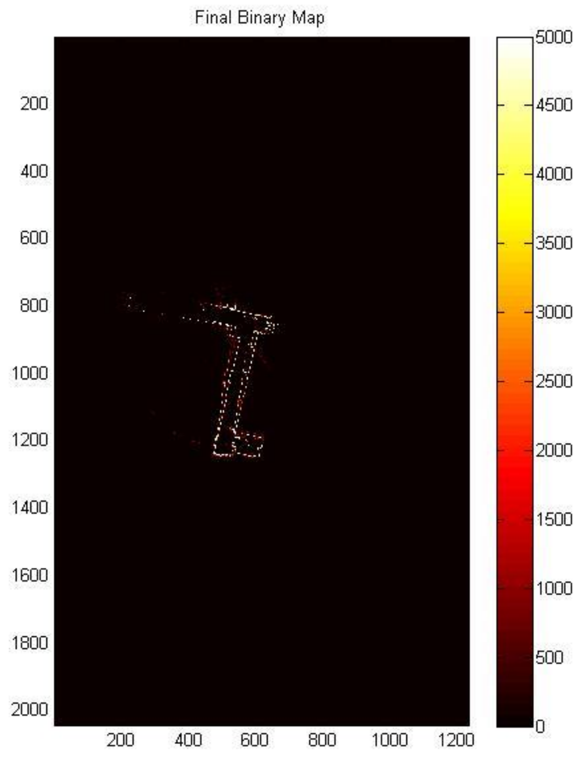
Dead Reckoning



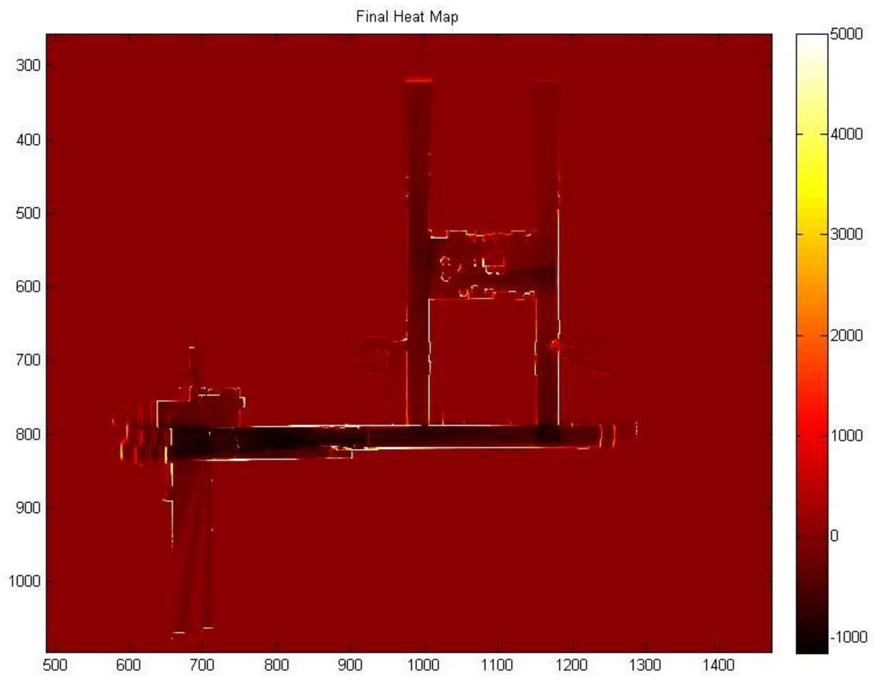

200
400
600
800
1000
1200
1400
1600
1800
2000
200
400
600
800
1000
1200

Final Maps

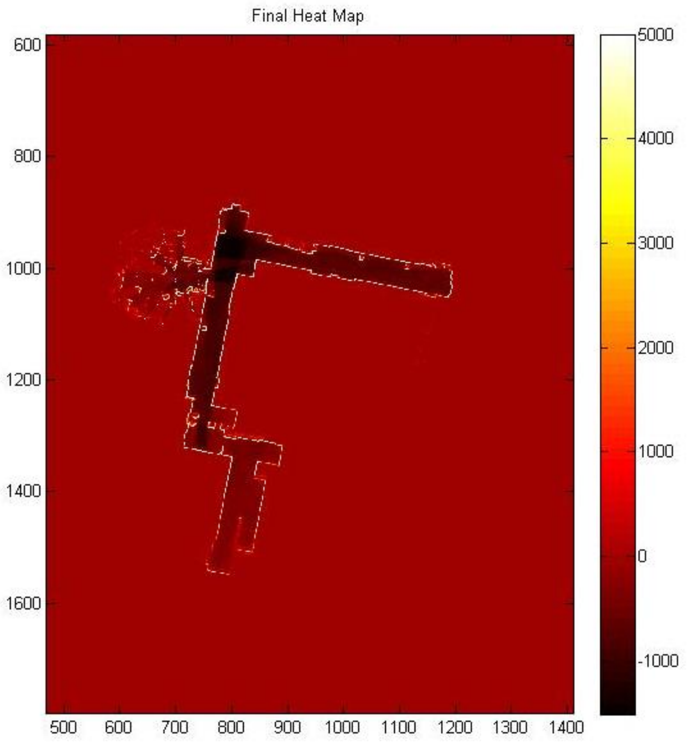

Final Heat Map
200
400
600
800
1000
1200
1400
1600
1800
2000
200
400
600
800
1000
1200
5000
4000
3000
2000
1000
0
-1000



Training Set Result Data number – 23



Training Set Result Data number – 20


Final Heat Map
600
800
1000
1200
1400
1600
500
600
700
800
900
1000
1100
1200
1300
1400
5000
4000
3000
2000
1000
0
-1000