ME 597 Final 2015 Submission

December 13, 2015

Ben V

Q1. A) The batch probabilistic roadmap method is a strategy used to generate a valid path to a destination in a given map. A sampling method samples random points on the map and saves the milestones that are not on lines, inside objects, or too close to an object. These valid milestones are connected to their 20 nearest neighbours unless a collision occurs, as determined from the map. The collision-checking algorithm that will be used is not Bresenham, but instead uses the simple format for edges provided in the map. It is more efficient to only consider the endpoints of the lines instead of each point along the line.

The collision algorithm is as follows, testing the line L1 (x1, x2, y1, y2) against all edges in the map (xi1, xi2, yi1, yi2). Assume no collision if both points in L1 is outside the same boundary of the rectangle formed by points in the edge. For L1 where point is inside area, get the angles formed from (x1, y1) to α = (xi1, yi1) and β = (xi2, yi2). If the angle to (x2, y2) is between the acute difference of α and β, the line must be directed towards the edge. Project the L1 segment onto the normal of the (xi yi) line segment and if it is more than the distance from point (x1, y1) to line segment (xi yi) then there is a collision.

Below shows Figure 1, a plot of the PRM algorithm in action. The total runtime to generate milestones, connect roadmap and find shortest path is 0.234493 seconds. The CPU time for inversion took 0.638367 seconds on the remote desktop Engterm environment. A total of 36 nodes with 391 collision checks were performed.

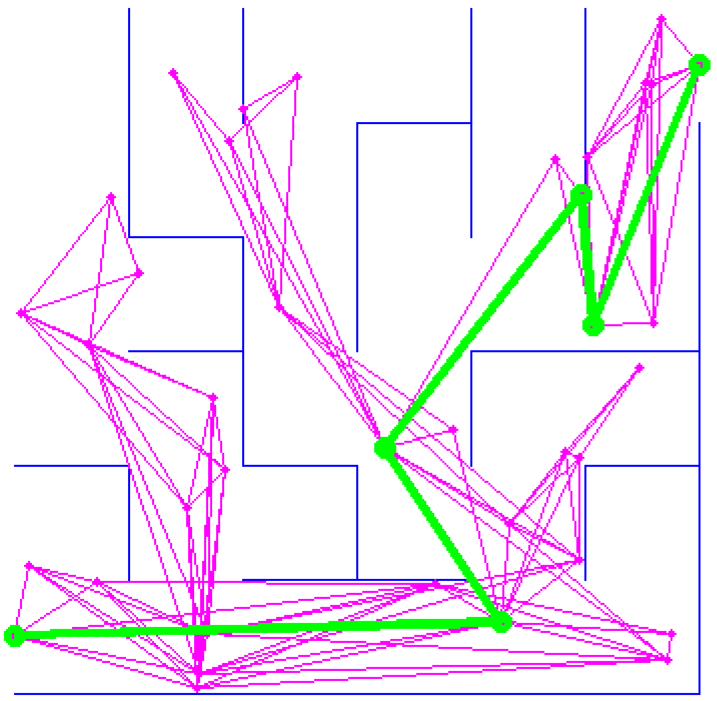

Figure 1: PRM on a 6x6 maze with 72 nodes.

b) The ideal sampling method would be at grid locations mid-edges; the M*N grid constructs edges along 0.5 values, so a set of (M-1)*(N-1) samples at 0.0 integer values evenly distributed will accommodate this. To maintain random sampling, the sampled valued will be rounded before being checked as milestones. This could result in duplicates, and starved locations which can be mitigated by increasing the number of samples generated.

When considering nearest neighbours, the maze is rectilinear, so points that do not lie on the same axis can be considered invalid. This reduces the number of operations per nearest neighbour. The results of the simulation with rounded random rectilinear points is shown in Figure 2. The PRM algorithm did not always find a solution because it does not exhaustively sample each possible point as mentioned in c). The total runtime to generate milestones, connect roadmap and find shortest path is 0.332643 seconds. The CPU time for inversion took 0.537331 seconds. A total of 72 nodes with 408 collision checks were performed.

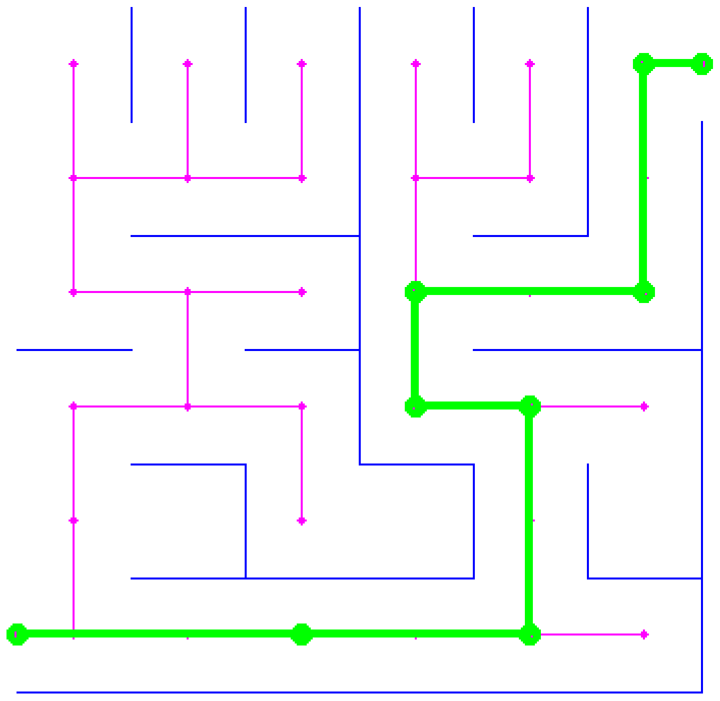

Figure 2: PRM with rounded randomly-sampled rectilinear nodes.

c) A deterministic approach is to select each sample as an integer point from 1 to size of map and distribute them uniformly across the map. The simulation for this is described below and shown in Figure 3. The total runtime to generate milestones, connect roadmap and find shortest path is 0.157297 seconds. The CPU time for inversion took 0.575493 seconds. A total of 42 nodes with 154 collision checks were performed. This result was the optimal result so far with the shortest runtime and fewest collisions checked.

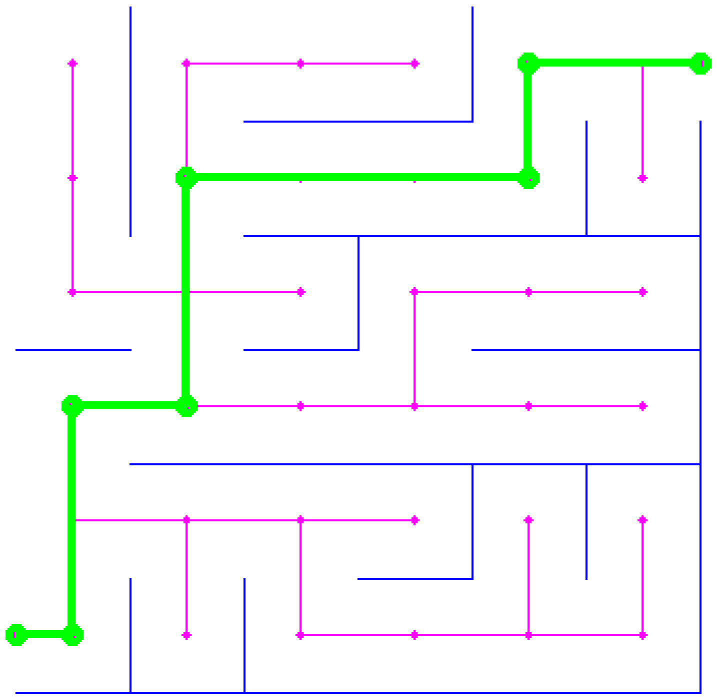

Figure 3: Deterministic sample selection strategy

d) All simulations are run on the remote desktop Engterm server. The first PRM algorithm in part a solved the inverse random matrix in 1.313569 seconds. The time to generate milestones, connect roadmap, and find shortest path is 64.381385 seconds. In total there were 1764 milestones with 16881 line checks performed. A plot of this is shown in Figure 4.

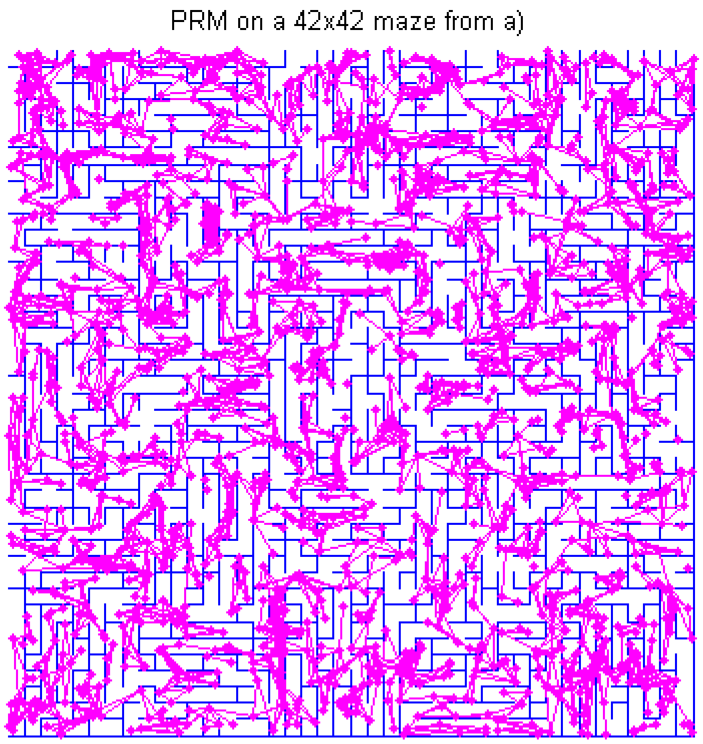

Figure 4: PRM on 42x42 size maze from part a)

The second PRM algorithm in part b solved the inverse random matrix in 0.400742 seconds. The time to generate milestones, connect roadmap, and find shortest path is 71.643174 seconds. It had more milestones, 5776, with 16658 line checks performed. A plot of this is shown in Figure 5.

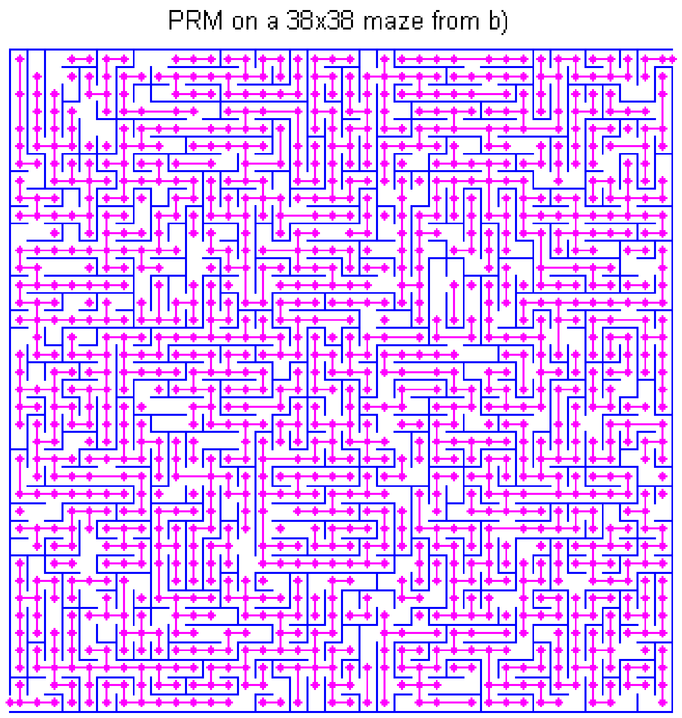

Figure 5: PRM on a 38x38 size maze from b)

The third PRM algorithm in part c solved the inverse random matrix in 0.545974 seconds. The time to generate milestones, connect roadmap, and find shortest path is 63.974353 seconds. In total there were 2550 milestones with 10176 line checks performed. A plot of this is shown in Figure 6.

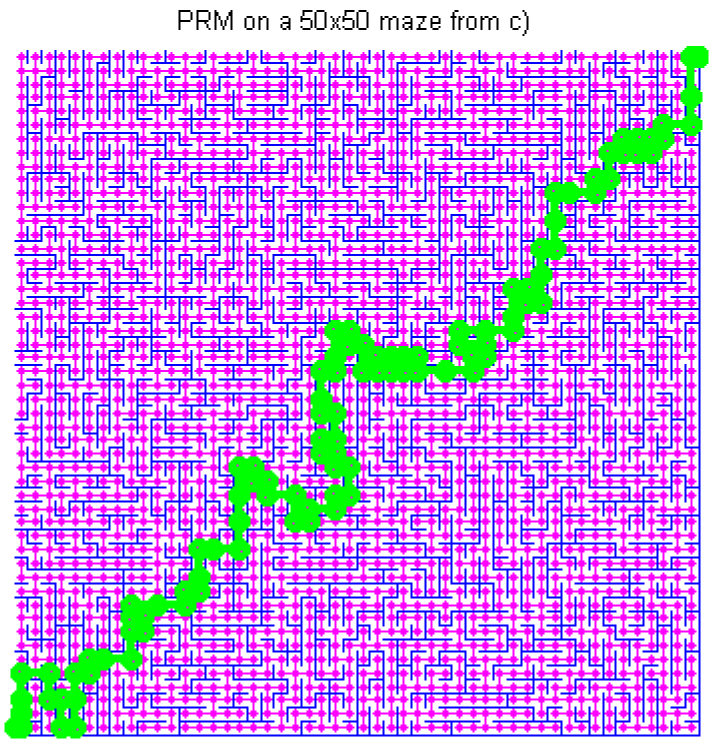

Figure 6: PRM on a 50x50 size maze from c)

These algorithms can all be equally improved by reducing the number of neighbours (currently 20) and eliminating duplicate nodes (in a) and b)). Those algorithms did not solve for the shortest path because there were not enough spread out samples; dead space with no samples caused broken links. This is mitigated in c) where every grid point is checked and there are guaranteed no duplicates.

Q2. A) It is assumed that while the quadcopter is over a bridge square, it cannot see out from it. The position of the features at time t are found within the 20mx15m field of view area relative to the position of the quad. The measurement model reads the position as radius r_m away from the quad at angle Φ from the x axis. This is then returned to an x,y position of the feature with the measurement model disturbance of $\sum_{xx} = \sum_{yy} = 0.1$ and $\sum_{xy} = \sum_{yx} = 0$.

$$r_m = \sqrt{(m_x - x)^2 + (m_y - y)^2}$$
$$\varphi_m = atan2(m_y - y, m_x - x)$$
$$\overline{m_x} = r_m \cos(\varphi_m) + \textstyle\sum_{xx}$$
$$\overline{m_y} = r_m \sin(\varphi_m) + \textstyle\sum_{yy}$$

b) The map contains static features and normal distribution noise, so the ideal SLAM algorithm to implement is EKFSLAM. In the prediction step, the vehicle state and covariance are updated based on the previous state. This prediction is used to update the measurement, and the Kalman gain is found from the linearized measurement model. The Kalman gain is calculated to decide the weight of the next measurement compared to the predicted position. This confidence coefficient is used to update the state and covariance, ensuring a more accurate measurement on subsequent readings.

The 2D linear motion model for the quadcopter is outlined in the equations below. The velocity is iterated with the previous acceleration a(t-1) and the position is updated with the command acceleration a(t). The command signal is generated from the controller. The disturbance is represented as $\sum_{xx} = \sum_{yy} = 0.02$ and $\sum_{vx} = \sum_{vy} = 0.01$.

$$x(t) = x(t-1) + (v_x(t-1) + a_x(t-1) * dt \ + \ \textstyle\sum_{vx}) * dt + \frac{1}{2} a_x(t) dt^2 + \textstyle\sum_{xx}$$
$$y(t) = y(t-1) + (v_y(t-1) + a_y(t-1) * dt \ + \ \textstyle\sum_{vy}) * dt + \frac{1}{2} a_x(t) dt^2 + \textstyle\sum_{yy}$$

c) Integral control is not required. The Kalman gain will help accommodate errors that integral control would usually handle. The Gaussian disturbance is handled well by the EKF as long as the features are distinct from each other, which they are for this simulation. The controller determines a_x and a_y for a given error in position and velocity.

$$a_x(t) = 2 * \frac{d_x - v_x * dt}{\sqrt{dt}} + d_{vx} * dt$$
$$a_y(t) = 2 * (d_y - v_y * dt)/\sqrt{dt} + d_{vy} * dt$$

d) N/A, no time ☹

Q3. a) The motion model is obtained from notes and is shown in the equations below. Signals v(t) and Φ(t) are generated in the controller, then turned into commands for left and right wheels, w_1(t) and w_2(t), where r is the wheel radius and l is the radius of the bot. The disturbances d(t) is generated from the disturbance matrix with covariance 1 on x and y, and covariance of 1 degree on theta. While the variances are large (1m), the map is proportionately larger so a large value must be used to generate visual disturbance.

$$x(t+1) = x(t) + \frac{r\omega_1(t) + r\omega_2(t)}{2} * \cos\big(\theta(t-1)\big) * dt + d_x(t)$$
$$y(t+1) = y(t) + \frac{r\omega_1(t) + r\omega_2(t)}{2} * \sin\big(\theta(t-1)\big) * dt + d_y(t)$$
$$\theta(t+1) = \theta(t) + \frac{r\omega_1(t) - r\omega_2(t)}{2l} * dt + d_\theta(t)$$

The controller finds the distance between the target point and current location at time t, and uses this difference to find the desired heading. The difference between the desired heading and the current heading is scaled by the kP control value and used as input Φ. The robots must not run into each other, so the distance between them is limited to a tolerance of 20px (this is the 2x the scaling of each bot radius in drawbot). Note that part e) specifies a distance of 45m instead of 20m. The control signals are generated as shown below. If dist between the target and tracker at time t is less than tolerance, the input signals are zeroed.

$$v(t) = k_P * v_{Target} * \frac{dist(t) - tol}{tol}$$
$$\varphi(t) = k_P * (\theta_{Des}(t) - \theta(t))$$

b) The plots at 1Hz update rate for the motion model with an input signal of [2t, 2cos(t)] is shown in Figure 7 below. The noise and tolerance values were scaled by 0.1 because the size of the map is significantly smaller.

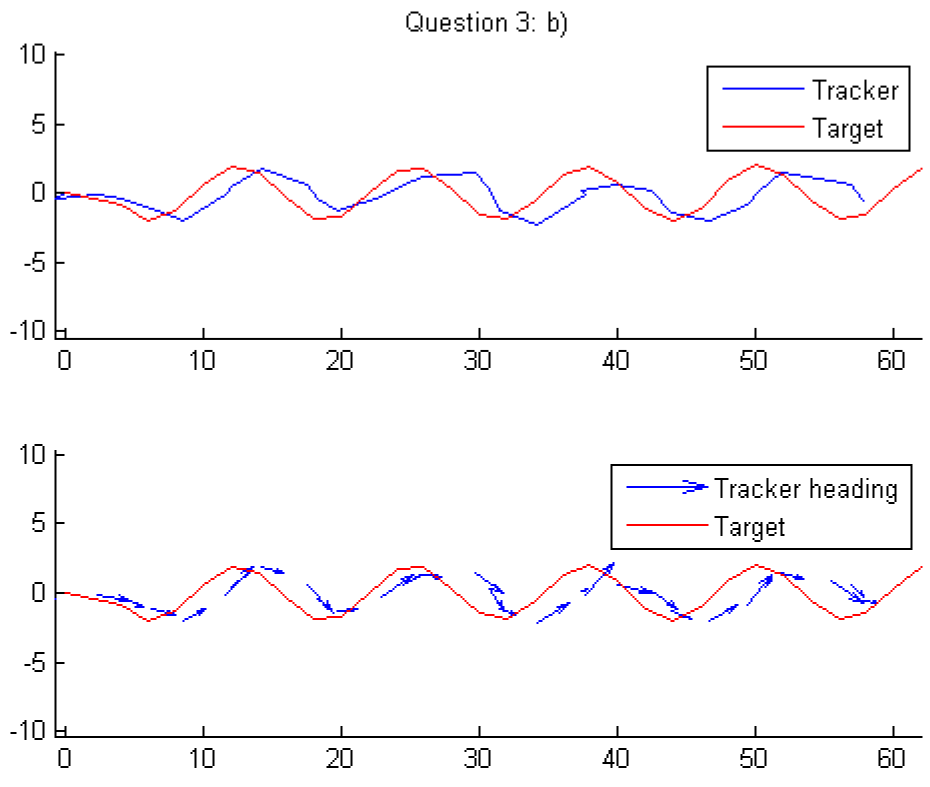

Figure 7: above: Tracker and target positions. Below: Tracker heading/velocity with Target

c) The occupancy grid mapping algorithm tracks the logit value of probability and adds the new logit value for each pixel scanned at each time step minus the initial logit value. The probability of a scanned pixel being occupied is 0.6 and not occupied 0.4; these values were set to match the provided occupancy grid orientation. To perform the algorithm, some assumptions must be made: the environment is static, cells are independent, the vehicle state is known at each time step, and the sensor model is known. The map is generated by reverting the logit values back to probabilities from 0 (empty) to 1 (occupied).

To avoid obstacles, a simple controls check is implemented. If the laser scan in front of the robot reads less than 10m, add -0.75 rad/s to the angular command for the next iteration. This will rotate the robot if it is in front of an obstacle so it will get a new heading and try again. This control is similar to a very simple bug algorithm, but will not be robust at preventing collisions before they happen. In case a robot jumps inside an object, it may rotate indefinitely.

Below, Figure 8 shows the partial occupancy grid data with the path of the tracking robot in white. The semicircular object in front of the robot is the target robot, 45m ahead of the tracker (as per e)). Note the corrective turn as it hits the corner of the first obstacle.

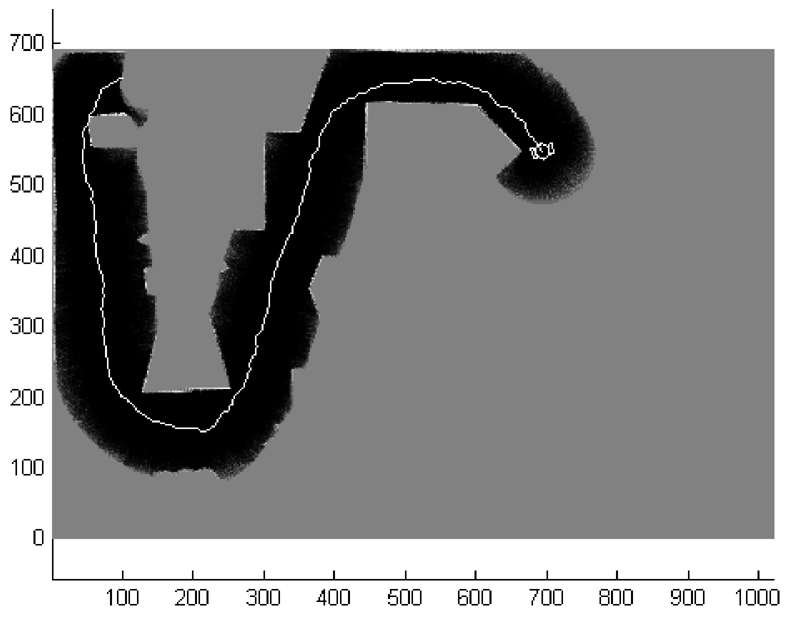

Figure 8: Partial occupancy grid with tracking robot's path at t=330.

d) The target robot is now added to the binary map each iteration at its current position. It was not determined how to find the position of the target robot using the changing probabilities in the occupancy grid and laser scan data. The probable position of the target robot must be determined and then used to direct the tracking robot by feeding the probable point into the controller of the tracker. This will generate the linear/angular velocity required to move the robot in the desired direction. This implementation will use the exact position of the target.

e) The map of the complete solution for the tracker robot is shown in Figure 9 to Figure 12 for 5, 20, 100 and 200 seconds, respectively. The tracker bot remains 45m behind the target. The complete track up to this time step is shown as well for the tracker (in green) and the target (red). Areas where the target is detected are subsequently emptied on the next iteration of the map update because the target is moving. At closer ranges (ie. 20m instead of 45m) the circular shape of the target robot is clearly visible in front of the tracker in the occupancy grid.

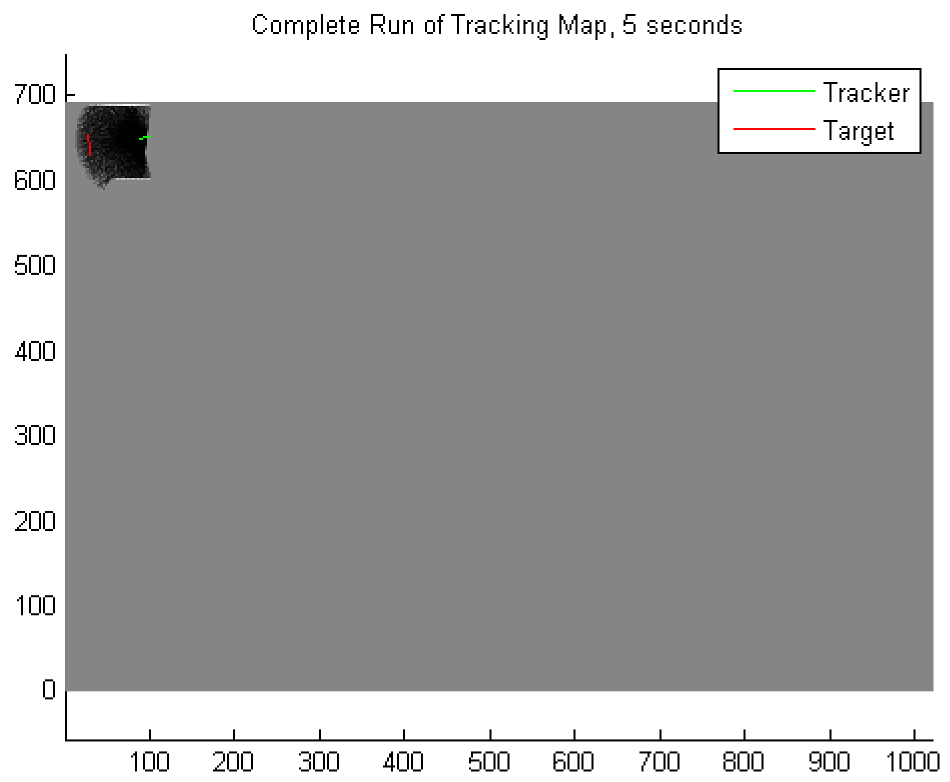

Figure 9: Partial occupancy grid for complete run for 5 seconds, following 45m behind.

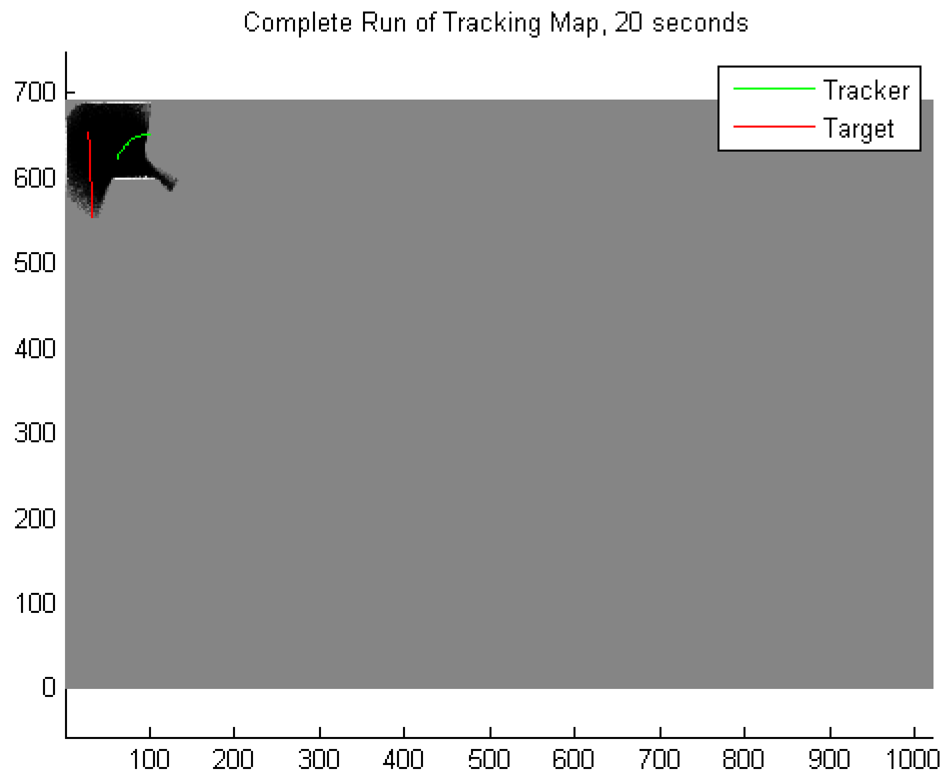

Figure 10: Partial occupancy grid for complete run for 20 seconds, following 45m behind.

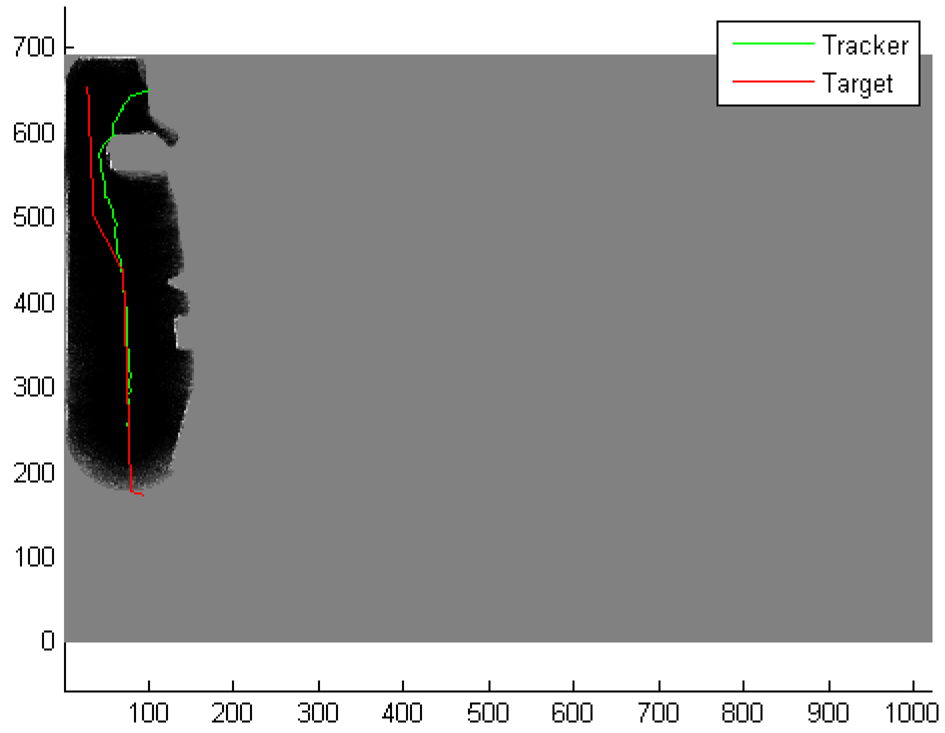

Figure 11: Partial occupancy grid for complete run for 100 seconds, following 45m behind.

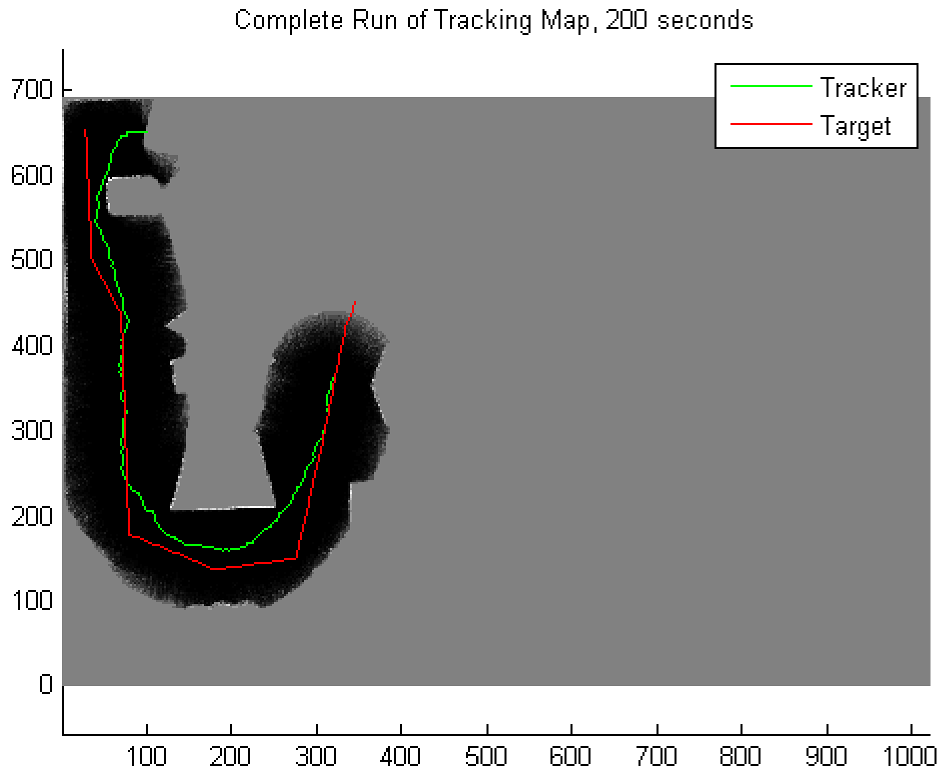

Figure 12: Partial occupancy grid for complete run for 200 seconds, following 45m behind.