

UNIVERSITY OF
Waterloo



Department of Mechanical and Mechatronics Engineering

MTE 544

HW #3

A Report Prepared For:

Prof. Hadi Adibi

MTE 544

Autonomous Vehicles

Prepared By:

Kapilan Satkunanathan - 20694418

December 3, 2020

Question 1

Implementation and types of heuristics

```
% Three types of heuristics

% 1- Euclidean Distance
% dist=sqrt((x1-x2)^2 + (y1-y2)^2);

% 2- Manhattan Distance
% dist=(abs(x1-x2) + abs(y1-y2));

% 3 - Greedy Algorithm
% dist=sqrt((x1-x2)^2 + (y1-y2)^2) - ((x1) + (y1));

% Dijkstras Algorithm
% dist = 0;
```

Figure 1 The Three Heuristic functions I implemented

As seen in Figure 1, the three heuristic functions that I implemented were the Euclidean Distance, Manhattan Distance and Greedy Algorithm.

The Euclidean distance heuristic takes the diagonal distance from the current location to the goal. The heuristic function is:

$$h(n) = \sqrt{(goal_x - curr_x)^2 + (goal_y - curr_y)^2}$$

The Manhattan Distance heuristic takes the sum of the lateral and longitude distance from the current location to the goal. The heuristic function is:

$$h(n) = abs(goal_x - curr_x) + abs(goal_y - curr_y)$$

The Greedy heuristic essentially subtract the cost function $g(n)$ which in this case is the sum of the lateral and longitude distance from the start in addition to a heuristic in this case Euclidean Distance. Which reduces the cost function to be: $f(n) = h(n)$ and the heuristic function is:

$$h(n) = \sqrt{(goal_x - curr_x)^2 + (goal_y - curr_y)^2} - (goal_x + goal_y)$$

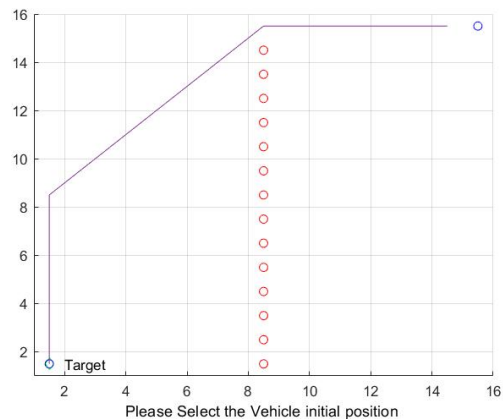
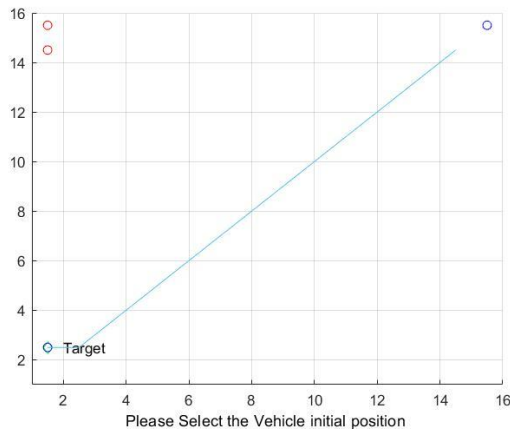
The Dijkstra Algorithm is a blind search algorithm where the heuristic function is just 0.

$$h(n) = 0$$

Path taken no obstacles vs obstacles

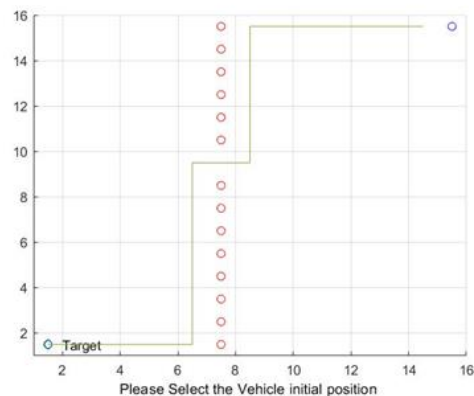
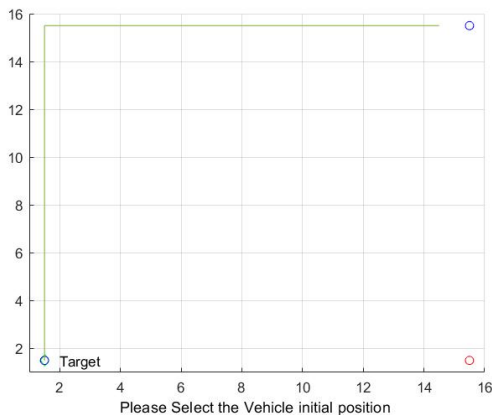
The following images show the path each heuristic and Dijkstra takes with no obstacles versus obstacles.

Euclidean distance



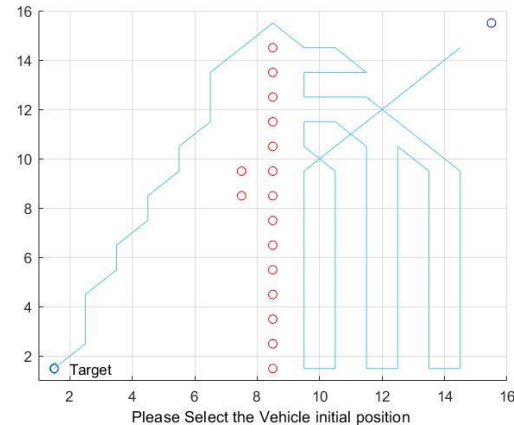
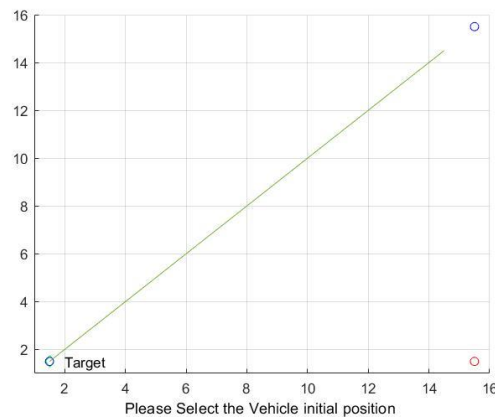
Euclidean finds the most optimal path possible by going straight diagonally when there are no obstacles. Even when faced with obstacles it finds the most optimal path, as it can detect the gap in the wall and pass through it with precision and then proceed to advance to the goal the quickest. Not only does Euclidean find the optimal solution regardless of obstacles it does it fast as well.

Manhattan distance



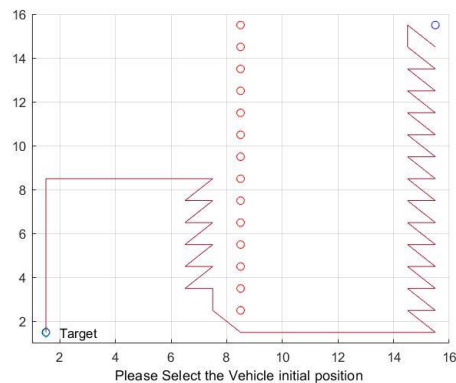
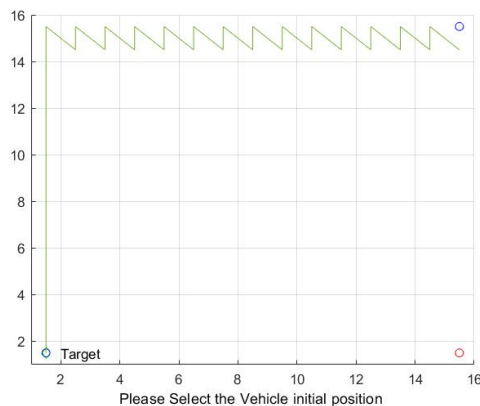
Manhattan excluding diagonal movements finds the most optimal path possible in both cases. It is basically the same as Euclidean excluding diagonal movement as it only considers lateral and longitude distance in its heuristic function.

Greedy



Greedy finds the most optimal path without obstacles but becomes severely lost when faced with obstacles and moves according to what it thinks is the best move without the prior cost. It does eventually find the target but is highly inefficient. Thus, Greedy is inefficient when faced with obstacles but is much less complex in terms of complexity as it doesn't consider cost and does find a path.

Dijkstra



Dijkstra's behavior is that it has "noise" and doesn't have a smooth trajectory. As seen in the figures it alternates directions and moves forward in a noisy manner. It also doesn't take the most optimal path but is less "lost" compared to greedy algorithm when faced with obstacles.

Question 2

Part A

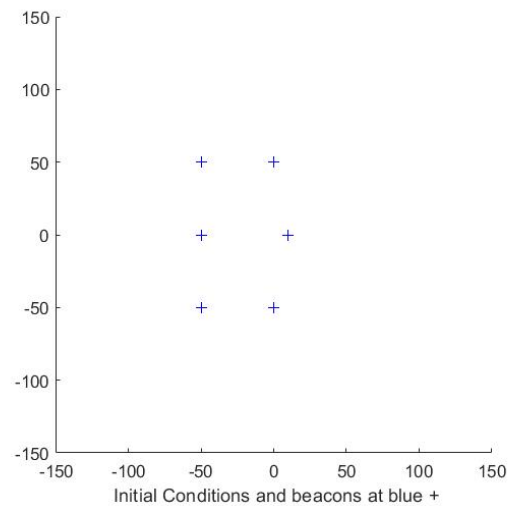


Figure 2 Map of beacons

```
% Lets put the beacons at desired positions ( need to match the number  
% defined)!!!! if nor some of then cannot be scanned!!  
LandFeatures(:, :, 1)=[0 -50]';  
LandFeatures(:, :, 2)=[0 50]';  
LandFeatures(:, :, 3)=[10 0]';  
LandFeatures(:, :, 4)=[-50 0]';  
LandFeatures(:, :, 5)=[-50 -50]';  
LandFeatures(:, :, 6)=[-50 50]';
```

Figure 3 Coordinates of beacons

```
% Initial position of the vehicle  
xVehicleTrue = [ 0 0 0]'; % start position (x, y, theta)  
% xVehicleTrue = [ 100 100 0]'; % start position (x, y, theta)
```


Figure 4 starting positions

As seen from the above figures, it lays out where the 6 beacons are located through visualizations and providing their coordinates. The world map is 300 by 300. Also, we will use (0,0,0) as the starting point for the robot and one simulation where the robot starts at (100,100,0).

Part B

```
% Set the initial conditions of the filter
xEst = [xVehicleTrue]; % already an column vector
PEst = diag([1 1 0.01]); % start with an medium covar ( Attention EKF need an good initialization)
% PEst = diag([10 10 0.1]); % start with an medium covar ( Attention EKF need an good initialization)
% PEst = diag([.1 .1 0.001]); % start with an medium covar ( Attention EKF need an good initialization)
```

Figure 5 The different covariance matrix conditions used for filter



Analytical Approach

- The analytical form of the covariance matrix for the new feature:

$$P_1 = \begin{bmatrix} P_{v,v}^0 & P_{v,m}^0 & P_{v,v}^0 \xi^T \\ P_{m,v}^0 & P_{m,m}^0 & P_{m,v}^0 \xi^T \\ \xi P_{v,v}^0 & \xi P_{v,m}^0 & \xi P_{v,v}^0 \xi^T + B \end{bmatrix}$$

$$\xi = \partial h / \partial (x_v, y_v, \varphi_v) = \begin{bmatrix} 1 & 0 & \cos(\varphi_v + \alpha - \pi / 2) \\ 0 & 1 & \sin(\varphi_v + \alpha - \pi / 2) \end{bmatrix}_{(\varphi_v, r, \alpha) = (\varphi_v(k/k), r(k), \alpha(k))}$$

$$B = \eta R \eta$$

$$\eta = (\partial h(X_v, Z) / \partial z)_{x_v(k,k), z(k)}$$

$$\eta = (\partial h(X_v, Z) / \partial z) = \partial h / \partial (r, \alpha) = \begin{bmatrix} \cos(\varphi_v + \alpha - \pi / 2) & -r \sin(\varphi_v + \alpha - \pi / 2) \\ \sin(\varphi_v + \alpha - \pi / 2) & r \cos(\varphi_v + \alpha - \pi / 2) \end{bmatrix}$$

Eduardo Nebot
SLAM

Figure 6 Covariance Matrix [1]

The PEst variable seen in Figure 5, is the covariance matrix seen in Figure 6. The three numbers initialize the PEst diagonal matrix, where the first one for x, second one for y and last for theta. Covariance is the joint measurability of two variables and is represented as an area of where the robot thinks the beacon is with uncertainty. The following figures are the results of what happens when we use large covariance, medium covariance and small covariance.

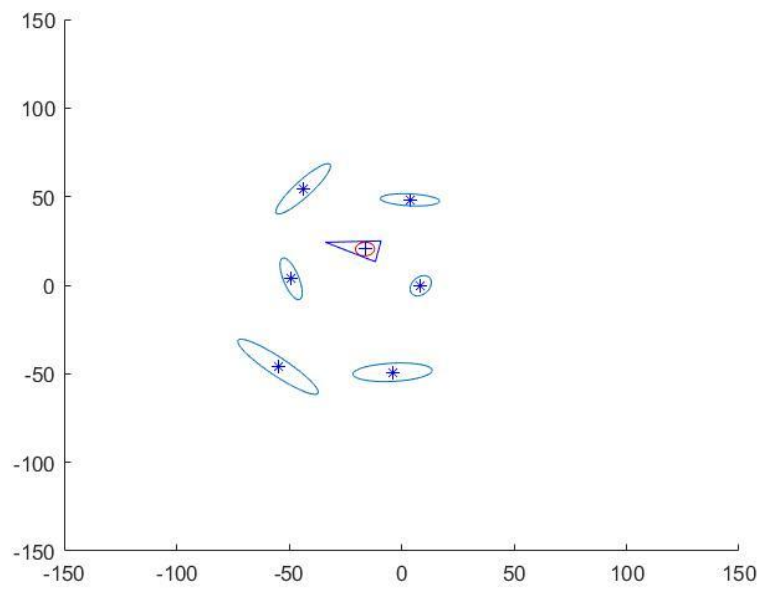


Figure 7 results when covariance at $[1 \ 1 \ 0.01]$

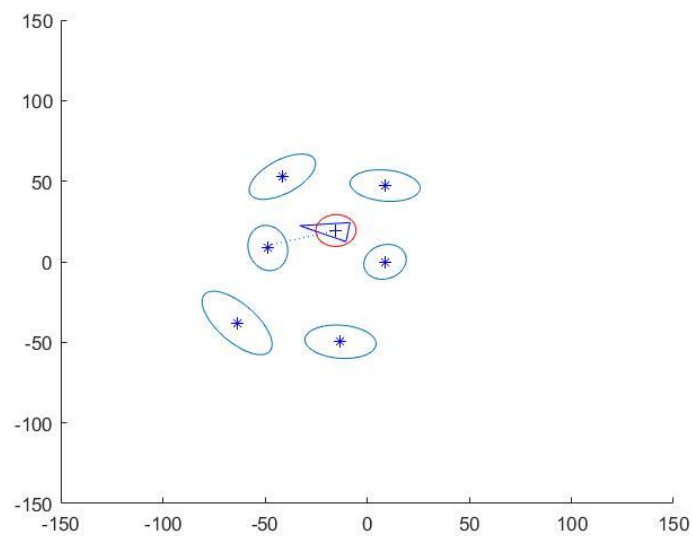


Figure 8 results when covariance at $[10 \ 10 \ 0.1]$

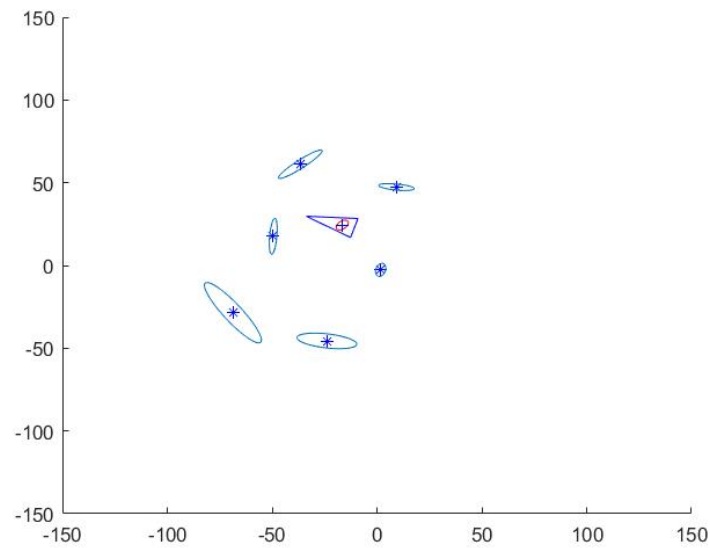


Figure 9 results when covariance at [0.1 0.1 0.001]

As seen in the above figures, the smaller the covariance the more small/accurate the predication of the beacon's locations become. However, if these were obstacles it might beneficial to use a large covariance as a safety measure to ensure the robot doesn't hit the beacon.

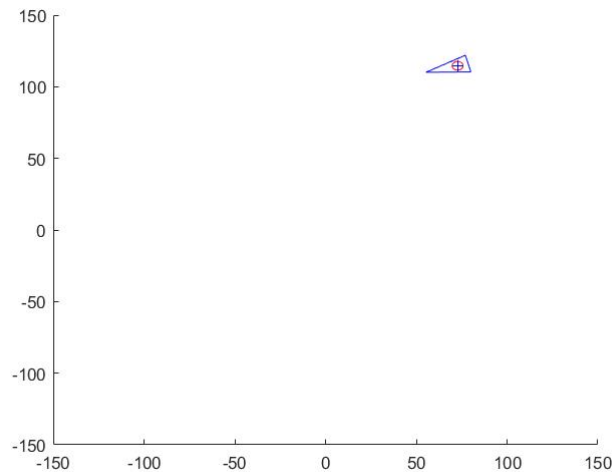


Figure 10 results when covariance at [1 1 0.01] and position at 100,100,0

When the robot location is put far away from the beacons, it doesn't detect any beacons, which shows that location of the robot does matter and the range of the sensor used. Even with good covariances it doesn't matter.

References

- [1] H. Durrant-Whyte, "Australian Centre for Field Robotics," The University of Sydney, Sydney.