Mobile Robotics Homework 9

Syed Hamza Azeem - sa02515
Syed Sameer Nadeem - sn02902

December 22, 2019

1 Introduction

1.1 Background

The A* algorithm is a graph traversal and path searching algorithm which is often useful in related problems owing to its optimality, completeness and efficiency. It was first proposed at the Stanford Research Institute in 1968 by Peter Hart, Nils Nilsson and Bertram Raphael. It is not the most efficient algorithm out there due to its space complexity which can burden memory resources due to the need for storing generated nodes in memory. Nevertheless, the A* algorithm still offers a useful tool for mobile robots in traversing their environments.

Without delving into too much detail, the A* algorithm uses a heurisitc function in the form of a cost estimate of the path from the current node where the object of interest is located to the goal location. By minimizing the cost estimate, the algorithm can selectively choose the nodes which would enable it to go to its goal in the most efficient route possible that it can determine. The heuristic function can be Euclidean or Mahattan distance.

1.2 Task

The goal is to implement the A* path traversal algorithm by considering an increasing number of neighbors of the central node. Increasing the number of neighbors of consideration of the current node enables more angles to be accommodated in the traversed path leading to potentially shorter paths and smoother turns. The task is to compare the performances of the A* algorithm with increasing numbers of neighbors by considering the time taken for the algorithm to run with different neighbor sizes.

The tic/toc functions of MATLAB are used to determine the running time for the algorithm. Note that the times obtained in the esults depend on where the tic and toc functions are placed in the code. In the case of this assignment, they are placed at the beginning and ending of the map generation and route calculation code segments. For each neighbor size, the algorithm is run 100 times to obtain a hundred running times in order to have a larger dataset for analysis.

2 Results

The following plots demonstrate the resulting paths and running times over 100 iterations for different neighbor sizes of the A* algorithm. These results are discussed in the Analysis section.

2.1 No. of Neighbors = 4

The number of neighbors is set to 4 with the following matrix configuration:

$$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

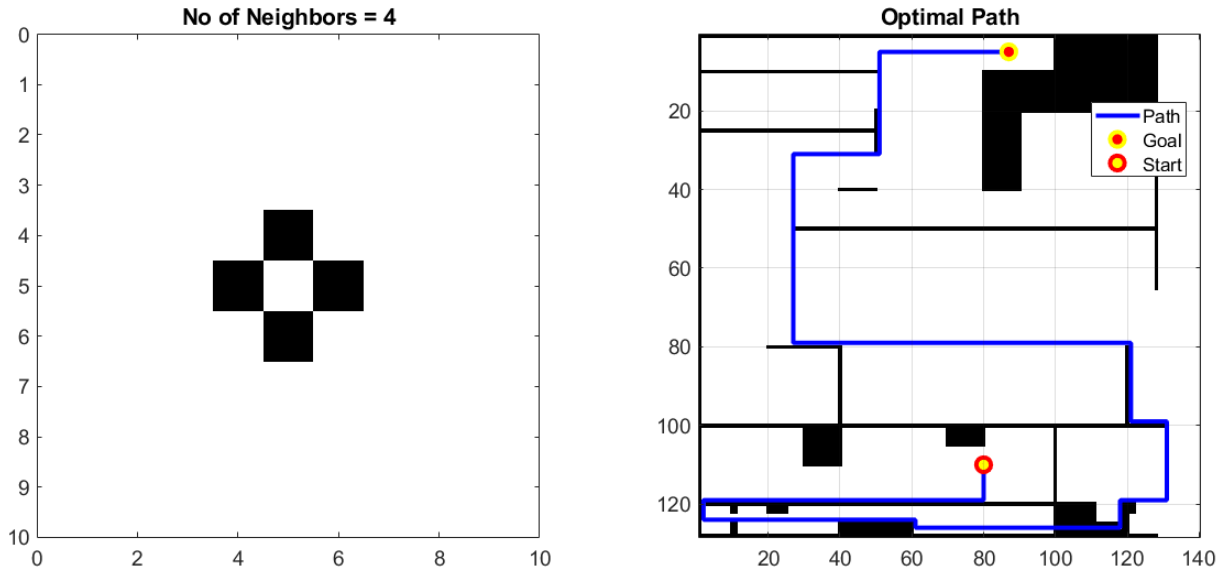

Figure 1: Optimal Path with No. of Neighbors = 4

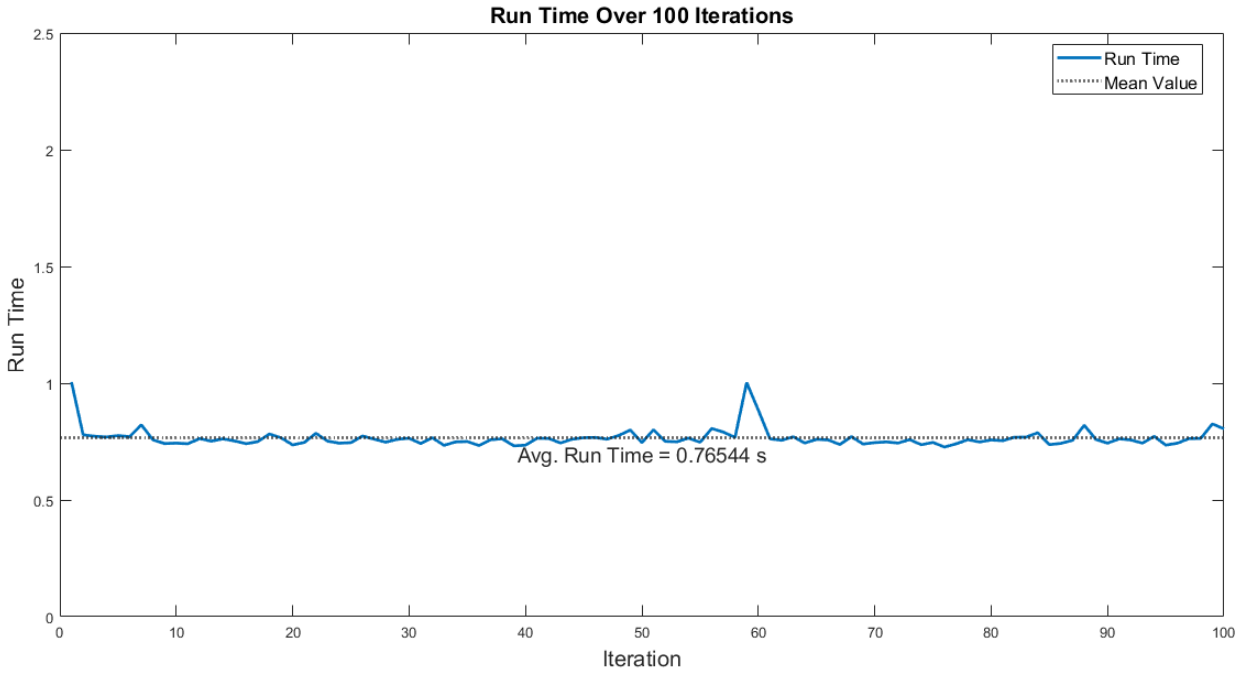

Figure 2: Algorithm Run Time with No. of Neighbors = 4

2.2 No. of Neighbors = 8

The number of neighbors is set to 8 with the following matrix configuration:

$$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

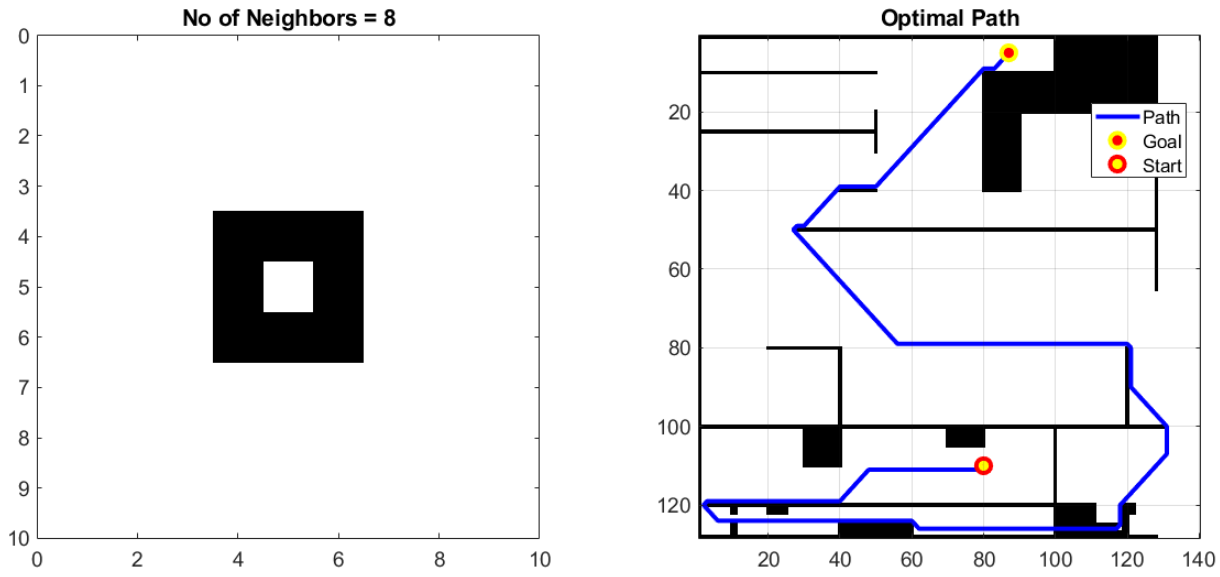

Figure 3: Optimal Path with No. of Neighbors = 8

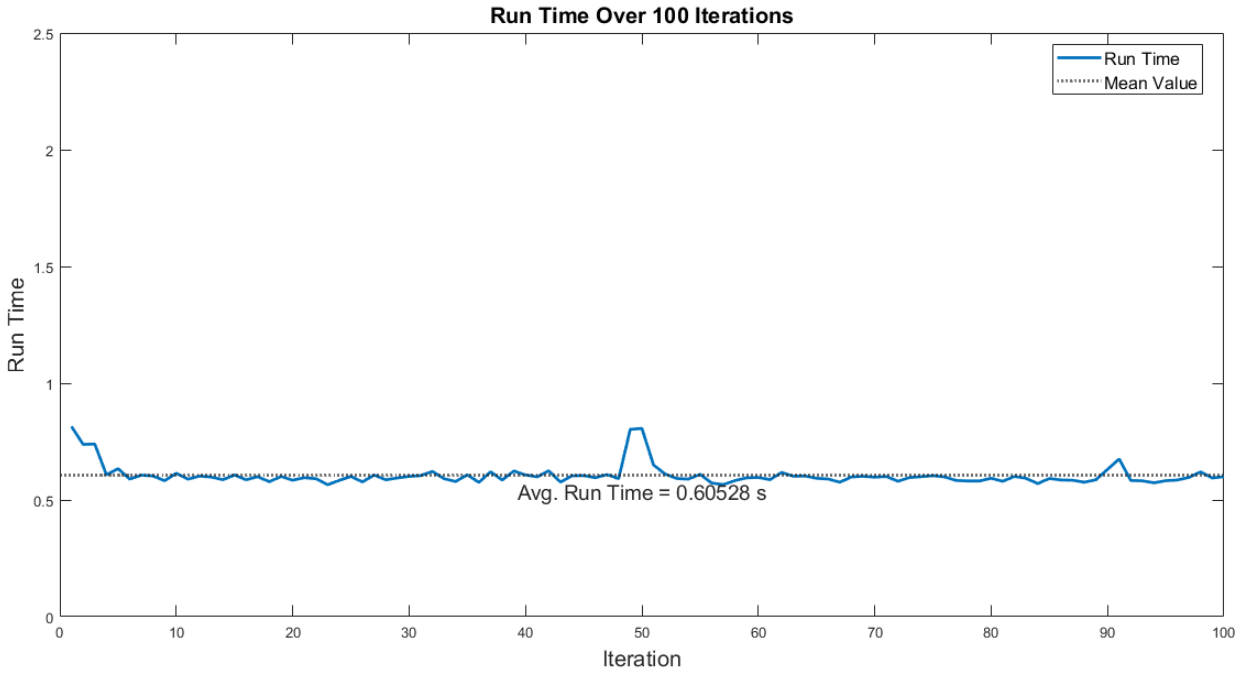

Figure 4: Algorithm Run Time with No. of Neighbors = 8

2.3 No. of Neighbors = 16

The number of neighbors is set to 16 with the following matrix configuration:

$$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

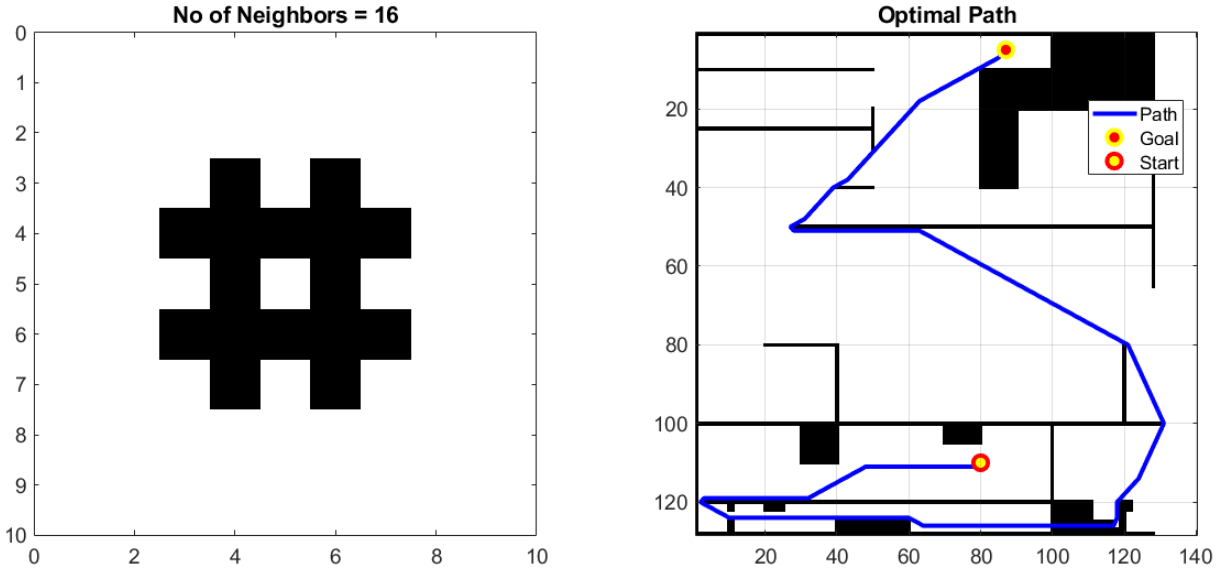

Figure 5: Optimal Path with No. of Neighbors = 16

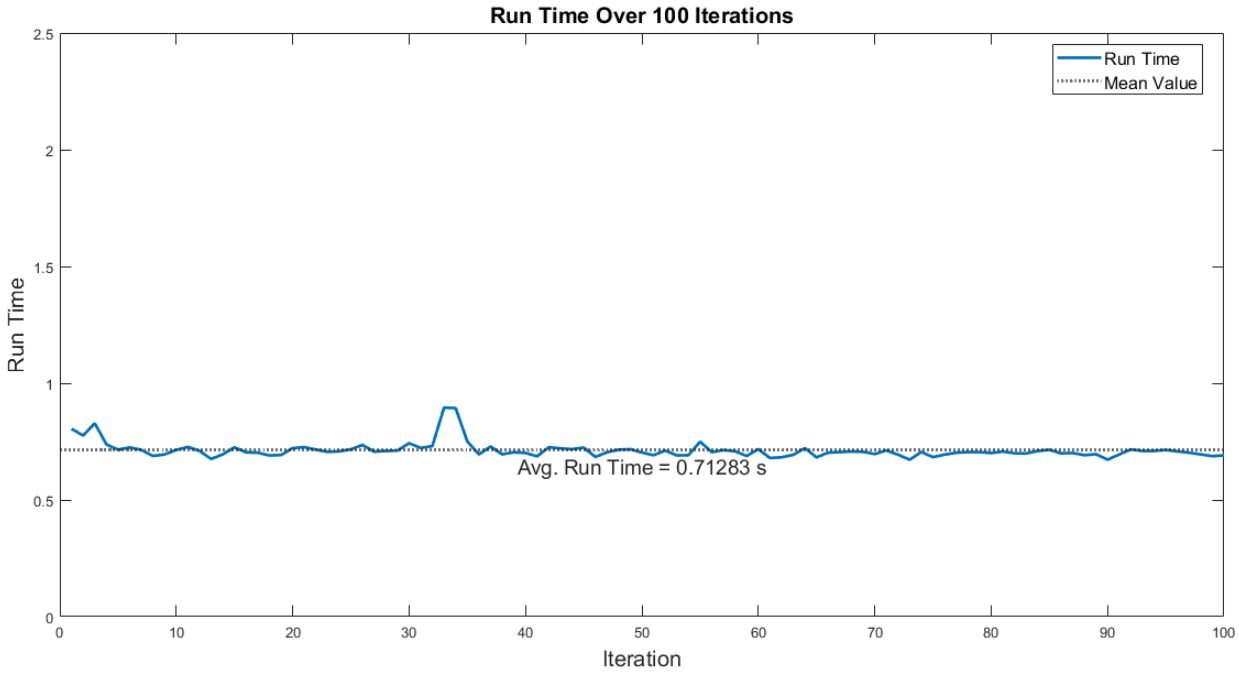

Figure 6: Algorithm Run Time with No. of Neighbors = 16

2.4 No. of Neighbors = 40

The number of neighbors is set to 40 with the following matrix configuration:

$$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

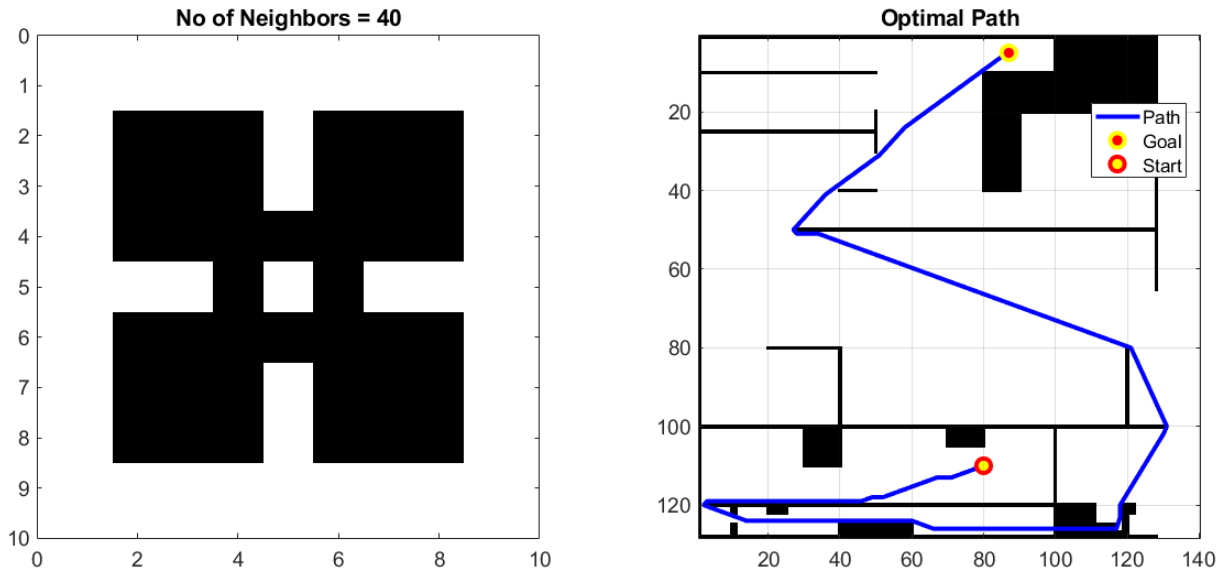

Figure 7: Optimal Path with No. of Neighbors = 40

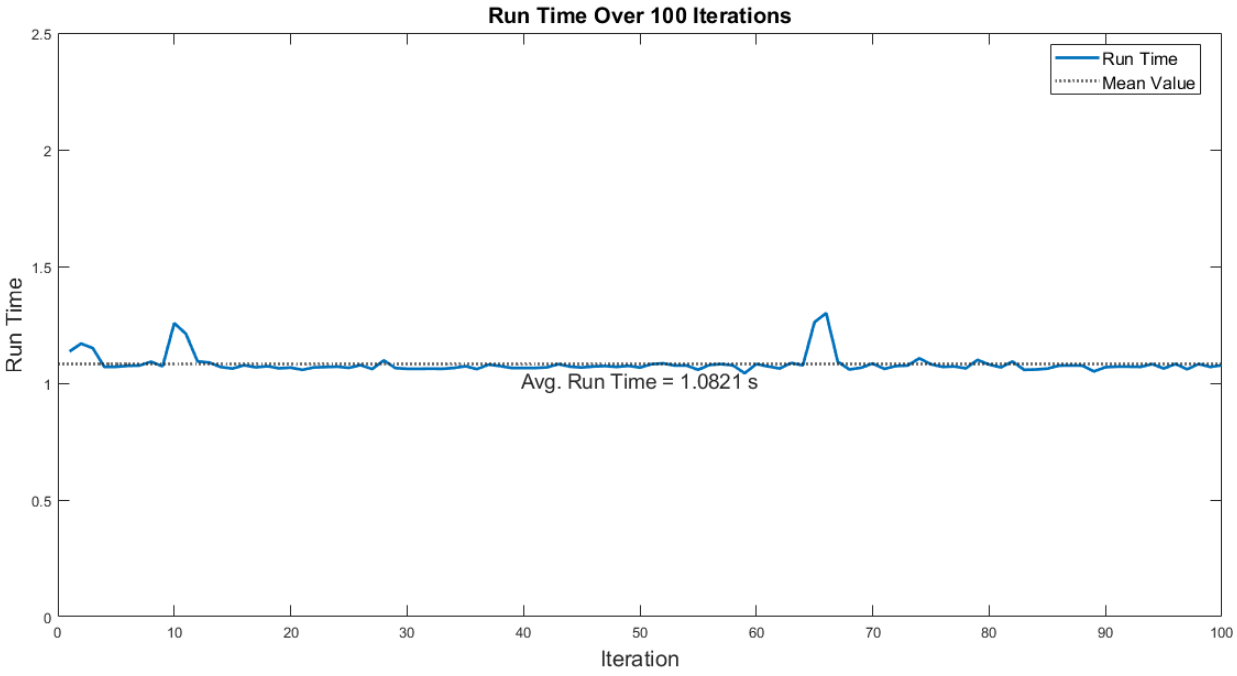

Figure 8: Algorithm Run Time with No. of Neighbors = 40

2.5 No. of Neighbors = 68

The number of neighbors is set to 68 with the following matrix configuration:

$$\begin{bmatrix} 1 & 1 & 1 & 1 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 1 & 1 & 1 & 1 \end{bmatrix}$$

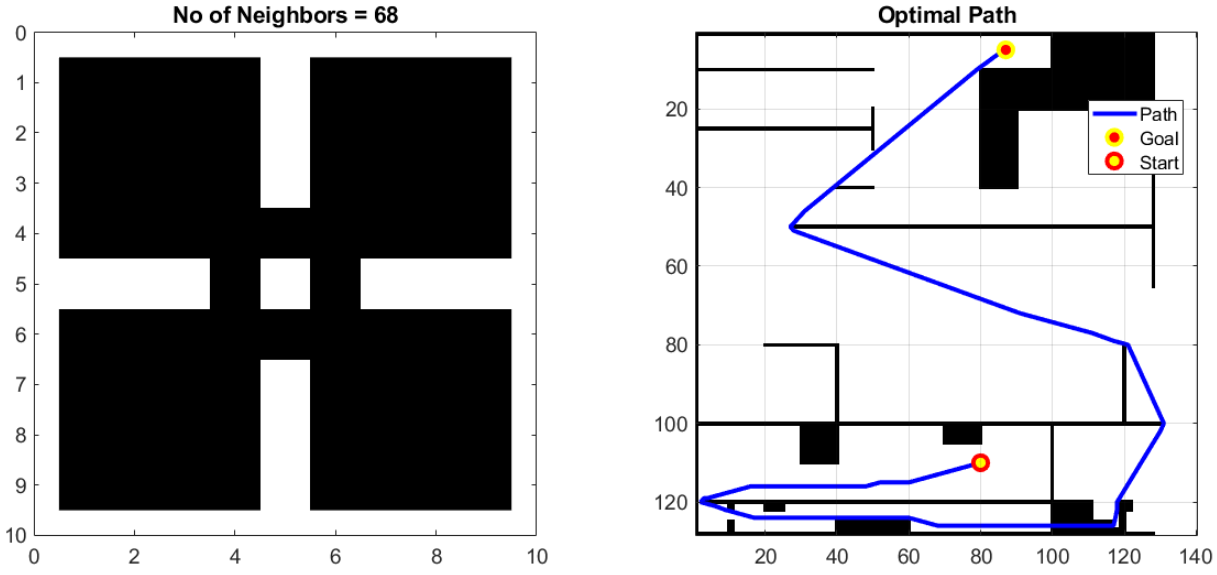

Figure 9: Optimal Path with No. of Neighbors = 68

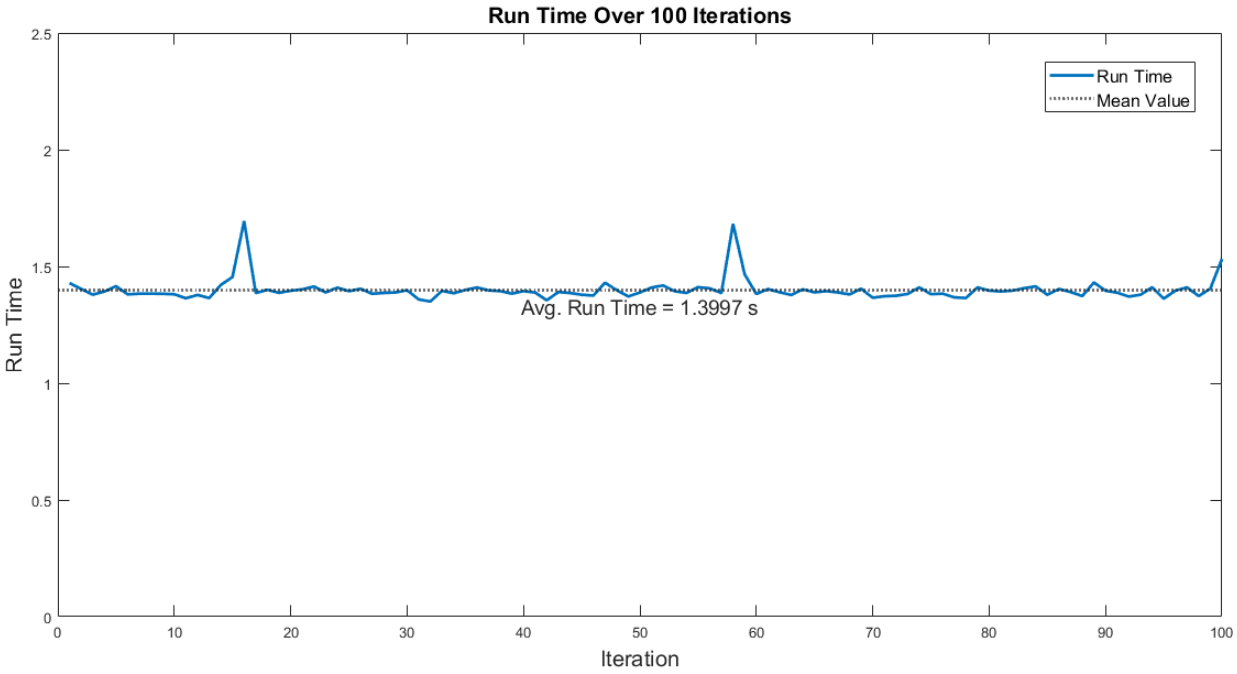

Figure 10: Algorithm Run Time with No. of Neighbors = 68

3 Analysis and Conclusion

The running times for the algorithm with different neighbor sizes can be compared with each other. Refer to the following plot which displays the running time of the algorithm over 100 iterations for different neighboring node sizes.

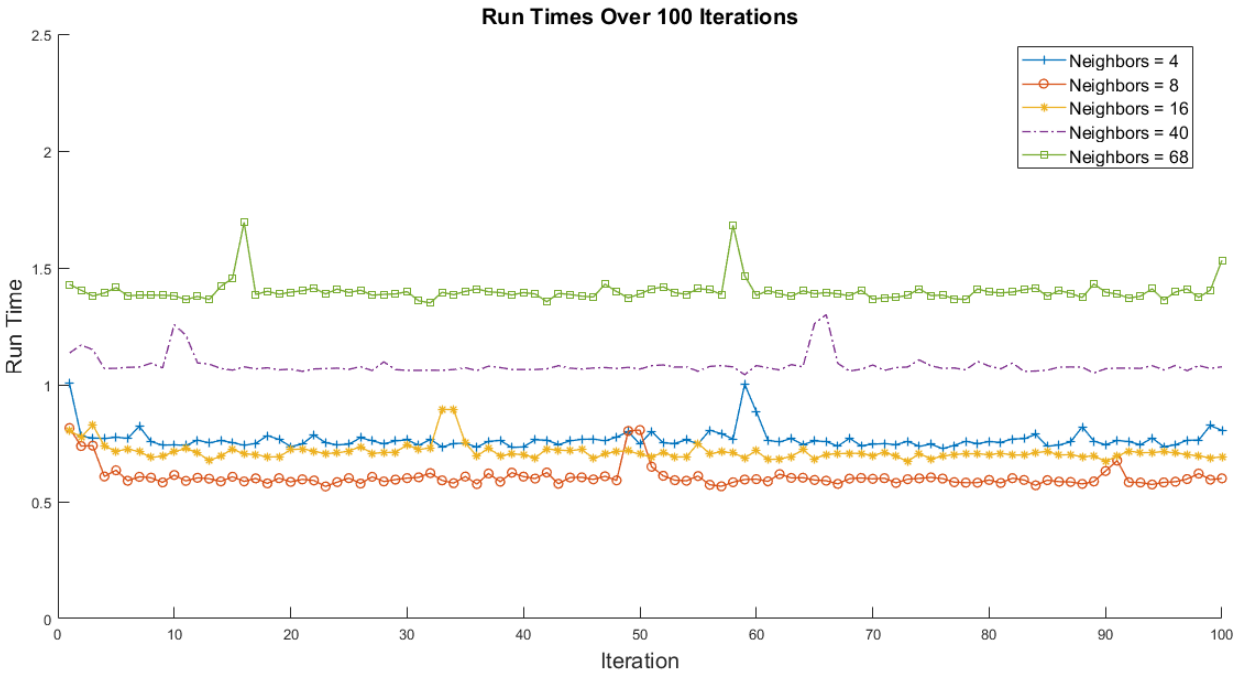

Figure 11: A* Algorithm Run Times with Different Neighboring Node Sizes

As it can be observed, the average running time increases with the size of the matrix that contains the number of neighboring nodes. This result may be expected due to the added overhead of additional computations when traversing over the matrix of neighboring nodes. The anomalous case, however, occurs with the matrix of least neighboring node size, i.e. No. of Neighbors = 4. It may be due to the nature of the central 4-connected sub-matrix within the larger matrix however it cannot be gauged, simply via simulation results, why the algorithm takes a comparatively longer time with No. of Neighbors = 4. The expected trend of increasing running times continues from No. of Neighbors = 8 onward.

From the images of the paths that are obtained, it can be observed that the path becomes much smoother as additional angles are able to be considered when deciding traversal path. Overall, the experiment yielded a good insight into how the A* algorithm can be applied to path traversal problems for mobile robots.

4 MATLAB Code

4.1 A* Algorithm

```
function times = AStarAlgo40c(neighbors)
% A-Star Algorithm in an occupancy grid
clc; close all;
LoadMap % map of obstabcles and available space
%Start Positions
StartX=80;
StartY=110;
%End Positions
TargetX=5;
TargetY=87;
%Height and width of matrix
[Height,Width]=size(MAP);
%Matrix keeping track of G-scores
GScore=zeros(Height,Width);
%Matrix keeping track of F-scores (only open list)
FScore=inf(Height,Width);
%Heuristic matrix
Hn=zeros(Height,Width);
%Matrix keeping of open grid cells
OpenSet=zeros(Height,Width);
%Matrix keeping track of closed grid cells
ClosedSet=zeros(Height,Width);
%Adding object-cells to closed matrix
ClosedSet(MAP==1)=1;
%Matrix keeping track of X position of parent
ParentX=zeros(Height,Width);
%Matrix keeping track of Y position of parent
ParentY=zeros(Height,Width);
%%% Setting up matrices representing neighboors to be investigated

NeighborCheck=int8(neighbors);
[row,col]=find(NeighborCheck==1);
Neighbors=[row col]-5;
N_Neighbors=size(col,1);

% timing code
t = 0;
times = [];
for ii = 1:100
% Creating Heuristic-matrix based on distance to nearest goal node
for k=1:size(MAP,1)
    for j=1:size(MAP,2)
        if MAP(k,j)==0
            % Difference of coordinates, finding distance
            Hn(k,j)=norm([TargetX TargetY]-[k j]);
        end
    end
end

%Initializign start node with FValue and opening first node.
FScore(StartY,StartX) = Hn(StartY,StartX);
OpenSet(StartY,StartX) = 1; RECONSTRUCTPATH=1;

```

```
    while 1==1 %Code will break when path found or when no path exist
    tic %starting stopwatch timer
    MINopenFSCORE=min(min(FScore));
    if MINopenFSCORE==inf;
        %Failuere!
        OptimalPath=[inf];
        RECONSTRUCTPATH=0;
        break
    end

    [CurrentX,CurrentY]=find(FScore==MINopenFSCORE);
    CurrentX=CurrentX(1);
    CurrentY=CurrentY(1);
    FScore(CurrentX,CurrentY)=inf;

    if CurrentX==TargetX && CurrentY==TargetY
        break
    end

    for p=1:N_Neighbors
        i=Neighbors(p,1); %X
        j=Neighbors(p,2); %Y
        % Checking if neighbor is inside or outside the MAP
        % the neighbor is outside the MAP, ignore it,
        % and move to next neighbor (use continue)
        if CurrentX+i<1 || CurrentX+i>Height ...
                || CurrentY+j<1 || CurrentY+j>Width
            continue
        end
        % Now checking obstacle
        Flag=1;
        if (ClosedSet(CurrentX+i,CurrentY+j)==0) %Neighbor is open to move
            if (abs(i)>1||abs(j)>1)
                JumpCellsi=(1:abs(i))*sign(i);
                JumpCellsj=(1:abs(j))*sign(j);
                for Ki=1:length(JumpCellsi)
                    for Kj=1:length(JumpCellsj)
                        if (MAP(CurrentX+JumpCellsi(Ki),...
                                CurrentY+JumpCellsj(Kj))==1)
                            Flag=0;
                        end
                    end
                end
            end
            %End of checking that the path does not pass an object
            if Flag==1;
                Dist_Curr_Neighb=sqrt(i^2+j^2);
                Tentative_GScore = GScore(CurrentX,CurrentY)+Dist_Curr_Neighb;
                if OpenSet(CurrentX+i,CurrentY+j)==0
                    OpenSet(CurrentX+i,CurrentY+j)=1;
                elseif GScore(CurrentX+i,CurrentY+j) < Tentative_GScore
                    continue
                end
                % This path to neighbor is better than any previous one; record it!
                ParentX(CurrentX+i,CurrentY+j)=CurrentX;
                ParentY(CurrentX+i,CurrentY+j)=CurrentY;
                GScore(CurrentX+i,CurrentY+j)=Tentative_GScore;
                FScore(CurrentX+i,CurrentY+j)= ...
                    GScore(CurrentX+i,CurrentY+j)...
```

```
                    + Hn(CurrentX+i,CurrentY+j);
            end
        end
    end
    t = t + toc;
    end
    times = [times t];
    t = 0;
end

if RECONSTRUCTPATH
    k=1; OptimalPath(1,:)=[CurrentX CurrentY];
    while 1==1
        k=k+1;
        CurrentYDummy=ParentY(CurrentX,CurrentY);
        if CurrentYDummy==0; break; end
        CurrentX=ParentX(CurrentX,CurrentY);
        CurrentY=CurrentYDummy;
        OptimalPath(k,:)=[CurrentX CurrentY];
        if CurrentX==StartX && CurrentY==StartY
            break
        end
    end
end
r=size(NeighborCheck,1);
figure; subplot 121; imagesc(NeighborCheck);colormap(flipud(gray));
axis([0 r+1 0 r+1]);title(['No of Neighbors = ' num2str(N_Neighbors)])
axis square; set(gca,'fontsize',14);
if size(OptimalPath,2)>1
    subplot 122;imagesc(MAP);colormap(flipud(gray)); hold on
    plot(OptimalPath(:,2),OptimalPath(:,1),'b','linewidth',3)
    plot(OptimalPath(1,2),OptimalPath(1,1),'o',...
        'color','y','markerfacecolor','r','markersize',10,'linewidth',3)
    plot(OptimalPath(end,2),OptimalPath(end,1),'o',...
        'color','r','markerfacecolor','y','markersize',10,'linewidth',3)
    legend('Path','Goal','Start','Location','Best');
    axis square; set(gca,'fontsize',14);grid
    title('Optimal Path')
else
    pause(1);
    h=msgbox('Sorry, No path exists to the Target!','warn');
    uiwait(h,5);
end
```

4.2 Main Script

```
% Neighbor Matrices
n4 = [ ...
       0 0 0 0 0 0 0 0 0;
       0 0 0 0 0 0 0 0 0;
       0 0 0 0 0 0 0 0 0;
       0 0 0 0 1 0 0 0 0;
       0 0 0 1 0 1 0 0 0;
       0 0 0 0 1 0 0 0 0;
       0 0 0 0 0 0 0 0 0;
       0 0 0 0 0 0 0 0 0;
```

```
        0 0 0 0 0 0 0 0 0];
n8 = [ ...
        0 0 0 0 0 0 0 0 0;
        0 0 0 0 0 0 0 0 0;
        0 0 0 0 0 0 0 0 0;
        0 0 0 1 1 1 0 0 0;
        0 0 0 1 0 1 0 0 0;
        0 0 0 1 1 1 0 0 0;
        0 0 0 0 0 0 0 0 0;
        0 0 0 0 0 0 0 0 0;
        0 0 0 0 0 0 0 0 0];
n16 = [ ...
         0 0 0 0 0 0 0 0 0
         0 0 0 0 0 0 0 0 0;
         0 0 0 1 0 1 0 0 0;
         0 0 1 1 1 1 1 0 0;
         0 0 0 1 0 1 0 0 0;
         0 0 1 1 1 1 1 0 0;
         0 0 0 1 0 1 0 0 0;
         0 0 0 0 0 0 0 0 0;
         0 0 0 0 0 0 0 0 0];
n40 = [ ...
         0 0 0 0 0 0 0 0 0;
         0 1 1 1 0 1 1 1 0;
         0 1 1 1 0 1 1 1 0;
         0 1 1 1 1 1 1 1 0;
         0 0 0 1 0 1 0 0 0;
         0 1 1 1 1 1 1 1 0;
         0 1 1 1 0 1 1 1 0;
         0 1 1 1 0 1 1 1 0;
         0 0 0 0 0 0 0 0 0];
n68 = [
         1 1 1 1 0 1 1 1 1;
         1 1 1 1 0 1 1 1 1;
         1 1 1 1 0 1 1 1 1;
         1 1 1 1 1 1 1 1 1;
         0 0 0 1 0 1 0 0 0;
         1 1 1 1 1 1 1 1 1;
         1 1 1 1 0 1 1 1 1;
         1 1 1 1 0 1 1 1 1;
         1 1 1 1 0 1 1 1 1];

times4 = AStarAlgo40c(n4);
times8 = AStarAlgo40c(n8);
times16 = AStarAlgo40c(n16);
times40 = AStarAlgo40c(n40);
times68 = AStarAlgo40c(n68);

% individual plot generation
figure; hold on
plot(times4,'-+','LineWidth', 1)
plot(times8,'-o','LineWidth', 1)
plot(times16,'-*','LineWidth', 1)
plot(times40,'-.','LineWidth', 1)
plot(times68,'-s','LineWidth', 1)
ylim([0 2.5])
title('Run Times Over 100 Iterations','FontSize', 15)
xlabel('Iteration','FontSize', 15)
ylabel('Run Time','FontSize', 15)
```

```
av4 = mean(times4)
av8 = mean(times8)
av16 = mean(times16)
av40 = mean(times40)
av68 = mean(times68)
str4 = num2str(av4)
str8 = num2str(av8)
str16 = num2str(av16)
str40 = num2str(av40)
str68 = num2str(av68)
legend('Neighbors = 4','Neighbors = 8','Neighbors = 16','Neighbors = 40','Neighbors = ...
    68','FontSize',12,'Location','Best')
% yline(mean(times68),':',['Avg. Run Time = ',str,' ...
    s'],'LineWidth',2,'LabelHorizontalAlignment','center','LabelVerticalAlignment','bottom','FontSize
    14)
```