Assignment 1

Q1

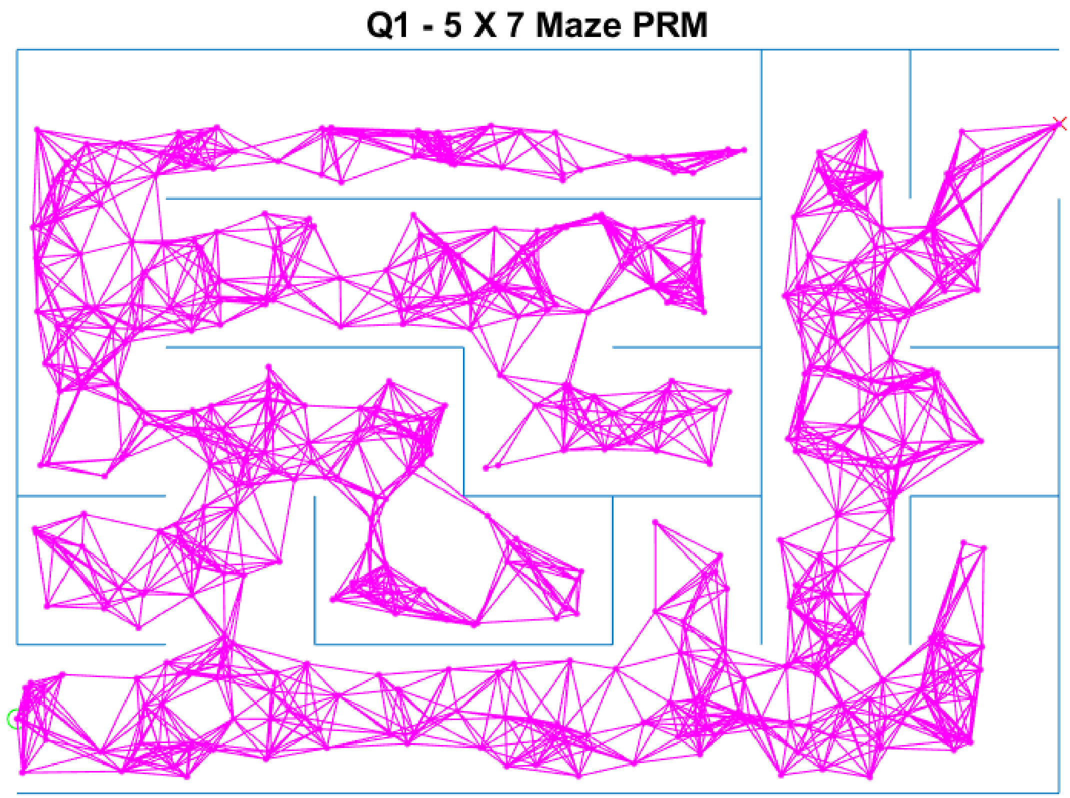

Figure 1: Q1 output

The milestones were generated randomly such that there are points that are very close to each other. The connections between points are also pretty excessive.

Q2

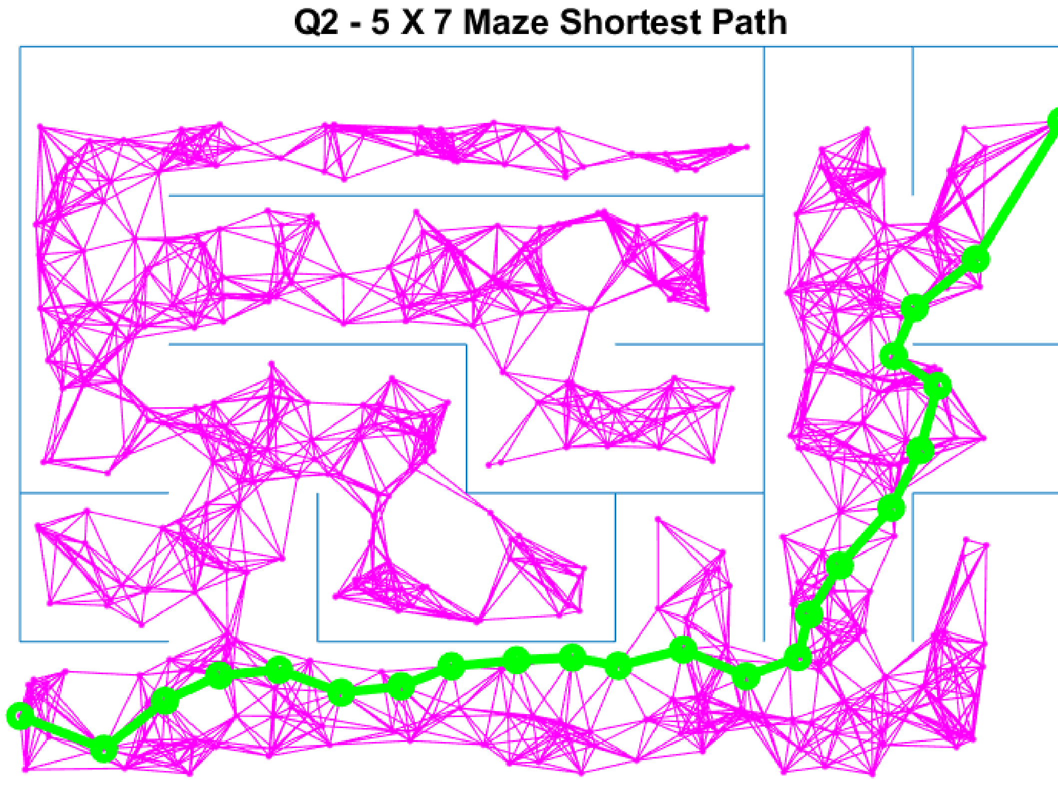

Figure 2: Q2 output

The algorithm that was used to solve this maze is A^*. The shortest path it generated seems to make sense. However, this path is curvy. Imagine a real robot is moving like this inside a hallway. This behaviour is weird.

Q3

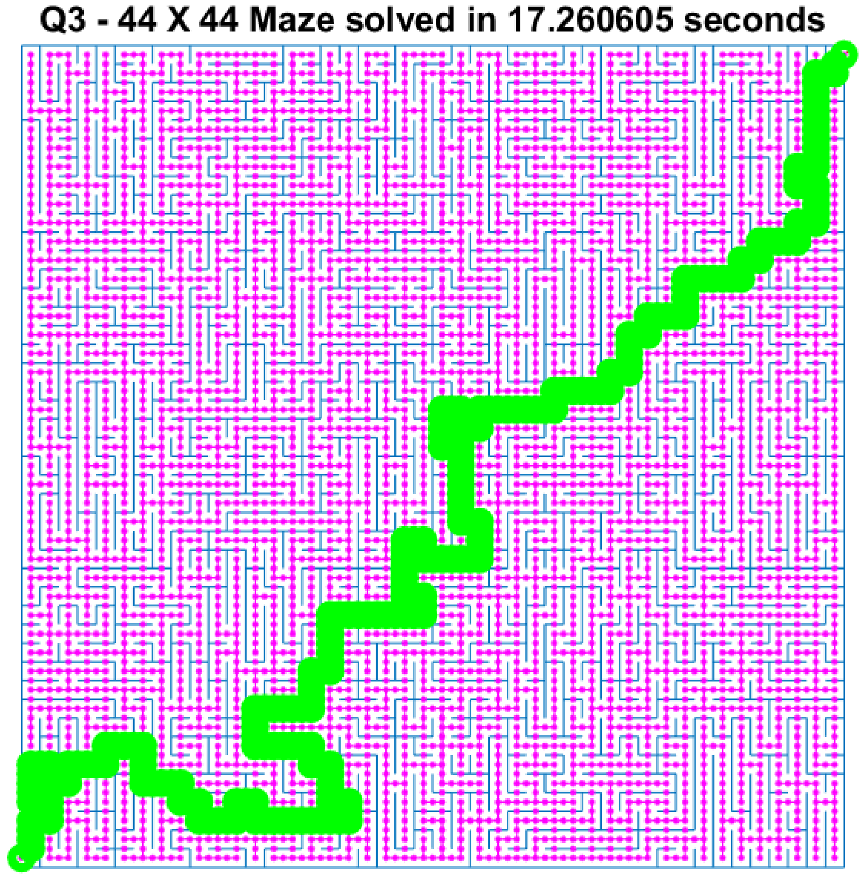

Figure 3: Q3 output

NOTE: My laptop has i7 −7700HQ 4 Core CPU, it can run 44x44 maze under 20 seconds (45x45 takes 21s). The ECF computer can run 50x50 maze under 20 seconds (15 seconds).

Mainly the milestone generation and child finding in A^* is improved to obtain a faster solution.

For milestone generation:

This new milestone generation strategy is totally different from the PRM. Since, we know the shap of the maze, I generated all possible points (with 0.5 increments) that lie "perfectly" in the maze and then check if the points are on the edges of the maze. During the checking phase, I only check the edges that are close to the points to further reduce the run time (this checking is not perfect, you can see there are points that lie on the edge of the horizontal edges. this can be further improved to reduce time used in connecting the milestones).

For child finding:

Instead of using a for loop that loops through all the edges, I used another matlab function, "find", to get the edges that contain the current node. Then, I can get the node's children using the indices produced by "find".

My code is listed starting from the next page.

```
% ======
% ROB521_assignment1.m
% ======
%
% This assignment will introduce you to the idea of motion planning for
% holonomic robots that can move in any direction and change direction of
% motion instantaneously.  Although unrealistic, it can work quite well for
% complex large scale planning.  You will generate mazes to plan through
% and employ the PRM algorithm presented in lecture as well as any
% variations you can invent in the later sections.
%
% There are three questions to complete (5 marks each):
%
%    Question 1: implement the PRM algorithm to construct a graph
%    connecting start to finish nodes.
%    Question 2: find the shortest path over the graph by implementing the
%    Dijkstra's or A* algorithm.
%    Question 3: identify sampling, connection or collision checking
%    strategies that can reduce runtime for mazes.
%
% Fill in the required sections of this script with your code, run it to
% generate the requested plots, then paste the plots into a short report
% that includes a few comments about what you've observed.  Append your
% version of this script to the report.  Hand in the report as a PDF file.
%
% requires: basic Matlab,
%
% S L Waslander, January 2022
%
clear; close all; clc;

% set random seed for repeatability if desired
rng(1,'twister');

% ==========================
% Maze Generation
% ==========================
%
```

```
% The maze function returns a map object with all of the edges in the maze.
% Each row of the map structure draws a single line of the maze.  The
% function returns the lines with coordinates [x1 y1 x2 y2].
% Bottom left corner of maze is [0.5 0.5],
% Top right corner is [col+0.5 row+0.5]


row = 5; % Maze rows
col = 7; % Maze columns
map = maze(row,col); % Creates the maze
start = [0.5, 1.0]; % Start at the bottom left
finish = [col+0.5, row]; % Finish at the top right

h = figure(1);clf; hold on;
plot(start(1), start(2),'go')
plot(finish(1), finish(2),'rx')
show_maze(map,row,col,h); % Draws the maze
drawnow;

% ==========================================================
% Question 1: construct a PRM connecting start and finish
% ==========================================================
%
% Using 500 samples, construct a PRM graph whose milestones stay at least
% 0.1 units away from all walls, using the MinDist2Edges function provided for
% collision detection.  Use a nearest neighbour connection strategy and the
% CheckCollision function provided for collision checking, and find an
% appropriate number of connections to ensure a connection from  start to
% finish with high probability.


% variables to store PRM components
nS = 500;  % number of samples to try for milestone creation
milestones = [start; finish];  % each row is a point [x y] in feasible space
edges = [];  % each row should be an edge of the form [x1 y1 x2 y2]

disp("Time to create PRM graph")
tic;
% ------insert your PRM generation code here-------
% get as many feasible milestones as nS milestones + 2 (start and finish)
```

```
while length(milestones) < (nS + 2)

    % random choose points in the map
    qx = (col-0.5).*rand(1,1) + 0.5;
    qy = (row-0.5).*rand(1,1) + 0.5;
    q = [qx, qy];

    % check if q is collision-free using MinDist2Edges
    if MinDist2Edges(q, map) >= 0.1
        milestones = [milestones; q];
    end
    milestones = unique(milestones,"rows");
end

% tested several numbers and 6 is the smallest one that has the highest prb
num_of_neighbours = 10;

% connect the milestones and hence form the PRM graph
for i = (1:length(milestones))

    cur_node = milestones(i,:);

    % remove the current node from the milestones
    rem_milestones = milestones;
    rem_milestones(i,:) = [];

    % use k nearest nodes
    for j = (1:num_of_neighbours)

        % find the closest point
        [k, dist] = dsearchn(rem_milestones,cur_node);
        % connect the current node with its closest node and check collision
        current_edge = [cur_node,rem_milestones(k,:)];

        % remove the current closest node from the reamining of milestones
        rem_milestones(k,:) = [];

        % check collision
        [ inCollision , edge ] = CheckCollision(current_edge(1:2), current_edge(3:4), map);

        if inCollision == 0 && length(edges) == 0
            edges = [edges;current_edge];
        else
```

```
            if inCollision == 0 && length(edges) ~= 0

                % check if current edge already exist and check collision
                [~,index] = ismember(current_edge,edges,"rows");
                if index == 0
                    edges = [edges;current_edge];
                end
            end
        end
    end
end
% ----------end of your PRM generation code ----------
toc;

figure(1);
plot(milestones(:,1),milestones(:,2),'m.');
if (~isempty(edges))
    line(edges(:,1:2:3)', edges(:,2:2:4)','Color','magenta') % line uses [x1 x2 y1 y2]
end
str = sprintf('Q1 - %d X %d Maze PRM', row, col);
title(str);
drawnow;

print -dpng assignment1_q1.png


% ==================================================================
% Question 2: Find the shortest path over the PRM graph
% ==================================================================
%
% Using an optimal graph search method (Dijkstra's or A*) , find the
% shortest path across the graph generated.  Please code your own
% implementation instead of using any built in functions.

disp('Time to find shortest path');
tic;

% Variable to store shortest path
spath = []; % shortest path, stored as a milestone row index sequence

% ----------insert your shortest path finding algorithm here----------
% A* implementation with Manhattan distance as the heuristic

start_node.point = start;
start_node.g = 0;
```

```
start_node.h = 0;% Manhattan(start,finish);
start_node.f = 0;% start_node.g+start_node.h;
start_node.parent = [];
[~, start_node.position] = ismember(start_node.point, milestones, "rows");
visited = [start_node.point];
open_list = [start_node];
closed_list = [];

while ~isempty(open_list)

    % get the current node that has the least f value using sorting
    [~,index] = sortrows([open_list.f].');
    open_list = open_list(index);
    node = open_list(1);

    % remove current from open and add to close
    open_list(1) = [];
    closed_list = [closed_list;node];

    % found the goal
    if node.point == finish
        cur_node = node;
        while ~isempty(cur_node)
            spath = [spath;cur_node.position];
            cur_node = cur_node.parent;
        end
        display("Congraz!!!");
        break
    end

    % generate children
    children = [];
    for i=(1:length(edges))
        if node.point == edges(i,1:2)
            child.point = edges(i,3:4);
            children = [children;child];
        elseif node.point == edges(i,3:4)
            child.point = edges(i,1:2);
            children = [children;child];
        end
    end

    % loop through children
    for i = (1:length(children))
        child = children(i);
```

```
        % check child in open or closed
        open_list_points = reshape([open_list.point],2,[])';
        [~,oindex] = ismember(child.point,open_list_points,"rows");
        close_list_points = reshape([closed_list.point],2,[])';
        [~,cindex] = ismember(child.point,close_list_points,"rows");

        % if child is in closed
        if cindex ~= 0
            continue;
        end

        % create the f g h values
        g = node.g + sum(abs(child.point-finish));

        % check if the child has not been visited yet
        % can be reached with smaller cost from the current
        if isfield(child,'g')
            if oindex == 0 ||  g < child.g
                child.g = g;
                child.h = sum(abs(child.point-finish));
                child.f = child.g+child.h;
                child.parent = node;
                [~,child.position] = ismember(child.point, milestones,"rows");

                if oindex == 0
                    open_list = [open_list;child];
                else
                    open_list(oindex) = child;
                end
            end
        else
            if oindex == 0
                child.g = g;
                child.h = sum(abs(child.point-finish));
                child.f = child.g+child.h;
                child.parent = node;
                [~,child.position] = ismember(child.point, milestones,"rows");

                if oindex == 0
                    open_list = [open_list;child];
                else
                    open_list(oindex) = child;
                end
            end
        end
```

```
    end
end

if node.point ~= finish
    disp('Falied');
end

% --------end of shortest path finding algorithm---------
toc;

% plot the shortest path
figure(1);
for i=1:length(spath)-1
    plot(milestones(spath(i:i+1),1),milestones(spath(i:i+1),2), 'go-',
        'LineWidth',3);
end
str = sprintf('Q2 - %d X %d Maze Shortest Path', row, col);
title(str);
drawnow;

print -dpng assingment1_q2.png


%% =======================================================================
%% Question 3: find a faster way
%% =======================================================================
%%
%% Modify your milestone generation, edge connection, collision
   detection
%% and/or shortest path methods to reduce runtime.  What is the
   largest maze
%% for which you can find a shortest path from start to goal in under
    20
%% seconds on your computer? (Anything larger than 40x40 will suffice
    for
%% full marks)

% My laptop has i7-7700HQ 4 Core CPU, it can run 44x44 maze under 20
% seconds (~16 seconds).
% The ECF computer can run 50x50 maze under 20 seconds (~15 seconds)

row = 44;
col = 44;
map = maze(row,col);
start = [0.5, 1.0];
finish = [col+0.5, row];
milestones = [start; finish];  % each row is a point [x y] in feasible
```

```
    space
edges = [];  % each row is should be an edge of the form [x1 y1 x2 y2]

h = figure(2);clf; hold on;
plot(start(1), start(2),'go')
plot(finish(1), finish(2),'rx')
show_maze(map,row,col,h); % Draws the maze
drawnow;

fprintf("Attempting large %d X %d maze... \n", row, col);

% --------insert your optimized algorithm here--------
% variables to store PRM components
% nS = 5000;  % number of samples to try for milestone creation
disp("Time to create PRM graph")
% --------insert your PRM generation code here---------
% get as many feasible milestones as nS milestones
tic;
% setup a list of numbers that will be used for the coordinates
% to reduce bad random generated numbers that will cause collision
Rx = 0.5:0.5:col;
Ry = 1:0.5:row;

% generate all possible points
[X, Y] = meshgrid(Rx, Ry);
q = [X(:),Y(:)];

% only check edges that are close (in terms of x-axis)
% to the current milestones
k = [];
for i = (1:length(q))
    if mod(i,row*4-2) <= row*2-1 && mod(i,row*4-2) > 0
        x1 = find(map(:,1)==q(i,1));
        x2 = find(map(:,3)==q(i,1));
        x = unique([x1;x2],"rows");

        if ~isempty(x)
            dx = MinDist2Edges(q(i,:), map(x,:));
            if dx < 0.1
                k = [k;i];
                continue
            end
        end

        y1 = find(map(:,2)==q(i,2));
        y2 = find(map(:,4)==q(i,2));
        y = unique([y1;y2],"rows");
```

```
        if ~isempty(y)
            dy = MinDist2Edges(q(i,:), map(:,y));
            if dy < 0.1
                k = [k;i];
            end
        end
    end
end

q(k,:) = [];
milestones = [finish;q];

num_of_neighbours = 2;

% connect the milestones and hence form the PRM graph
for i = (1:length(milestones))

    cur_node = milestones(i,:);

    % remove the current node from the milestones
    rem_milestones = milestones;
    rem_milestones(i,:) = [];

    if ~isempty(edges)
        % remove the parent of this node from the milestones
        [~,index] = ismember(cur_node,edges(3:4),"rows");
        if index ~= 0
            rem_milestones(index,:) = [];
        end
    end

    % use k nearest nodes
    for j = (1:num_of_neighbours)

        % this is faster than dsearchn
        distance=sqrt((rem_milestones(:,1)-cur_node(1)).^2+(
           rem_milestones(:,2)-cur_node(2)).^2);
        [C,k] = min(distance);
        E = rem_milestones(k,:);

        % remove the current closest node from the reamining of
           milestones
        rem_milestones(k,:) = [];

        % connect the current node with its closest node and check
           collision
```

```
        current_edge = [cur_node,E];

        % check collision
        [ inCollision , edge ] = CheckCollision(current_edge(1:2),
           current_edge(3:4), map);

        if inCollision == 0 && isempty(edges)
            edges = [edges;current_edge];
        else
            if inCollision == 0 && ~isempty(edges)
                edges = [edges;current_edge];
            end
        end
    end

end

% -------end of your PRM generation code --------

% Variable to store shortest path
spath = []; % shortest path, stored as a milestone row index sequence
% A* implementation with Manhattan distance as the heuristic

start_node.point = start;
start_node.g = 0;
start_node.h = 0;% Manhattan(start,finish);
start_node.f = 0;% start_node.g+start_node.h;
start_node.parent = [];
[~, start_node.position] = ismember(start_node.point, milestones, "
   rows");
visited = [start_node.point];
open_list = [start_node];
closed_list = [];

while ~isempty(open_list)

    % get the current node that has the least f value (faster than
       sort)
    [~,index] = min([open_list.f]);
    node = open_list(index);

    % remove current from open and add to close
    open_list(index) = [];
    closed_list = [closed_list;node];

    % found the goal

```

```
    if node.point == finish
        cur_node = node;
        while ~isempty(cur_node) % this loop is fast, no need to optmize
            spath = [spath;cur_node.position];
            cur_node = cur_node.parent;
        end
        display("Congraz!!!");
        break
    end

    % generate children
    children = [];
    % if node.point is edges(i,1:2) then its child is edges(i,3:4)
    node_x_idx = find(edges(:,1)==node.point(1));
    node_y_idx = find(edges(:,2)==node.point(2));
    child_idx = intersect(node_x_idx,node_y_idx);
    for i=(1:length(child_idx))
        child.point = edges(child_idx(i),3:4);
        children = [children;child];
    end
    % if node.point is edges(i,3:4) then its child is edges(i,1:2)
    node_x_idx = find(edges(:,3)==node.point(1));
    node_y_idx = find(edges(:,4)==node.point(2));
    child_idx = intersect(node_x_idx,node_y_idx);
    for i=(1:length(child_idx))
        child.point = edges(child_idx(i),1:2);
        children = [children;child];
    end

    % loop through children
    for i = (1:length(children))
        child = children(i);

        % check child in open or closed
        open_list_points = reshape([open_list.point],2,[])';
        [~,oindex] = ismember(child.point,open_list_points,"rows");
        close_list_points = reshape([closed_list.point],2,[])';
        [~,cindex] = ismember(child.point,close_list_points,"rows");

        % if child is in closed
        if cindex ~= 0
            continue;
        end

        % create the f g h values
        g = node.g + sum(abs(child.point-finish));
```

```matlab

        % check if the child has not been visited yet
        % can be reached with smaller cost from the current
        if isfield(child,'g')
            if oindex == 0 ||  g < child.g
                child.g = g;
                child.h = sum(abs(child.point-finish));
                child.f = child.g+child.h;
                child.parent = node;
                [~,child.position] = ismember(child.point, milestones,"rows");

                if oindex == 0
                    open_list = [open_list;child];
                else
                    open_list(oindex) = child;
                end
            end
        else
            if oindex == 0
                child.g = g;
                child.h = sum(abs(child.point-finish));
                child.f = child.g+child.h;
                child.parent = node;
                [~,child.position] = ismember(child.point, milestones,"rows");

                if oindex == 0
                    open_list = [open_list;child];
                else
                    open_list(oindex) = child;
                end
            end
        end
    end
end
% ------end of your optimized algorithm-------
dt = toc;


[~,index] = ismember(finish,milestones,"rows");
if node.position ~= index
    disp('Falied');
end

figure(2); hold on;
plot(milestones(:,1),milestones(:,2),'m.');
```

```
if (~isempty(edges))
    line(edges(:,1:2:3)', edges(:,2:2:4)','Color','magenta')
end
if (~isempty(spath))
    for i=1:length(spath)-1
        plot(milestones(spath(i:i+1),1),milestones(spath(i:i+1),2), 'go-', 'LineWidth',3);
    end
end
str = sprintf('Q3 - %d X %d Maze solved in %f seconds', row, col, dt);
title(str);

print -dpng assignment1_q3.png
```