```
% ======
% 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);

% ==========================
% 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 is should be an edge of the form [x1 y1 x2 y2]

disp("Time to create PRM graph")
tic;
% ------insert your PRM generation code here-------
nR = 20;         % no. of points to sample along the row dimension
nC = 25;         % no. of points to sample along the column dimension
row_min = 0.5;
row_max = 5.5;
col_min = 0.5;
col_max = 7.5;
row_pts = linspace(row_min, row_max, nR);
col_pts = linspace(col_min, col_max, nC);

% milestone calculation
for x = col_pts
    for y = row_pts
        min_dis = MinDist2Edges([x, y], map);
        if min_dis>=0.1
```

```
            milestones = [milestones; [x, y]];
        end
    end
end

% build graph
nn_dist = 0.5;  % node that characterizes into nearest neighbour

for i = 1:length(milestones)
    for j = i+1:length(milestones)
        node1    = milestones(i,:);
        node2    = milestones(j,:);
        distance = norm(node1 - node2);

        if distance<=nn_dist
            [flag, ~] = CheckCollision(node1, node2, map);
            if flag==0   % no collisions
                edges = [edges; [node1(1), node1(2), node2(1),
 node2(2)]];
            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-------
visited  = [start];     % list maintaining the visited nodes
```

```
pr_queue = [start 0];   % priority queue, the third column is cost-to-
come

% dictionary for backtracking
M = containers.Map('KeyType','int32', 'ValueType','any');

while ~isempty(pr_queue)
   top_element   = pr_queue(1,:);      % get top element
   cost_to_come  = top_element(3);     % cost-to-come for the node
   parent_node   = top_element(1:2);   % get the node
   pr_queue(1,:) = [];                 % pop top element

   if parent_node == finish
       disp('Reached goal!')
       break
   end

   neighbor_list = find_neighbours(parent_node, edges);
   for i = 1:length(neighbor_list)
       if ismember(neighbor_list(i,:),visited,'rows')==0
           visited = [visited; neighbor_list(i,:)];
           tot_cost_to_come = cost_to_come + norm(neighbor_list(i,:) -
 parent_node);

           % populate dictionary for backtracking
           cur_node_idx    = find_milestone_idx(neighbor_list(i,:),
 milestones);
           par_node_idx    = find_milestone_idx(parent_node,
 milestones);
           M(cur_node_idx) = [par_node_idx, tot_cost_to_come];

           % make appends to priority queue
           pr_queue = [pr_queue; neighbor_list(i,:) tot_cost_to_come];
       end
   end

   pr_queue = sortrows(pr_queue, 3);

end

spath    = [2];     % 2 is milestone index
x        = 2;
finish_p = M(2);
short_path = finish_p(2);
X = ['The shortest path is: ', num2str(short_path), ' units'];
disp(X)

start_idx  = 1;

while x~=start_idx
    parent = M(x);
    x      = int32(parent(1));
    spath  = [x; spath];
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)

```
row = 42;
col = 42;
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);
tic;
% ------insert your optimized algorithm here------

% =================================================================
% Part I: Graph Generation
% =================================================================

nR = row;          % no. of points to sample along the row dimension
nC = col;          % no. of points to sample along the column dimension
row_min = 1;
row_max = row;
col_min = 1;
col_max = col;
row_pts = linspace(row_min, row_max, nR);
```

```
col_pts = linspace(col_min, col_max, nC);

% milestone calculation; no need to check for collision
for x = col_pts
    for y = row_pts
        milestones = [milestones; [x, y]];
    end
end

% build graph
nn_dist = 1;  % node that characterizes into nearest neighbour

for i = 1:length(milestones)
    for j = i+1:length(milestones)
        node1    = milestones(i,:);
        node2    = milestones(j,:);
        distance = norm(node1 - node2);

        if distance<=nn_dist
            [flag, ~] = CheckCollision(node1, node2, map);
            if flag==0   % no collisions
                edges = [edges; [node1(1), node1(2), node2(1),
 node2(2)]];
            end
        end
    end
end

% ================================================================
% Part II: Finding shortest path
% ================================================================

spath = [];

visited  = [start];     % list maintaining the visited nodes
pr_queue = [start 0];   % priority queue, the third column is cost-to-
come

% dictionary for backtracking
M = containers.Map('KeyType','int32', 'ValueType','any');

while ~isempty(pr_queue)
   top_element   = pr_queue(1,:);     % get top element
   cost_to_come  = top_element(3);    % cost-to-come for the node
   parent_node   = top_element(1:2);  % get the node
   pr_queue(1,:) = [];                % pop top element

   if parent_node == finish
       disp('Reached goal!')
       break
   end

   neighbor_list = find_neighbours(parent_node, edges);
   neighbor_size = size(neighbor_list);
```

```
   for i = 1:neighbor_size(1)
       if ismember(neighbor_list(i,:),visited,'rows')==0
           visited = [visited; neighbor_list(i,:)];
           tot_cost_to_come = cost_to_come + norm(neighbor_list(i,:) -
 parent_node);

           % populate dictionary for backtracking
           cur_node_idx    = find_milestone_idx(neighbor_list(i,:),
 milestones);
           par_node_idx    = find_milestone_idx(parent_node,
 milestones);
           M(cur_node_idx) = [par_node_idx, tot_cost_to_come];

           % make appends to priority queue
           pr_queue = [pr_queue; neighbor_list(i,:) tot_cost_to_come];
       end
   end

   pr_queue = sortrows(pr_queue, 3);

end

spath    = [2];     % 2 is milestone index
x        = 2;
finish_p = M(2);
short_path = finish_p(2);
X = ['The shortest path is: ', num2str(short_path), ' units'];
disp(X)

start_idx  = 1;

while x~=start_idx
    parent = M(x);
    x      = int32(parent(1));
    spath  = [x; spath];
end

% ------end of your optimized algorithm-------
dt = toc;

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
```

Custom function definitions

```
function nbors = find_neighbours(node, edge_list)
% given a node and the graph edge list, returns the neighbours of the
 node

    nbors     = [];   % neighbour list
    tot_edges = length(edge_list);

    for i = 1:tot_edges
        if edge_list(i,1:2)==node
            nbors = [nbors; edge_list(i,3:4)];
        elseif edge_list(i,3:4)==node
            nbors = [nbors; edge_list(i,1:2)];
        else
            continue
        end

    end
end

function index = find_milestone_idx(node, milestones)
% returns the index of the given node in the milestones list

    for i = 1:length(milestones)
        if milestones(i,:)==node
            index = i;
            return
        end
    end

end
```

Published with MATLAB® R2020a