

Assignment 1

Q1

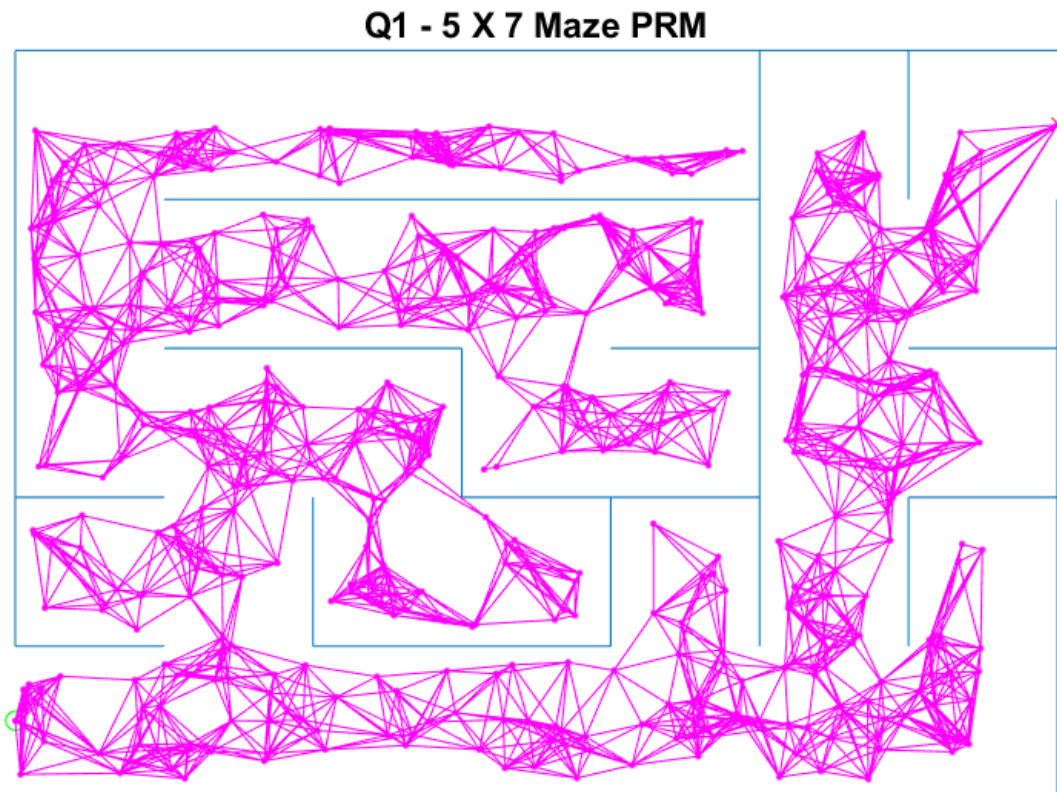


Figure 1: Q1 output

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

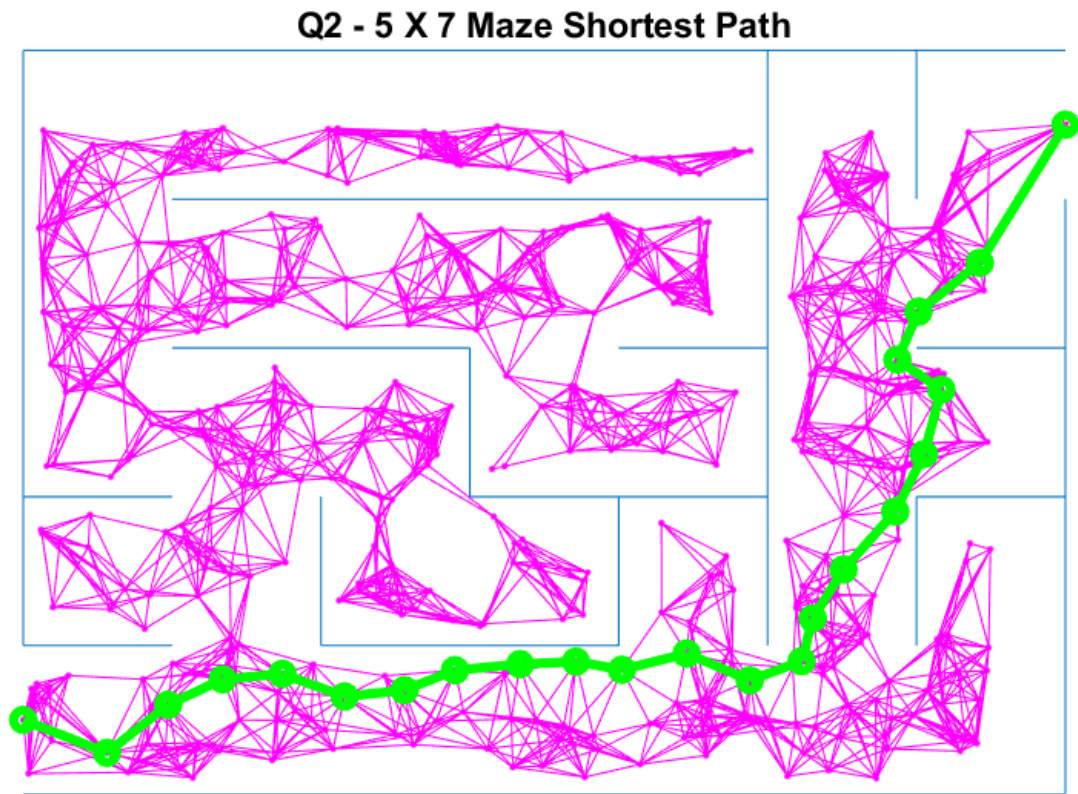


Figure 2: Q2 output

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.

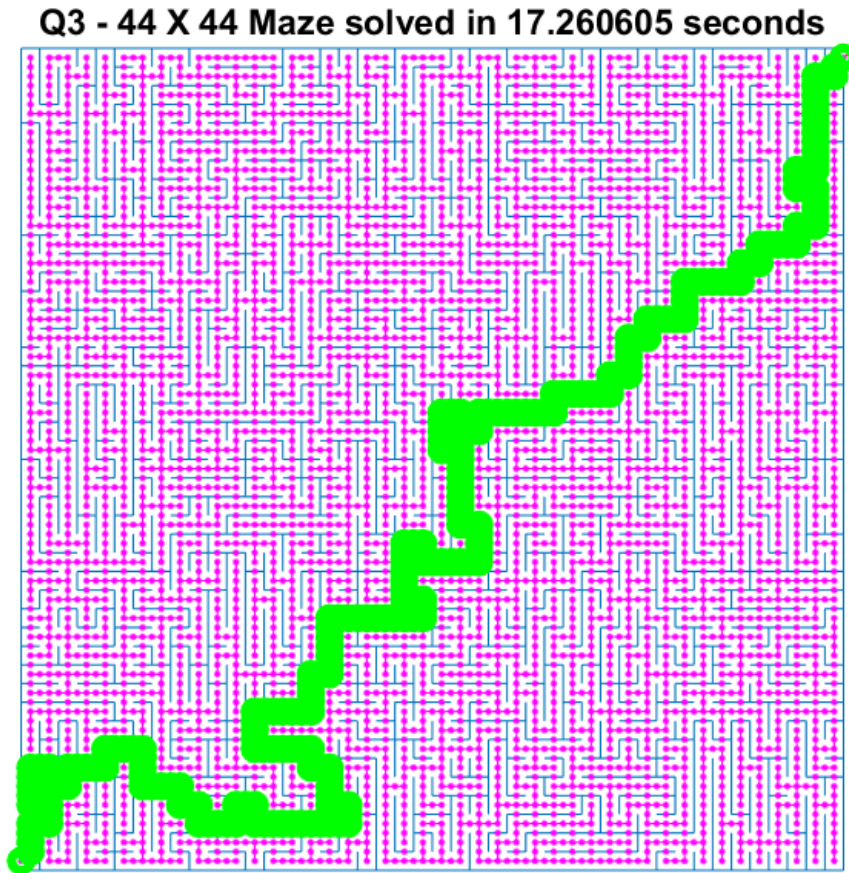


Figure 3: Q3 output

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 shape 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.

```

1 % =====
2 % ROB521_assignment1.m
3 % =====
4 %
5 % This assignment will introduce you to the idea of motion planning
   for
6 % holonomic robots that can move in any direction and change direction
   of
7 % motion instantaneously. Although unrealistic, it can work quite
   well for
8 % complex large scale planning. You will generate mazes to plan
   through
9 % and employ the PRM algorithm presented in lecture as well as any
10 % variations you can invent in the later sections.
11 %
12 % There are three questions to complete (5 marks each):
13 %
14 %     Question 1: implement the PRM algorithm to construct a graph
15 %     connecting start to finish nodes.
16 %     Question 2: find the shortest path over the graph by implementing
   the
17 %     Dijkstra's or A* algorithm.
18 %     Question 3: identify sampling, connection or collision checking
19 %     strategies that can reduce runtime for mazes.
20 %
21 % Fill in the required sections of this script with your code, run it
   to
22 % generate the requested plots, then paste the plots into a short
   report
23 % that includes a few comments about what you've observed. Append
   your
24 % version of this script to the report. Hand in the report as a PDF
   file.
25 %
26 % requires: basic Matlab,
27 %
28 % S L Waslander, January 2022
29 %
30 clear; close all; clc;
31
32 % set random seed for repeatability if desired
33 rng(1, 'twister');
34
35 % =====
36 % Maze Generation
37 % =====
38 %

```

```

39 % The maze function returns a map object with all of the edges in the
    maze.
40 % Each row of the map structure draws a single line of the maze. The
41 % function returns the lines with coordinates [x1 y1 x2 y2].
42 % Bottom left corner of maze is [0.5 0.5],
43 % Top right corner is [col+0.5 row+0.5]
44
45
46 row = 5; % Maze rows
47 col = 7; % Maze columns
48 map = maze(row,col); % Creates the maze
49 start = [0.5, 1.0]; % Start at the bottom left
50 finish = [col+0.5, row]; % Finish at the top right
51
52 h = figure(1); clf; hold on;
53 plot(start(1), start(2), 'go')
54 plot(finish(1), finish(2), 'rx')
55 show_maze(map,row,col,h); % Draws the maze
56 drawnow;
57
58 % =====
59 % Question 1: construct a PRM connecting start and finish
60 % =====
61 %
62 % Using 500 samples, construct a PRM graph whose milestones stay at
    least
63 % 0.1 units away from all walls, using the MinDist2Edges function
    provided for
64 % collision detection. Use a nearest neighbour connection strategy
    and the
65 % CheckCollision function provided for collision checking, and find an
66 % appropriate number of connections to ensure a connection from start
    to
67 % finish with high probability.
68
69
70 % variables to store PRM components
71 nS = 500; % number of samples to try for milestone creation
72 milestones = [start; finish]; % each row is a point [x y] in feasible
    space
73 edges = []; % each row should be an edge of the form [x1 y1 x2 y2]
74
75 disp("Time to create PRM graph")
76 tic;
77 % -----insert your PRM generation code here-----
78 % get as many feasible milestones as nS milestones + 2 (start and
    finish)

```

```

79 while length(milestones) < (nS + 2)
80
81     % random choose points in the map
82     qx = (col-0.5).*rand(1,1) + 0.5;
83     qy = (row-0.5).*rand(1,1) + 0.5;
84     q = [qx, qy];
85
86     % check if q is collision-free using MinDist2Edges
87     if MinDist2Edges(q, map) >= 0.1
88         milestones = [milestones; q];
89     end
90     milestones = unique(milestones, 'rows');
91 end
92
93 % tested several numbers and 6 is the smallest one that has the
    highest prb
94 num_of_neighbours = 10;
95
96 % connect the milestones and hence form the PRM graph
97 for i = (1:length(milestones))
98
99     cur_node = milestones(i,:);
100
101     % remove the current node from the milestones
102     rem_milestones = milestones;
103     rem_milestones(i,:) = [];
104
105     % use k nearest nodes
106     for j = (1:num_of_neighbours)
107
108         % find the closest point
109         [k, dist] = dsearchn(rem_milestones, cur_node);
110         % connect the current node with its closest node and check
            collision
111         current_edge = [cur_node, rem_milestones(k,:) ] ;
112
113         % remove the current closest node from the reaminging of
            milestones
114         rem_milestones(k,:) = [];
115
116         % check collision
117         [ inCollision, edge ] = CheckCollision(current_edge(1:2),
            current_edge(3:4), map);
118
119         if inCollision == 0 && length(edges) == 0
120             edges = [edges; current_edge];
121         else

```

```

122         if inCollision == 0 && length(edges) ~= 0
123
124             % check if current edge already exist and check
              collision
125             [~,index] = ismember(current_edge,edges,"rows");
126             if index == 0
127                 edges = [edges;current_edge];
128             end
129         end
130     end
131 end
132 end
133 % -----end of your PRM generation code -----
134 toc;
135
136 figure(1);
137 plot(milestones(:,1),milestones(:,2),'m. ');
138 if (~isempty(edges))
139     line(edges(:,1:2:3)', edges(:,2:2:4)', 'Color','magenta') % line
        uses [x1 x2 y1 y2]
140 end
141 str = sprintf('Q1 - %d X %d Maze PRM', row, col);
142 title(str);
143 drawnow;
144
145 print -dpng assignment1_q1.png
146
147
148 % =====
149 % Question 2: Find the shortest path over the PRM graph
150 % =====
151 %
152 % Using an optimal graph search method (Dijkstra's or A*) , find the
153 % shortest path across the graph generated. Please code your own
154 % implementation instead of using any built in functions.
155
156 disp('Time to find shortest path');
157 tic;
158
159 % Variable to store shortest path
160 spath = []; % shortest path, stored as a milestone row index sequence
161
162 % -----insert your shortest path finding algorithm here-----
163 % A* implementation with Manhattan distance as the heuristic
164
165 start_node.point = start;
166 start_node.g = 0;

```

```

167 start_node.h = 0;% Manhattan(start,finish);
168 start_node.f = 0;% start_node.g+start_node.h;
169 start_node.parent = [];
170 [~, start_node.position] = ismember(start_node.point, milestones, "
    rows");
171 visited = [start_node.point];
172 open_list = [start_node];
173 closed_list = [];
174
175 while ~isempty(open_list)
176
177     % get the current node that has the least f value using sorting
178     [~,index] = sortrows([open_list.f].');
179     open_list = open_list(index);
180     node = open_list(1);
181
182     % remove current from open and add to close
183     open_list(1) = [];
184     closed_list = [closed_list;node];
185
186     % found the goal
187     if node.point == finish
188         cur_node = node;
189         while ~isempty(cur_node)
190             spath = [spath;cur_node.position];
191             cur_node = cur_node.parent;
192         end
193         display("Congraz!!!");
194         break
195     end
196
197     % generate children
198     children = [];
199     for i=(1:length(edges))
200         if node.point == edges(i,1:2)
201             child.point = edges(i,3:4);
202             children = [children;child];
203         elseif node.point == edges(i,3:4)
204             child.point = edges(i,1:2);
205             children = [children;child];
206         end
207     end
208
209     % loop through children
210     for i = (1:length(children))
211         child = children(i);
212

```



```

213 % check child in open or closed
214 open_list_points = reshape([open_list.point],2,[])';
215 [~,oindex] = ismember(child.point,open_list_points,"rows");
216 close_list_points = reshape([closed_list.point],2,[])';
217 [~,cindex] = ismember(child.point,close_list_points,"rows");
218
219 % if child is in closed
220 if cindex ~= 0
221     continue;
222 end
223
224 % create the f g h values
225 g = node.g + sum(abs(child.point-finish));
226
227 % check if the child has not been visited yet
228 % can be reached with smaller cost from the current
229 if isfield(child,'g')
230     if oindex == 0 || g < child.g
231         child.g = g;
232         child.h = sum(abs(child.point-finish));
233         child.f = child.g+child.h;
234         child.parent = node;
235         [~,child.position] = ismember(child.point, milestones
236             ,"rows");
237
238         if oindex == 0
239             open_list = [open_list;child];
240         else
241             open_list(oindex) = child;
242         end
243     end
244 else
245     if oindex == 0
246         child.g = g;
247         child.h = sum(abs(child.point-finish));
248         child.f = child.g+child.h;
249         child.parent = node;
250         [~,child.position] = ismember(child.point, milestones
251             ,"rows");
252
253         if oindex == 0
254             open_list = [open_list;child];
255         else
256             open_list(oindex) = child;
257         end
258     end
259 end

```

```

258     end
259 end
260
261 if node.point ~= finish
262     disp('Falied');
263 end
264
265 % -----end of shortest path finding algorithm-----
266 toc;
267
268 % plot the shortest path
269 figure(1);
270 for i=1:length(spath)-1
271     plot(milestones(spath(i:i+1),1),milestones(spath(i:i+1),2), 'go-',
272         'LineWidth',3);
273 end
274 str = sprintf('Q2 - %d X %d Maze Shortest Path', row, col);
275 title(str);
276 drawnow;
277 print -dpng assingment1_q2.png
278
279
280 % % =====
281 % % Question 3: find a faster way
282 % % =====
283 % %
284 % % Modify your milestone generation, edge connection, collision
285 % % and/or shortest path methods to reduce runtime. What is the
286 % % largest maze
287 % % for which you can find a shortest path from start to goal in under
288 % % 20
289 % % seconds on your computer? (Anything larger than 40x40 will suffice
290 % % for
291 % % full marks)
292
293
294 % My laptop has i7-7700HQ 4 Core CPU, it can run 44x44 maze under 20
295 % seconds (~16 seconds).
296 % The ECF computer can run 50x50 maze under 20 seconds (~15 seconds)
297
298 row = 44;
299 col = 44;
300 map = maze(row,col);
301 start = [0.5, 1.0];
302 finish = [col+0.5, row];
303 milestones = [start; finish]; % each row is a point [x y] in feasible

```

```

    space
300 edges = []; % each row is should be an edge of the form [x1 y1 x2 y2]
301
302 h = figure(2); clf; hold on;
303 plot(start(1), start(2), 'go')
304 plot(finish(1), finish(2), 'rx')
305 show_maze(map,row,col,h); % Draws the maze
306 drawnow;
307
308 fprintf("Attempting large %d X %d maze... \n", row, col);
309
310 % ————insert your optimized algorithm here———
311 % variables to store PRM components
312 % nS = 5000; % number of samples to try for milestone creation
313 disp("Time to create PRM graph")
314 % ————insert your PRM generation code here———
315 % get as many feasible milestones as nS milestones
316 tic;
317 % setup a list of numbers that will be used for the coordinates
318 % to reduce bad random generated numbers that will cause collision
319 Rx = 0.5:0.5:col;
320 Ry = 1:0.5:row;
321
322 % generate all possible points
323 [X, Y] = meshgrid(Rx, Ry);
324 q = [X(:), Y(:)];
325
326 % only check edges that are close (in terms of x-axis)
327 % to the current milestones
328 k = [];
329 for i = (1:length(q))
330     if mod(i,row*4-2) <= row*2-1 && mod(i,row*4-2) > 0
331         x1 = find(map(:,1)==q(i,1));
332         x2 = find(map(:,3)==q(i,1));
333         x = unique([x1;x2], "rows");
334
335         if ~isempty(x)
336             dx = MinDist2Edges(q(i,:), map(x,:));
337             if dx < 0.1
338                 k = [k;i];
339                 continue
340             end
341         end
342
343         y1 = find(map(:,2)==q(i,2));
344         y2 = find(map(:,4)==q(i,2));
345         y = unique([y1;y2], "rows");

```

```

346
347         if ~isempty(y)
348             dy = MinDist2Edges(q(i,:), map(:,y));
349             if dy < 0.1
350                 k = [k;i];
351             end
352         end
353     end
354 end
355
356 q(k,:) = [];
357 milestones = [finish;q];
358
359 num_of_neighbours = 2;
360
361 % connect the milestones and hence form the PRM graph
362 for i = (1:length(milestones))
363
364     cur_node = milestones(i,:);
365
366     % remove the current node from the milestones
367     rem_milestones = milestones;
368     rem_milestones(i,:) = [];
369
370     if ~isempty(edges)
371         % remove the parent of this node from the milestones
372         [~,index] = ismember(cur_node, edges(3:4), "rows");
373         if index ~= 0
374             rem_milestones(index,:) = [];
375         end
376     end
377
378     % use k nearest nodes
379     for j = (1:num_of_neighbours)
380
381         % this is faster than dsearchn
382         distance=sqrt((rem_milestones(:,1)-cur_node(1)).^2+(
383             rem_milestones(:,2)-cur_node(2)).^2);
384         [C,k] = min(distance);
385         E = rem_milestones(k,:);
386
387         % remove the current closest node from the reaminging of
388         % milestones
389         rem_milestones(k,:) = [];
390
391         % connect the current node with its closest node and check
392         % collision

```

```

390         current_edge = [cur_node,E];
391
392         % check collision
393         [ inCollision , edge ] = CheckCollision(current_edge(1:2) ,
            current_edge(3:4) , map);
394
395         if inCollision == 0 && isempty(edges)
396             edges = [edges;current_edge];
397         else
398             if inCollision == 0 && ~isempty(edges)
399                 edges = [edges;current_edge];
400             end
401         end
402     end
403
404 end
405
406 % -----end of your PRM generation code -----
407
408 % Variable to store shortest path
409 spath = []; % shortest path, stored as a milestone row index sequence
410 % A* implementation with Manhattan distance as the heuristic
411
412 start_node.point = start;
413 start_node.g = 0;
414 start_node.h = 0;% Manhattan(start,finish);
415 start_node.f = 0;% start_node.g+start_node.h;
416 start_node.parent = [];
417 [~, start_node.position] = ismember(start_node.point , milestones , "
    rows");
418 visited = [start_node.point];
419 open_list = [start_node];
420 closed_list = [];
421
422 while ~isempty(open_list)
423
424     % get the current node that has the least f value (faster than
        sort)
425     [~,index] = min([open_list.f]);
426     node = open_list(index);
427
428     % remove current from open and add to close
429     open_list(index) = [];
430     closed_list = [closed_list;node];
431
432     % found the goal
433

```

```

434 if node.point == finish
435     cur_node = node;
436     while ~isempty(cur_node) % this loop is fast, no need to
        optimize
437         spath = [spath;cur_node.position];
438         cur_node = cur_node.parent;
439     end
440     display("Congraz!!!");
441     break
442 end
443
444 % generate children
445 children = [];
446 % if node.point is edges(i,1:2) then its child is edges(i,3:4)
447 node_x_idx = find(edges(:,1)==node.point(1));
448 node_y_idx = find(edges(:,2)==node.point(2));
449 child_idx = intersect(node_x_idx,node_y_idx);
450 for i=(1:length(child_idx))
451     child.point = edges(child_idx(i),3:4);
452     children = [children;child];
453 end
454 % if node.point is edges(i,3:4) then its child is edges(i,1:2)
455 node_x_idx = find(edges(:,3)==node.point(1));
456 node_y_idx = find(edges(:,4)==node.point(2));
457 child_idx = intersect(node_x_idx,node_y_idx);
458 for i=(1:length(child_idx))
459     child.point = edges(child_idx(i),1:2);
460     children = [children;child];
461 end
462
463 % loop through children
464 for i = (1:length(children))
465     child = children(i);
466
467     % check child in open or closed
468     open_list_points = reshape([open_list.point],2,[])';
469     [~,oindex] = ismember(child.point,open_list_points,"rows");
470     close_list_points = reshape([closed_list.point],2,[])';
471     [~,cindex] = ismember(child.point,close_list_points,"rows");
472
473     % if child is in closed
474     if cindex ~= 0
475         continue;
476     end
477
478     % create the f g h values
479     g = node.g + sum(abs(child.point-finish));

```

```

480
481 % check if the child has not been visited yet
482 % can be reached with smaller cost from the current
483 if isfield(child, 'g')
484     if oindex == 0 || g < child.g
485         child.g = g;
486         child.h = sum(abs(child.point-finish));
487         child.f = child.g+child.h;
488         child.parent = node;
489         [~, child.position] = ismember(child.point, milestones
            , "rows");
490
491         if oindex == 0
492             open_list = [open_list; child];
493         else
494             open_list(oindex) = child;
495         end
496     end
497 else
498     if oindex == 0
499         child.g = g;
500         child.h = sum(abs(child.point-finish));
501         child.f = child.g+child.h;
502         child.parent = node;
503         [~, child.position] = ismember(child.point, milestones
            , "rows");
504
505         if oindex == 0
506             open_list = [open_list; child];
507         else
508             open_list(oindex) = child;
509         end
510     end
511 end
512 end
513 end
514 % -----end of your optimized algorithm-----
515 dt = toc;
516
517
518 [~, index] = ismember(finish, milestones, "rows");
519 if node.position ~= index
520     disp('Falied');
521 end
522
523 figure(2); hold on;
524 plot(milestones(:,1), milestones(:,2), 'm. ');

```

```

525 if (~isempty(edges))
526     line(edges(:,1:2:3) ', edges(:,2:2:4) ', 'Color', 'magenta')
527 end
528 if (~isempty(spath))
529     for i=1:length(spath)-1
530         plot(milestones(spath(i:i+1),1), milestones(spath(i:i+1),2), '
                    go-', 'LineWidth', 3);
531     end
532 end
533 str = sprintf('Q3 - %d X %d Maze solved in %f seconds', row, col, dt);
534 title(str);
535
536 print -dpng assignment1_q3.png

```