

ROB521

Assignment 1: PRM for Maze Solving

Jonathan Spraggett
Student number: 1003958737

February 10th 2023

Abstract

In this assignment, I will use the Probabilistic Roadmap (PRM) approach to navigate through a maze environment. The assignment consists of three parts - constructing a graph using the PRM algorithm, finding the shortest path over the graph with the A* algorithm, and optimizing the approach to solving a 44 x 44 maze in 18.81 seconds.

1 Question 1

Construct a PRM graph connecting start and finish nodes

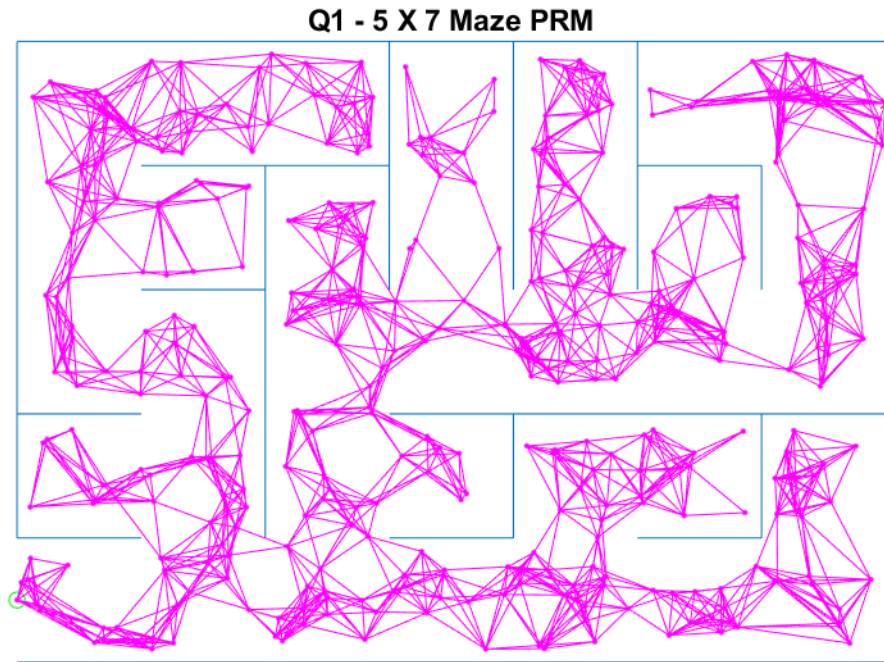


Figure 1

The PRM graph was constructed in 0.121040 seconds and can be found in the figure above. This performance is overall decent for the size of the maze (5x7). The sampling was randomly generated within the bound of the maze. The generated milestones were checked if they were less than 0.1 units away from the walls of the maze to remove potentially colliding milestones. It uses a k nearest-neighbor strategy that links the nearest 10 milestones. 8 was originally chosen for k, but the graph would not always connect to the goal milestone. All links were checked for collision with the maze walls.

2 Question 2

Find the shortest path over the PRM graph

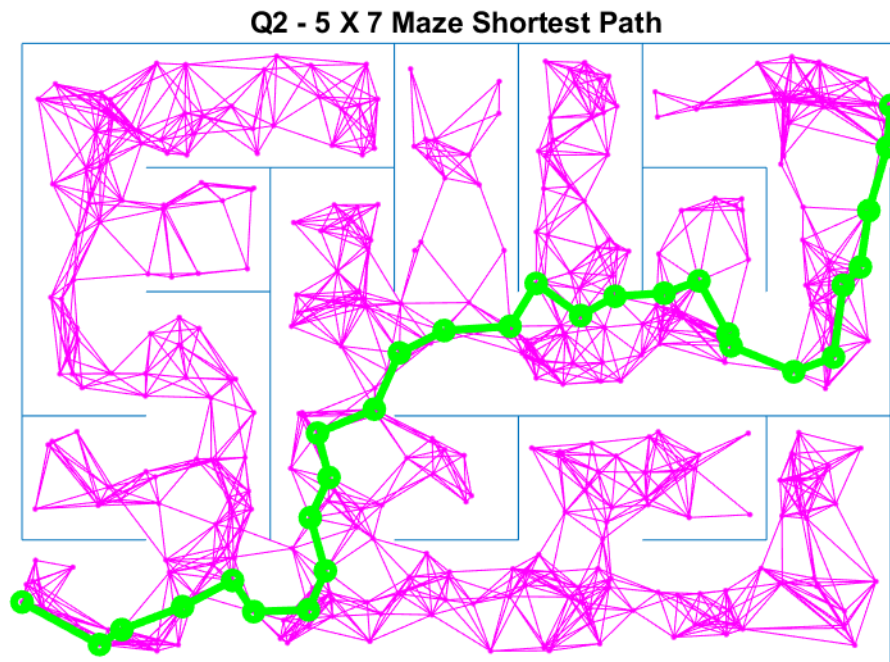


Figure 2

A^* was used as the optimal graph search method to find the shortest path across the graph generated. The shortest path for a maze of size 5x7 was found in 0.293550 seconds and can be seen in the figure above. The main heuristic chosen was the collision-free euclidean distance between the current milestone and the goal milestone. This performance is overall decent for the size of the maze (5x7)

3 Question 3

Optimize previous methods to solve a 40x40 maze in less than 20 second

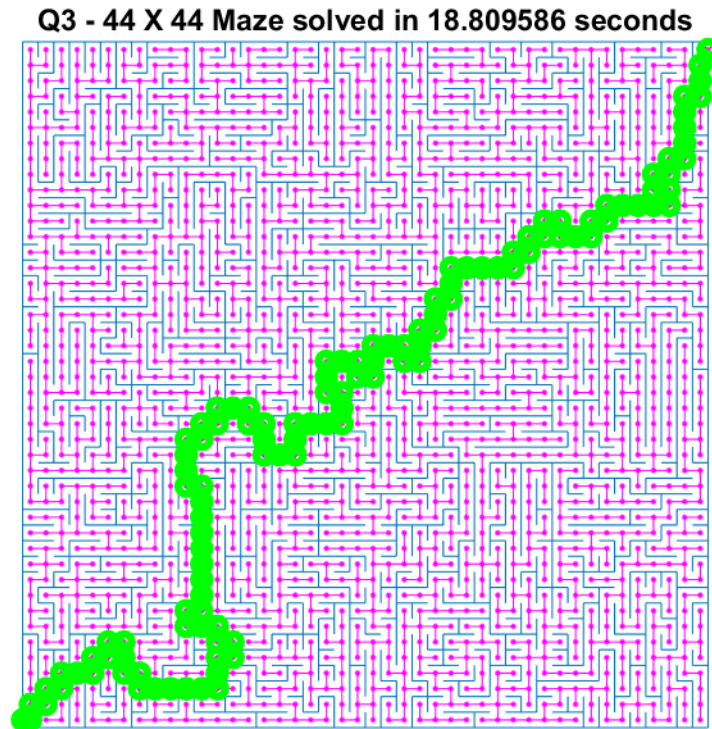


Figure 3

The sampling method was modified to sample the maze in a uniform grid pattern instead of a random sampling. Due to the grid pattern and the proximity of milestones to each other, the number of nearest neighbors linked went from 10 to 6 for increased computation speed while still connecting the start and goal milestones. This approach has been shown to increase the construction of the PRM graph and allow for the start and finish milestones to be connected no matter how large the maze is. The largest maze that could be solved using this new sampling method and the previous A^* search algorithm was 44x44 units. This maze was able to be solved in 18.81 seconds.

Matlab Code

```

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);
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
79
80 % 1. Generate N random points in maze boundary
81 x_high = col+0.5;
82 x_low = 0.5;
83 y_high = row+0.5;
84 y_low = 0.5;
85 x = (x_high-x_low).*rand(nS,1) + x_low;
86 y = (y_high-y_low).*rand(nS,1) + y_low;
87 pts = [x y];
88
89 % 2.remove collided points if its 0.1 nearest to a wall

```

```

90 dist = MinDist2Edges(pts,map);
91 indices = [];
92 count = 1;
93 for i = 1:(nS)
94     if dist(i) < 0.1
95         indices(count) = i;
96         count = count + 1;
97     end
98 end
99 pts(indices,:) = [];
100 milestones = cat(1,milestones,pts);
101
102 % 3. Find nearest 10 neighbors and link them
103 for i = 1:(length(milestones))
104     %[Idx, d] = knnsearch(milestones,milestones(i,:), 'K',8,'Distance','
        minkowski');
105
106     x_d = milestones(i,1) - milestones(:,1);
107     y_d = milestones(i,2) - milestones(:,2);
108     distance = sqrt(x_d.^2 + y_d.^2);
109     [out,Idx] = sort(distance);
110
111     for j = 1:10
112         if CheckCollision(milestones(i,:),milestones(Idx(j)),map) == 0 %
            check for collision in the edge path
113             edges(length(edges) + 1,:) = [milestones(i,:) milestones(Idx(j)
                ,:)]];
114         end
115     end
116 end
117
118
119 % -----end of your PRM generation code -----
120 toc;
121
122 figure(1);
123 plot(milestones(:,1),milestones(:,2),'m. ');
124 if (~isempty(edges))
125     line(edges(:,1:2:3)', edges(:,2:2:4)', 'Color','magenta') % line uses [x1
        x2 y1 y2]
126 end
127 str = sprintf('Q1 - %d X %d Maze PRM', row, col);
128 title(str);
129 drawnow;
130
131 print -dpng assignment1_q1.png

```

```
132
133 % % =====
134 % % Question 2: Find the shortest path over the PRM graph
135 % % =====
136 % %
137 % % Using an optimal graph search method (Dijkstra's or A*) , find the
138 % % shortest path across the graph generated. Please code your own
139 % % implementation instead of using any built in functions.
140
141 disp('Time to find shortest path');
142 tic;
143
144 % Variable to store shortest path
145 spath = []; % shortest path, stored as a milestone row index sequence
146
147
148 % -----insert your shortest path finding algorithm here-----
149 unvisited = 2:length(milestones); % removed start
150 Q_idx = [];
151 Q_cost = [];
152
153 cost_gx = inf(length(milestones), 1); % Remember previous
154
155 % A*
156
157 % Calculate h(x) for start
158 x_d = start(1) - finish(1);
159 y_d = start(2) - finish(2);
160 hx = sqrt(y_d^2 + x_d^2);
161 cost = 0 + hx;
162
163 % Tracks previous obtained milestone
164 previous = -1*ones(length(milestones),1);
165
166 % Add start to Q
167 Q_idx(1) = 1;
168 Q_cost(1) = cost;
169 cost_gx(1) = 0;
170
171 success = false;
172
173 while isempty(Q_idx) == 0
174     % select first from list
175     x_idx = Q_idx(1);
176     x_cost = Q_cost(1);
177     % Delete from list
```



```

178     Q_idx(1) = [];
179     Q_cost(1) = [];
180
181     % Found goal
182     if x_idx == 2
183         success = true;
184         break
185     end
186
187     % Get edges
188     from_edge_idx = find(ismember(edges(:, 1:2), milestones(x_idx,:), 'rows')
189         );
189
190     % find milestone from edges
191     pt_idx = find(ismember(milestones(:, 1:2), edges(from_edge_idx, 3:4), '
192         rows')));
192
193     % Go through all neighbours
194     for i = 1:length(pt_idx)
195         % Check if points are visited or not
196         if ismember(pt_idx(i), unvisited)
197             % remove from list
198             unvisited(unvisited == pt_idx(i)) = [];
199
200             % Calculate g(x) and c(x) between edges
201             x_d = milestones(pt_idx(i), 1) - milestones(x_idx, 1);
202             y_d = milestones(pt_idx(i), 2) - milestones(x_idx, 2);
203             cx = sqrt(x_d^2 + y_d^2);
204             gx = x_cost + cx;
205
206             % only select paths that are more efficient then previous
207             if gx < cost_gx(pt_idx(i))
208                 cost_gx(pt_idx(i)) = gx;
209                 previous(pt_idx(i)) = x_idx;
210
211             % Calculate cost and h(x)
212             x_d = milestones(pt_idx(i), 1) - finish(1);
213             y_d = milestones(pt_idx(i), 2) - finish(2);
214             hx = abs(x_d) + abs(y_d);
215             cost = gx + hx;
216
217             % Adding to Q depends on rank
218             if isempty(Q_cost)
219                 Q_idx(length(Q_idx) + 1) = pt_idx(i);
220                 Q_cost(length(Q_cost) + 1) = cost;
221             else

```

```

222         id = find(Q_cost < cost);
223
224         if isempty(id) % smallest value
225             Q_cost = [cost Q_cost(:,:)];
226             Q_idx = [pt_idx(i) Q_idx(:,:)];
227         else % In the middle
228             Q_cost = [Q_cost(1:id(length(id))) cost Q_cost(id(
229                 length(id)):length(Q_cost))];
230             Q_idx = [Q_idx(1:id(length(id))) pt_idx(i) Q_idx(
231                 id(length(id)):length(Q_idx))];
232
233         end
234     end
235
236 end
237
238 end
239 end
240
241 % create the shortest path
242 if success
243     idx = 2;
244     spath=[spath ; 2];
245     while idx ~= 1
246         spath=[previous(idx) ; spath];
247         idx = previous(idx);
248
249     end
250 else
251     disp("No path exists")
252 end
253
254 % -----end of shortest path finding algorithm-----
255 toc;
256
257 % plot the shortest path
258 figure(1);
259 for i=1:length(spath)-1
260     plot(milestones(spath(i:i+1),1),milestones(spath(i:i+1),2), 'go-', '
261         LineWidth',3);
262 end
263 str = sprintf('Q2 - %d X %d Maze Shortest Path', row, col);
264 title(str);
265 drawnow;

```

```

265
266 print -dpng assingment1_q2.png
267
268 % =====
269 % Question 3: find a faster way
270 % =====
271 %
272 % Modify your milestone generation, edge connection, collision detection
273 % and/or shortest path methods to reduce runtime. What is the largest maze
274 % for which you can find a shortest path from start to goal in under 20
275 % seconds on your computer? (Anything larger than 40x40 will suffice for
276 % full marks)
277
278
279 row = 44;
280 col = 44;
281 map = maze(row,col);
282 start = [0.5, 1.0];
283 finish = [col+0.5, row];
284 milestones = [start; finish]; % each row is a point [x y] in feasible space
285 edges = []; % each row is should be an edge of the form [x1 y1 x2 y2]
286
287 h = figure(2);clf; hold on;
288 plot(start(1), start(2), 'go')
289 plot(finish(1), finish(2), 'rx')
290 show_maze(map,row,col,h); % Draws the maze
291 drawnow;
292
293 fprintf("Attempting large %d X %d maze... \n", row, col);
294 tic;
295 % -----insert your optimized algorithm here-----
296
297 % -----insert your PRM generation code here-----
298 % 1. Generate N points uniformly in maze boundary
299 pts = [];
300 for i = 0:col
301     for j = 0:row
302         pts(length(pts)+1, :) = [i j];
303     end
304 end
305
306 nS = length(pts);
307 % 2.remove collided points if its 0.1 nearest to a wall
308 dist = MinDist2Edges(pts,map);
309 indices = [];
310 count = 1;

```

```

311 for i = 1:(nS)
312     if dist(i) < 0.1
313         indices(count) = i;
314         count = count + 1;
315     end
316 end
317 pts(indices,:) = [];
318 milestones = cat(1,milestones,pts);
319
320 % 3. Find nearest 10 neighbors and link them
321 for i = 1:(length(milestones))
322     % [Idx, d] = knnsearch(milestones,milestones(i,:), 'K',8,'Distance','
    minkowski');
323
324     x_d = milestones(i,1) - milestones(:,1);
325     y_d = milestones(i,2) - milestones(:,2);
326     distance = sqrt(x_d.^2 + y_d.^2);
327     [out,Idx] = sort(distance);
328
329     for j = 1:6
330         if CheckCollision(milestones(i,:),milestones(Idx(j),:),map) == 0 %
            check for collision in the edge path
331             edges(length(edges) + 1,:) = [milestones(i,:) milestones(Idx(j)
                ,:)]];
332         end
333     end
334 end
335
336 % -----end of your PRM generation code -----
337
338
339 % Variable to store shortest path
340 spath = []; % shortest path, stored as a milestone row index sequence
341
342
343 % -----insert your shortest path finding algorithm here-----
344 unvisted = 2:length(milestones); % removed start
345 Q_idx = [];
346 Q_cost = [];
347
348 cost_gx = inf(length(milestones), 1); % Remember previous
349
350 % A*
351
352 % Calculate h(x) for start
353 x_d = start(1) - finish(1);

```

```
354 y_d = start(2) - finish(2);
355 hx = sqrt(y_d^2 + x_d^2);
356 cost = 0 + hx;
357
358 % Tracks previous obtained milestone
359 previous = -1*ones(length(milestones),1);
360
361 % Add start to Q
362 Q_idx(1) = 1;
363 Q_cost(1) = cost;
364 cost_gx(1) = 0;
365
366 success = false;
367
368 while isempty(Q_idx) == 0
369     % select first from list
370     x_idx = Q_idx(1);
371     x_cost = Q_cost(1);
372     % Delete from list
373     Q_idx(1) = [];
374     Q_cost(1) = [];
375
376     % Found goal
377     if x_idx == 2
378         success = true;
379         break
380     end
381
382     % Get edges
383     from_edge_idx = find(ismember(edges(:, 1:2), milestones(x_idx,:), 'rows')
384         );
385
386     % find milestone from edges
387     pt_idx = find(ismember(milestones(:, 1:2), edges(from_edge_idx, 3:4), '
388         rows')));
389
390     % Go through all neighbours
391     for i = 1:length(pt_idx)
392         % Check if points are visited or not
393         if ismember(pt_idx(i), unvisited)
394             % remove from list
395             unvisited(unvisited == pt_idx(i)) = [];
396
397             % Calculate g(x) and c(x) between edges
398             x_d = milestones(pt_idx(i),1) - milestones(x_idx,1);
399             y_d = milestones(pt_idx(i),2) - milestones(x_idx,2);
```

```

398     cx = sqrt(x_d^2 + y_d^2);
399     gx = x_cost + cx;
400
401     % only select paths that are more efficient then previous
402     if gx < cost_gx(pt_idx(i))
403         cost_gx(pt_idx(i)) = gx;
404         previous(pt_idx(i)) = x_idx;
405
406         % Calculate cost and h(x)
407         x_d = milestones(pt_idx(i),1) - finish(1);
408         y_d = milestones(pt_idx(i),2) - finish(2);
409         hx = abs(x_d) + abs(y_d);
410         cost = gx + hx;
411
412         % Adding to Q depends on rank
413         if isempty(Q_cost)
414             Q_idx(length(Q_idx) + 1) = pt_idx(i);
415             Q_cost(length(Q_cost) + 1) = cost;
416         else
417             id = find(Q_cost < cost);
418
419             if isempty(id) % smallest value
420                 Q_cost = [cost Q_cost(:,:)];
421                 Q_idx = [pt_idx(i) Q_idx(:,:)];
422             else % In the middle
423                 Q_cost = [Q_cost(1:id(length(id))) cost Q_cost(id(
424                     length(id)):length(Q_cost))];
425                 Q_idx = [Q_idx(1:id(length(id))) pt_idx(i) Q_idx(
426                     id(length(id)):length(Q_idx))];
427
428             end
429         end
430     end
431 end
432 end
433 end
434 end
435
436 % create the shortest path
437 if success
438     idx = 2;
439     spath=[spath ; 2];
440     while idx ~= 1
441         spath=[previous(idx) ; spath];

```

```
442         idx = previous(idx);
443
444     end
445 else
446     disp("No path exists")
447 end
448
449 % -----end of shortest path finding algorithm-----
450
451 % -----end of your optimized algorithm-----
452 dt = toc;
453
454 figure(2); hold on;
455 plot(milestones(:,1),milestones(:,2),'m. ');
456 if (~isempty(edges))
457     line(edges(:,1:2:3)', edges(:,2:2:4)', 'Color','magenta')
458 end
459 if (~isempty(spath))
460     for i=1:length(spath)-1
461         plot(milestones(spath(i:i+1),1),milestones(spath(i:i+1),2), 'go-', 'LineWidth',3);
462     end
463 end
464 str = sprintf('Q3 - %d X %d Maze solved in %f seconds', row, col, dt);
465 title(str);
466
467 print -dpng assignment1_q3.png
```