ROB521
Assignment 1: PRM for Maze Solving

Jonathan Spraggett
Student number: 1003958737

February 10^{th} 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

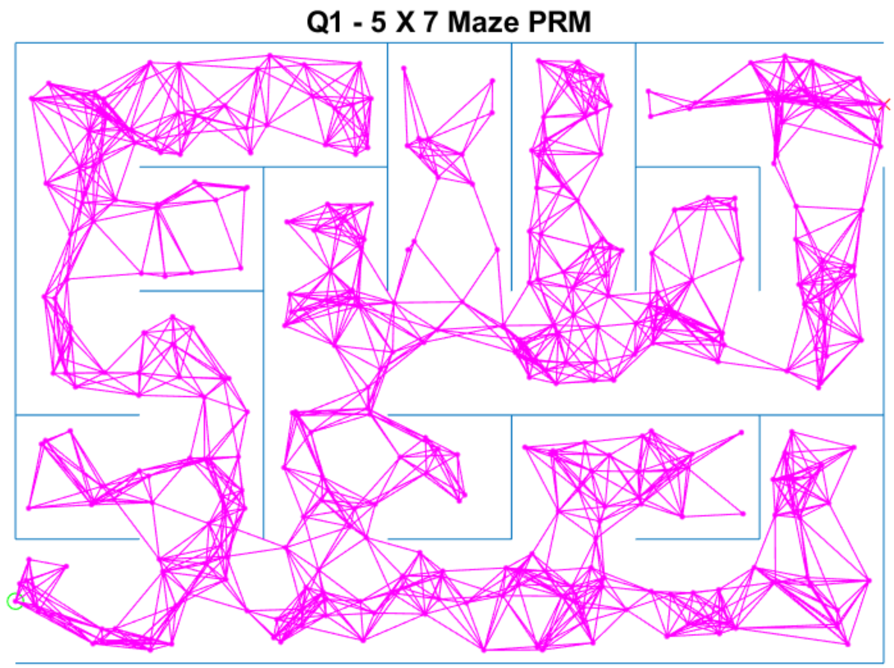

Figure 1

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

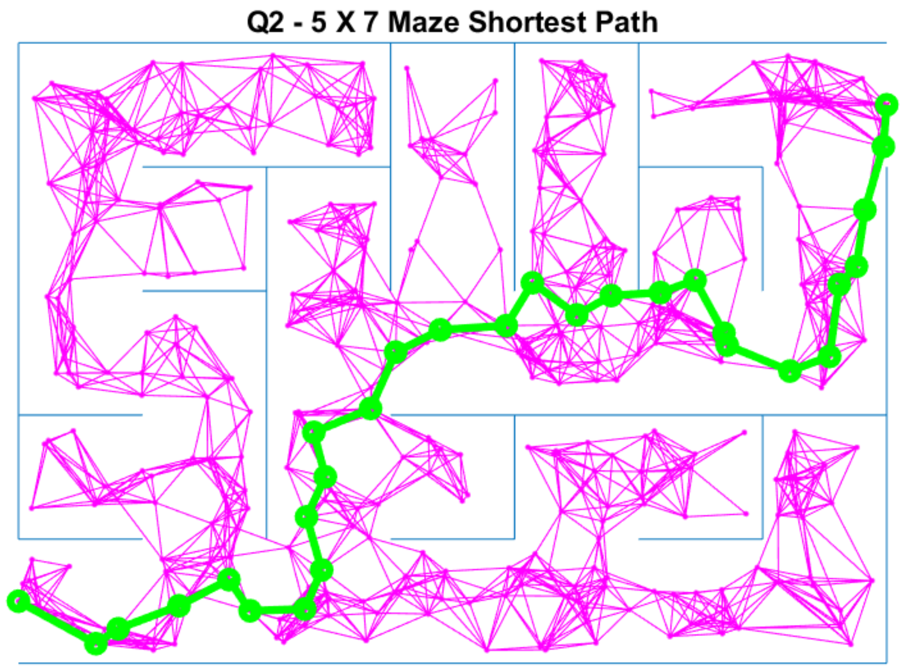

Figure 2

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

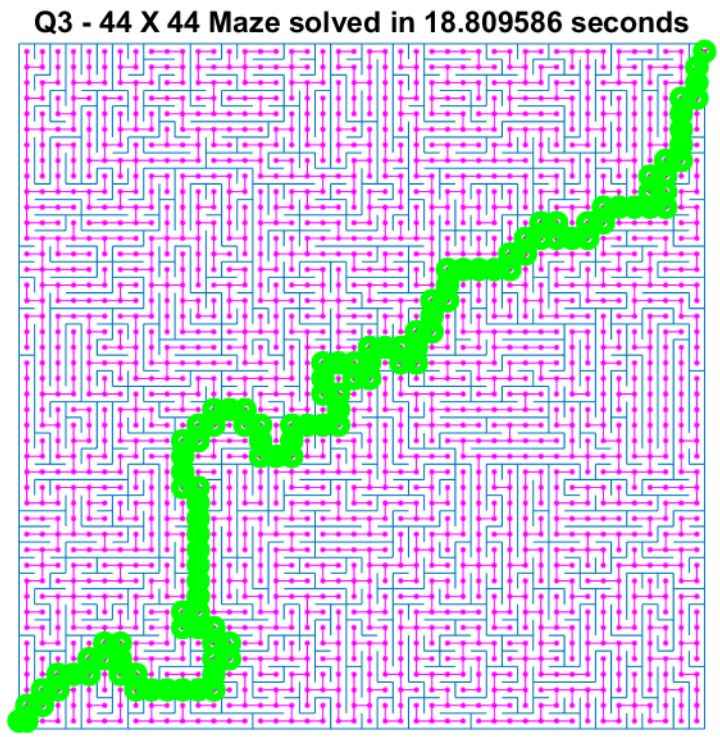

Figure 3

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

```matlab
% ======
% 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---------


% 1. Generate N random points in maze boundary
x_high = col+0.5;
x_low = 0.5;
y_high = row+0.5;
y_low = 0.5;
x = (x_high-x_low).*rand(nS,1) + x_low;
y = (y_high-y_low).*rand(nS,1) + y_low;
pts = [x y];

% 2.remove collided points if its 0.1 nearest to a wall
```

```
dist = MinDist2Edges(pts,map);
indices = [];
count = 1;
for i = 1:(nS)
    if dist(i) < 0.1
        indices(count) = i;
        count = count + 1;
    end
end
pts(indices,:) = [];
milestones = cat(1,milestones,pts);

% 3. Find nearest 10 neighbors and link them
for i = 1:(length(milestones))
    %[Idx, d]  = knnsearch(milestones,milestones(i,:), 'K',8,'Distance','minkowski');

    x_d = milestones(i,1) - milestones(:,1);
    y_d = milestones(i,2) - milestones(:,2);
    distance = sqrt(x_d.^2 + y_d.^2);
    [out,Idx] = sort(distance);

    for j = 1:10
        if CheckCollision(milestones(i,:),milestones(Idx(j),:),map) == 0 % check for collision in the edge path
            edges(length(edges) + 1,:) = [milestones(i,:) milestones(Idx(j),:)];
        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-------
unvisted = 2:length(milestones);  % removed start
Q_idx = [];
Q_cost = [];

cost_gx = inf(length(milestones), 1); % Remember previous

% A*

% Calculate h(x) for start
x_d = start(1) - finish(1);
y_d = start(2) - finish(2);
hx = sqrt(y_d^2 + x_d^2);
cost = 0 + hx;

% Tracks previous obtained milestone
previous = -1*ones(length(milestones),1);

% Add start to Q
Q_idx(1) = 1;
Q_cost(1) = cost;
cost_gx(1) = 0;

success = false;

while isempty(Q_idx) == 0
    % select first from list
    x_idx = Q_idx(1);
    x_cost = Q_cost(1);
    % Delete from list
```

```
    Q_idx(1) = [];
    Q_cost(1) = [];

    % Found goal
    if x_idx == 2
        success = true;
        break
    end

    % Get edges
    from_edge_idx = find(ismember(edges(:, 1:2), milestones(x_idx,:),'rows'));

    % find milestone from edges
    pt_idx = find(ismember(milestones(:, 1:2), edges(from_edge_idx,3:4),'rows'));

    % Go through all neighbours
    for i = 1:length(pt_idx)
        % Check if points are visited or not
        if ismember(pt_idx(i), unvisted)
            % remove from list
            unvisted(unvisted == pt_idx(i)) = [];

            % Calculate g(x) and c(x) between edges
            x_d = milestones(pt_idx(i),1) - milestones(x_idx,1);
            y_d = milestones(pt_idx(i),2) -  milestones(x_idx,2);
            cx = sqrt(x_d^2 + y_d^2);
            gx = x_cost + cx;

            % only select paths that are more efficient then previous
            if gx < cost_gx(pt_idx(i))
                cost_gx(pt_idx(i)) = gx;
                previous(pt_idx(i)) = x_idx;

                % Calculate cost and h(x)
                x_d = milestones(pt_idx(i),1) - finish(1);
                y_d = milestones(pt_idx(i),2) - finish(2);
                hx = abs(x_d) + abs(y_d);
                cost = gx + hx;

                % Adding to Q depends on rank
                if isempty(Q_cost)
                    Q_idx(length(Q_idx) + 1) = pt_idx(i);
                    Q_cost(length(Q_cost) + 1) = cost;
                else
```

```
                    id = find(Q_cost < cost);

                    if isempty(id) % smallest value
                        Q_cost =  [cost Q_cost(:,:)];
                        Q_idx =   [pt_idx(i) Q_idx(:,:)];
                    else % In the middle
                        Q_cost = [Q_cost(1:id(length(id)))  cost  Q_cost(id(length(id)):length(Q_cost))];
                        Q_idx = [Q_idx(1:id(length(id)))  pt_idx(i)  Q_idx(id(length(id)):length(Q_idx))];

                    end

                end
            end


        end
    end
end

% create the shortest path
if success
    idx = 2;
    spath=[spath ; 2];
    while idx ~= 1
        spath=[previous(idx) ; spath];
        idx = previous(idx);

    end
else
    disp("No path exists")
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 = 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);
tic;
% ------insert your optimized algorithm here------

% ------insert your PRM generation code here-------
% 1. Generate N points uniformally in maze boundary
pts = [];
for i = 0:col
    for j = 0:row
        pts(length(pts)+1, :) = [i j];
    end
end

nS = length(pts);
% 2.remove collided points if its 0.1 nearest to a wall
dist = MinDist2Edges(pts,map);
indices = [];
count = 1;
```

```
for i = 1:(nS)
    if dist(i) < 0.1
        indices(count) = i;
        count = count + 1;
    end
end
pts(indices,:) = [];
milestones = cat(1,milestones,pts);

% 3. Find nearest 10 neighbors and link them
for i = 1:(length(milestones))
%     [Idx, d]  = knnsearch(milestones,milestones(i,:), 'K',8,'Distance','minkowski');

    x_d = milestones(i,1) - milestones(:,1);
    y_d = milestones(i,2) - milestones(:,2);
    distance = sqrt(x_d.^2 + y_d.^2);
    [out,Idx] = sort(distance);

    for j = 1:6
        if CheckCollision(milestones(i,:),milestones(Idx(j),:),map) == 0 % check for collision in the edge path
            edges(length(edges) + 1,:) = [milestones(i,:) milestones(Idx(j),:)];
        end
    end
end

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


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


% --------insert your shortest path finding algorithm here--------
unvisted = 2:length(milestones);  % removed start
Q_idx = [];
Q_cost = [];

cost_gx = inf(length(milestones), 1); % Remember previous

% A*

% Calculate h(x) for start
x_d = start(1) - finish(1);
```

```
y_d = start(2) - finish(2);
hx = sqrt(y_d^2 + x_d^2);
cost = 0 + hx;

% Tracks previous obtained milestone
previous = -1*ones(length(milestones),1);

% Add start to Q
Q_idx(1) = 1;
Q_cost(1) = cost;
cost_gx(1) = 0;

success = false;

while isempty(Q_idx) == 0
    % select first from list
    x_idx = Q_idx(1);
    x_cost = Q_cost(1);
    % Delete from list
    Q_idx(1) = [];
    Q_cost(1) = [];

    % Found goal
    if x_idx == 2
        success = true;
        break
    end

    % Get edges
    from_edge_idx = find(ismember(edges(:, 1:2), milestones(x_idx,:),'rows')
        );

    % find milestone from edges
    pt_idx = find(ismember(milestones(:, 1:2), edges(from_edge_idx,3:4),'
        rows'));

    % Go through all neighbours
    for i = 1:length(pt_idx)
        % Check if points are visited or not
        if ismember(pt_idx(i), unvisted)
            % remove from list
            unvisted(unvisted == pt_idx(i)) = [];

            % Calculate g(x) and c(x) between edges
            x_d = milestones(pt_idx(i),1) - milestones(x_idx,1);
            y_d = milestones(pt_idx(i),2) -  milestones(x_idx,2);
```

```
            cx = sqrt(x_d^2 + y_d^2);
            gx = x_cost + cx;

            % only select paths that are more efficient then previous
            if gx < cost_gx(pt_idx(i))
                cost_gx(pt_idx(i)) = gx;
                previous(pt_idx(i)) = x_idx;

                % Calculate cost and h(x)
                x_d = milestones(pt_idx(i),1) - finish(1);
                y_d = milestones(pt_idx(i),2) - finish(2);
                hx = abs(x_d) + abs(y_d);
                cost = gx + hx;

                % Adding to Q depends on rank
                if isempty(Q_cost)
                    Q_idx(length(Q_idx) + 1) = pt_idx(i);
                    Q_cost(length(Q_cost) + 1) = cost;
                else
                    id = find(Q_cost < cost);

                    if isempty(id) % smallest value
                        Q_cost =  [cost Q_cost(:,:)];
                        Q_idx =   [pt_idx(i) Q_idx(:,:)];
                    else % In the middle
                        Q_cost = [Q_cost(1:id(length(id)))  cost  Q_cost(id(
                           length(id)):length(Q_cost))];
                        Q_idx = [Q_idx(1:id(length(id)))  pt_idx(i)  Q_idx(
                           id(length(id)):length(Q_idx))];

                    end

                end
            end


        end
    end
end

% create the shortest path
if success
    idx = 2;
    spath=[spath ; 2];
    while idx ~= 1
        spath=[previous(idx) ; spath];
```

```
        idx = previous(idx);

    end
else
    disp("No path exists")
end

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

% ----------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
```