036044 - תכן תנועת רובוטים וניווט ע"י חיישנים – אביב

פרויקט - חלק ראשון

אור קוזלובסקי – 305447476

שיר קוזלובסקי - 204321228

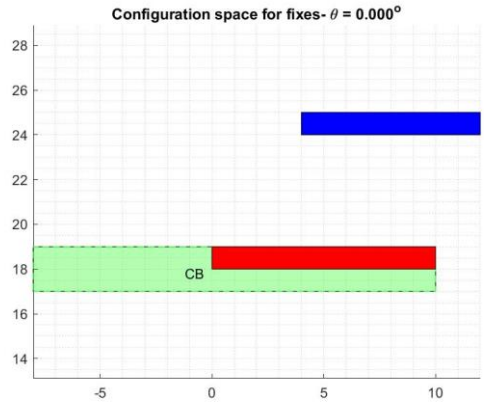
בחלק זה בפרויקט נציג מימוש של אלגוריתם לחישוב מרחב הקונפיגורציה של גוף קשיח (רובוט). המימוש נעשה בתוכנת מטלב בשימוש ב-$Object\ Oriented$.

המטרה היא לפתח אלגוריתם לחישוב מסלול של רובוט מצולע קמור מנקודת התחלה ידועה באוריינטציה ידועה לנקודת סיום ידועה, גם היא באוריינטציה ידועה. בעולם דו-ממדי המכיל מכשולים מצולעים קמורים ועל בסיס כך נבחרה שיטת המימוש, החלוקה למחלקות ולפונקציות.

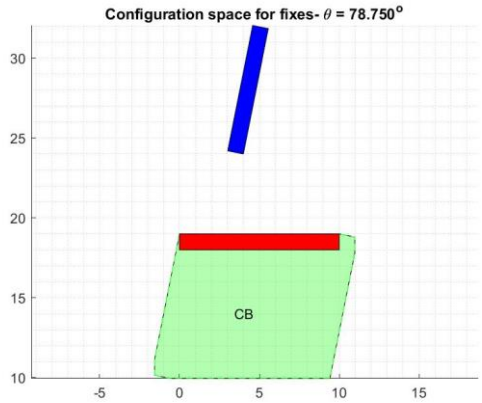
<u>חלק א</u>

מימשנו את חלק זה בעזרת אלגוריתם מיזוג המעגלים. חישבנו את המכשול במרחב הקונפיגורציה עבור חתכי שווי θ שונים כאשר נתון מכשול אחד ידוע באוריינטציה ידועה וקבועה ורובוט ידוע באוריינטציה קבועה לכל תחריש, אך משתנה בין תרחיש לתרחיש.

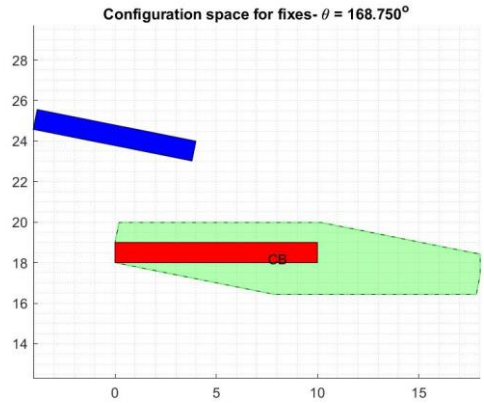
נציג מספר דוגמאות לגרפים המתקבלים בחלק זה כאשר כל תרחיש מתאר זווית שונה בה מצוי הרובוט עבור הזוויות $0°, 78.75°, 168.75°, 348.75°$.



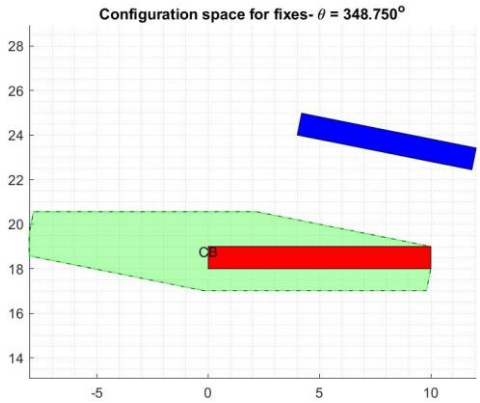
איור 1 – עבור $\theta = 0°$. מכשול באדום. רובוט בכחול. המכשול במרחב הקונפיגורציה עבר חתך שווה θ בפוליגון ירוק מקווקו.



איור 2 - עבור $\theta = 78.75°$. מכשול באדום. רובוט בכחול. המכשול במרחב הקונפיגורציה עבר חתך שווה θ בפוליגון ירוק מקווקו.



איור 1 - עבור $\theta = 168.75°$. מכשול באדום. רובוט בכחול. המכשול במרחב הקונפיגורציה עבר חתך שווה θ בפוליגון ירוק מקווקו.



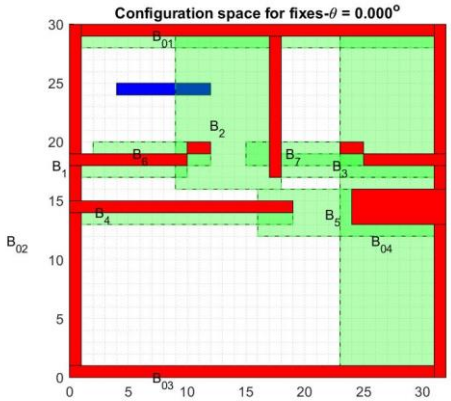
איור 2 - עבור $\theta = 348.75°$. מכשול באדום. רובוט בכחול. המכשול במרחב הקונפיגורציה עבר חתך שווה θ בפוליגון ירוק מקווקו.

ניתן לראות כי ככל שהרובוט קרוב בזוויתו לאורנטציה של המכשול, מכשול מרחב הקונפיגורציה מינימלי. דבר התואם את ההיגיון.

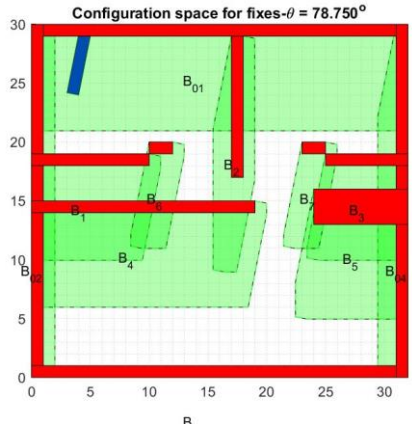
חלק ב

חלק זה הינו הרחבה של חלק א. בחלק זה נתונים מספר מכשולים בגדלים ידועים ובאוריינטציות ידועות. כאשר גם כאן בכל תרחיש הרובוט מצוי באוריינטציה שונה. נחשב את מרחב הקונפיגורציה והמכשולים בו עבור חתכי θ שונים.

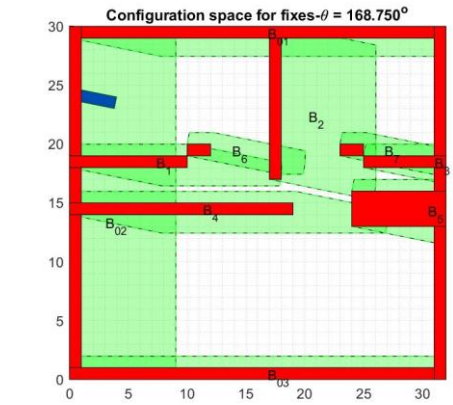
נציג מספר דוגמאות לגרפים המתקבלים בחלק זה כאשר כל תרחיש מתאשר זווית שונה בה מצוי הרובוט: $0°, 78.75°, 168.75°, 348.75°$.



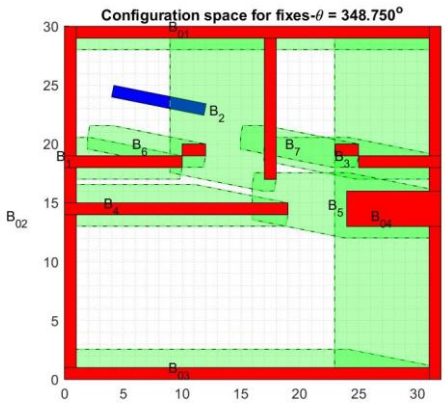
איור 5 - עבור $\theta = 0°$. מכשול באדום. רובוט בכחול. המכשול במרחב הקונפיגורציה עבר חתך שווה θ בפוליגון ירוק מקווקו.



איור 6 - עבור $\theta = 78.75°$. מכשול באדום. רובוט בכחול. המכשול במרחב הקונפיגורציה עבר חתך שווה θ בפוליגון ירוק מקווקו.



איור 7 - עבור $\theta = 168.75°$. מכשול באדום. רובוט בכחול המכשול במרחב הקונפיגורציה עבר חתך שווה θ בפוליגון ירוק מקווקו.

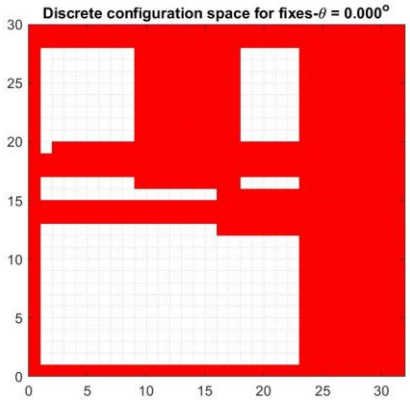


איור 3 - עבור $\theta = 348.75°$. מכשול באדום. רובוט בכחול. המכשול במרחב הקונפיגורציה עבר חתך שווה θ בפוליגון ירוק מקווקו.

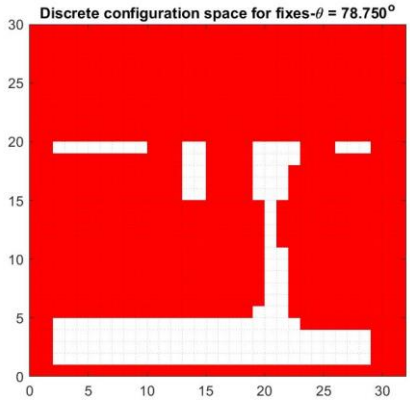
ניתן לראות כי בכל חתך θ מכשולי מרחב הקונפיגורציה משתנים הן מבחינת גודלם והן בצורתם.

חלק ג

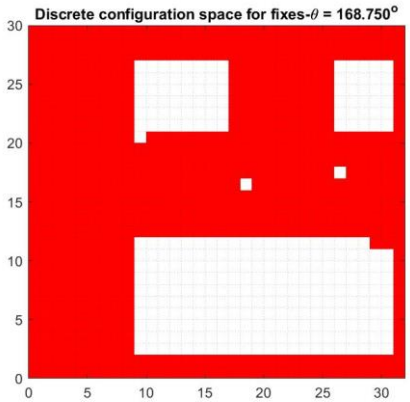
בחלק זה ביצענו דיסקרטיצזיה של מרחב הקונפיגורציה שחישבנו בחלק ב'.
הדיסקריטיזציה היא ברזולוציה של 1:1. לשם ההמרה של המרחב הרציף למרחב הדיסקרטי מימשנו פונקציה, *drwa_line*, אשר מוסיפה למטריצה קו דיסקרטי בין שתי אינדקסים במטריצה. כך המרנו כל שפת מכשול לשקו דיסקרטי ולאחר שהמרנו את כל שפות המכשול מלאנו את השטח הכלוא בפוליגון שנוצר באמצעות הפונקציה *fill*. מכיוון שהמכשולים סומנו כ-'1' במטריצה ומרחב בקונפיגורציה החופשית כ-'0' יכלנו להשתמש בחשבון בוליאני פשוט לאיחוד מרחבי קונפיגורציה שונים.



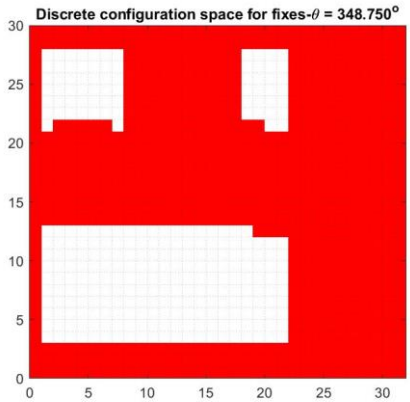
איור 9 - עבור $\theta = 0°$. מכשולי מרחב הקונפיגורציה באדום ומרחב הקונפיגורציה החופשי בלבן.



איור 10 - עבור $\theta = 78.75°$. מכשולי מרחב הקונפיגורציה באדום ומרחב הקונפיגורציה החופשי בלבן.



איור 11 - עבור $\theta = 168.75°$. מכשולי מרחב הקונפיגורציה באדום ומרחב הקונפיגורציה החופשי בלבן.



איור 12 - עבור $\theta = 348.75°$. מכשולי מרחב הקונפיגורציה באדום ומרחב הקונפיגורציה החופשי בלבן.

ניתן לראות את התאימות בין חלק ב לחלק ג מבחינת הגרפים. החלקים האסור המוצגים בחלק ג זהים לאלו בחלק הקודם. בשונה מחלק ב' בו האובייקטים הוצגו באופן רציף, בחלק זה ישנו ייצוג דיסקרטי של מרחב הקונפיגורציה.

אופן המימוש

לצורך מימוש המשימה השתמשנו בכמה מחלקות:

1. מחלקת Map_object - מממשת אבסטרקציה של אובייקט מפה כללי (רובוט או מכשול). מקבלת רשימה של קודקודים, כיוון של הנורמלים (כדי לאפשר נורמלים כלפי פנים האובייקט עבור הרובוט וכלפי חוץ עבור המכשולים) ושם של אובייקט. מאפשר חישוב של הנורמלים של האובייקט והדפסה של האובייקט.

```
classdef Map_Object < handle
    properties
        vertices
        norms
        InOut
        name
    end

    methods
        function obj = Map_Object(vertices, InOut, name)
            % assumse the vertices are counter-clock wish order
            set_vertices(obj, vertices);
            obj.InOut = InOut;
            calc_norms(obj);
            obj.name = name;
        end

        function set_vertices(obj, vertices)
            obj.vertices = vertices;
            obj.vertices(end+1,:) = vertices(1,:);
        end

        function calc_norms(obj)
            % Inout = 1 means out mormals.
            % Inout = -1 means in mormals.
            num = size(obj.vertices,1);
            obj.norms = zeros(num-1, 5);
            for i=1:(num-1)
                x = obj.vertices(i:i+1,1);
                y = obj.vertices(i:i+1,2);
                obj.norms(i,:) = get_norm(x,y,obj.InOut);
            end

            function obj_norm = get_norm(x,y,InOut)
                dy = diff(y);
                dx = diff(x);
                m = (dy/dx);
                % Slope of new line
                L = 0.3*sqrt(dy^2+dx^2);
                minv = -1/m;
                if abs(minv) == Inf
                    obj_norm = [atan2d(L*sign(minv)*sign(InOut),0) , mean(x), mean(x), mean(y), mean(y)
                                L*sign(minv)*sign(InOut)];
                else
                    obj_norm = [atan2d(L*minv*sign(dy)*sign(InOut),L*sign(dy)*sign(InOut)), mean(x), mean(x)-
                                L*sign(dy)*sign(InOut), mean(y), mean(y)-L*minv*sign(dy)*sign(InOut)]; %#ok<*CPROP>
                end
            end
        end

        function add_label_to_plot(obj)
            ver_x = obj.vertices(:,1);
            ver_y = obj.vertices(:,2);
            text(mean(ver_x), mean(ver_y), obj.name , 'HorizontalAlignment' , 'center' , 'VerticalAlignment' , 'middle')
        end

        function print_obstacle(obj, with_norms)
            ver_x = obj.vertices(:,1);
            ver_y = obj.vertices(:,2);

            if obj.InOut == -1
                fill(ver_x,ver_y,'b');
                line(ver_x, ver_y, 'Color', 'k');
            elseif obj.InOut == 1
                fill(ver_x,ver_y,'r');
                line(ver_x, ver_y, 'Color', 'k');
            else
                fill(ver_x,ver_y,'g', 'facealpha',.3);
                line(ver_x, ver_y, 'Color', 'g', 'LineStyle', '--');
            end

            if with_norms
                for i=1:size(obj.norms,1)
                    line(obj.location(1) + [obj.norms(i,2); obj.norms(i,3)], obj.location(2) + [obj.norms(i,4);
                         obj.norms(i,5)], 'Color', 'r');
                end
            end
        end
    end
end
```

2. מחלקת $Robot$ – הרחבה של מחלקת Map_object המוסיפה לאבסטרקציה תכונה של זווית הניתנת לשינוי. המחלקה מממשת סיבוב של אובייקט באמצעות מטריצת סיבוב.

```
classdef Robot < Map_Object
    properties
        theta
        org_vertices
    end

    methods
        function obj = Robot(vertices, theta, name)
            % theta needs to be in degree
            % theta > 0 - counterclockwise
            obj@Map_Object(vertices, -1, name);
            obj.org_vertices = vertices;
            set_theta(obj, theta)
        end

        function set_theta(obj, theta)
            obj.theta = theta;
            obj.set_vertices(rotate_robot(obj.org_vertices, theta));
            calc_norms(obj)

            function new_vertices = rotate_robot(vertices, theta)
                R = [cosd(theta) -sind(theta); sind(theta) cosd(theta)];
                robot_dim = vertices - vertices(1,:);
                new_vertices = (R*robot_dim')' + vertices(1,:);
            end
        end
    end
end
```

3. מחלקת Map – המחלקה הראשית המסמלת מרחב ניווט. מכילה מופעים של מחלקת $robot$ ומחלקת Map_object. מקבל רשימה של קודקודי רובוט, זווית התחלתית ורשימה של רשימות קודקודי מכשולים. הפונקציות המרכזיות הן חישוב מכשולי קונפיגורצית מרחב ודיסקריטיזציה של המרחב הרציף למרחב בדיד. לאחר אתחול מרחב הניווט המחלקה מאפשרת שינוי של מרחב המצב ביעילות ונוחות.

```
classdef Map < handle
    properties
        map_size

        matrix_map
        obstacls
        c_obstacls
        robot

        non_obstacle_i
        obstacle_i
        robot_i
        wall_i
    end

    methods
        function obj = Map(robot, theta, obstacls)
            %UNTITLED Construct an instance of this class
            %   Detailed explanation goes here

            obj.map_size = [30 32]; % detarmine in the question

            %inside map index
            obj.obstacle_i = 1;
            obj.non_obstacle_i = 0;
            obj.robot_i = 2;

            % create the robot
            obj.robot = Robot(robot{1}, theta, robot{2});

            % creat the map obstacles and c-obstacles
            obj.obstacls = cell(1,length(obstacls));

            % saving the original obstacles objects and the c_obstacls list
            % for print_map
            for i=1:length(obstacls)
                obj.obstacls{i} = Map_Object(obstacls{i}{1},1, obstacls{i}{2});
            end

            calc_c_obstacles(obj);
        end

        function mat = init_matrix_map(obj)
            mat = zeros(obj.map_size(1), obj.map_size(2));
        end

        function update_matrix_map(obj)
            % the 'imfill' works on binery images. thus we fill the matrix
            % map with the robot (index '2') only after we activate the 'imfill'.
            obj.matrix_map = obj.init_matrix_map();

            for i=1:length(obj.obstacls)
                tmp_mat = obj.draw_object_on_matrix_map(obj.c_obstacls{i}.vertices, obj.obstacle_i);
                obj.matrix_map = min((obj.matrix_map + imfill(tmp_mat)),1);
```

```
            end

%               tmp_mat = obj.draw_object_on_matrix_map(obj.robot.vertices, obj.robot_i);
%               obj.matrix_map = max(obj.matrix_map,tmp_mat);
        end

        %%%% print functions %%%
        function fig = get_map(obj, show)
            with_norms = false; %set to true will print red lines to mark the normals from each line.

            fig = figure('visible', show); hold on;
            obj.robot.print_obstacle(with_norms);
            for i=1:length(obj.c_obstacls)
                obj.c_obstacls{i}.print_obstacle(with_norms);
            end
            for i=1:length(obj.obstacls)
                obj.obstacls{i}.print_obstacle(with_norms);
            end
            for i=1:length(obj.c_obstacls)
                obj.c_obstacls{i}.add_label_to_plot;
            end
            axis equal;
            fig.Children.XLim = [0 obj.map_size(2)];
            fig.Children.YLim = [0 obj.map_size(1)];
            grid minor; title(['Configuration space for fixes-\theta = ' num2str(obj.robot.theta, '%.3f') '^o']);
        end

        function matrix_map = get_map_matrix(obj, show)
            update_matrix_map(obj);

            c = [1 1 1; 0 1 0; 1 0 0]; %obstacles in red, wall in black and robot in blue;

            matrix_map = figure('visible', show);
            [a,b] = size(obj.matrix_map);
            imagesc(0.5,0.5, obj.matrix_map); ax = gca; ax.YDir = 'normal';
            grid minor; axis equal; xlim([0 b]); ylim([0 a]); title(['Discrete configuration space for fixes-\theta = '
num2str(obj.robot.theta, '%.3f') '^o']);
            colormap(c);
        end

        function print_maps(obj)
            get_map(obj, 'on');
            get_map_matrix(obj, 'on');
        end

        %%%% draw objects and lines %%%%
        function mat = draw_object_on_matrix_map(obj, map_obj, val)
            mat = init_matrix_map(obj);
            %the objects vertices are initilized so the last vertix is the
            %same as the first so a line will be draw between them in this
            %loop - thats why the loop is with '-1';
            max_x = max(map_obj(:,1));
            max_y = max(map_obj(:,2));
            for i=1:length(map_obj)-1
                mat = obj.drwa_line(mat, floor(map_obj(i,:)+1), floor(map_obj(i+1,:)+1), val, [max_x max_y]);
            end
        end

        function mat = drwa_line(obj, mat, start_point, end_point, val, max_val)
            start_point = convret_point_to_map_size_point(obj, start_point, max_val);
            end_point = convret_point_to_map_size_point(obj, end_point, max_val);

            nPoints = 100;
            rIndex = round(linspace(start_point(2), end_point(2), nPoints));  % Row indices
            cIndex = round(linspace(start_point(1), end_point(1), nPoints));  % Column indices
            index = sub2ind(size(mat), rIndex, cIndex);     % Linear indices
            mat(index) = val;
        end

        function map_point = convret_point_to_map_size_point(obj, point, max_val)
            map_point = max([min([point(1) obj.map_size(2) max_val(1)]) min([point(2) obj.map_size(1) max_val(2)])],1);
        end


        %%%% c obstacles %%%%
        function calc_c_obstacles(obj)
            obj.c_obstacls = cell(1,length(obj.obstacls));
            for i=1:length(obj.obstacls)
                obj.c_obstacls{i} = Map_Object(obj.calc_c_obstacle(obj.obstacls{i}), 0, obj.obstacls{i}.name);
            end
        end

        function c_obstacle = calc_c_obstacle(obj, obstacle)
            robot_dim = obj.robot.vertices - obj.robot.vertices(1,:); %the method works with the robot relative vertices (the
robot dims)
            m = size(obj.robot.norms,1);
            n = size(obstacle.norms,1);

            var_axes = [1:m 1:n; ones(1,m), ones(1,n)*2; obj.robot.norms(:,1)', obstacle.norms(:,1)']; %help matrix for the
method implementation.
            [sort_var_axes(3,:), i] = sort(var_axes(3,:));
            sort_var_axes(1:2,:) = var_axes(1:2,i);

            x = find(sort_var_axes(2,:) == 2,1);
            y = find(sort_var_axes(2,:) == 1,1);
            sort_var_axes = [sort_var_axes, sort_var_axes(:,1:x+1) + [0 0 360]'];

            c_obstacle = [];
            tmp = [];

            for i=y:length(sort_var_axes)-1
                if sort_var_axes(2,i) == 1
```

```
                    tmp = [tmp; robot_dim(sort_var_axes(1,i),:)];
                else
                    if sort_var_axes(3,i) == sort_var_axes(3,i-1)
                        c_obstacle = [c_obstacle; obstacle.vertices(sort_var_axes(1,i),:) - tmp];
                    else
                        tmp = [tmp; robot_dim(sort_var_axes(1,i+1),:)];
                        c_obstacle = [c_obstacle; obstacle.vertices(sort_var_axes(1,i),:) - tmp];
                    end
                    tmp = [];
                end
            end
        end
    end
end
```

לאחר המימוש של מחלקות אלה ניתן היה לענות בצורה פשוטה יחסית על השאלות בפרויקט:

שאלה 1:

```
%% Initialization
close all; clear; clc;
warning('off' ,'images:initSize:adjustingMag');
robot = {[0 0; 8 0; 8 1; 0 1] + [4 24], 'robot'};
B1 = {[0 0; 10 0; 10 1; 0 1]+[0 18], 'B_1'};

%% calc all layers
[matrix_map, maps] = Q1_(robot,B1);

%% show and save relevant layers
maps{1}.Visible = 'on';
matrix_map{1}.Visible = 'on';
maps{8}.Visible = 'on';
matrix_map{8}.Visible = 'on';
maps{16}.Visible = 'on';
matrix_map{16}.Visible = 'on';
maps{32}.Visible = 'on';
matrix_map{32}.Visible = 'on';

saveas(maps{1},'graphs\Q1\map1.jpg');
saveas(matrix_map{1},'graphs\Q1\matrix_map1.jpg');
saveas(maps{8},'graphs\Q1\map8.jpg');
saveas(matrix_map{8},'graphs\Q1\matrix_map8.jpg');
saveas(maps{16},'graphs\Q1\map16.jpg');
saveas(matrix_map{1},'graphs\Q1\matrix_map16.jpg');
saveas(maps{32},'graphs\Q1\map32.jpg');
saveas(matrix_map{32},'graphs\Q1\matrix_map32.jpg');

%%
function [matrix_map, maps] = Q1_(robot,obstacle)
dt= 360/32;
maps = cell(1,32);
matrix_map = cell(1,32);
i=0;

obstacle_obj = {obstacle};

% initilaize map object
obj = Map(robot, 0, obstacle_obj);

% calc CB for each theta and get the discrete and continues maps
for theta=0:dt:360-dt
    i = i + 1;
    obj.robot.set_theta(theta);
    obj.calc_c_obstacles;
    maps{i} = obj.get_map('off');
    matrix_map{i} = get_map_matrix(obj, 'off');
end

end
```

שאלה 2:

```
%% Initialization
warning('off' ,'images:initSize:adjustingMag');
close all; clear; clc;
robot = {[0 0; 8 0; 8 1; 0 1] + [4 24], 'robot'};
B0_1 = {[0 0; 32 0; 32 1; 0 1]+[0 29], 'B_{01}'};
B0_2 = {[0 0; 1 0; 1 30; 0 30]+[0 0], 'B_{02}'};
B0_3 = {[0 0; 32 0; 32 1; 0 1]+[0 0], 'B_{03}'};
B0_4 = {[0 0; 1 0; 1 30; 0 30]+[31 0], 'B_{04}'};

B1 = {[0 0; 10 0; 10 1; 0 1]+[0 18], 'B_1'};
B2 = {[0 0; 1 0; 1 12; 0 12]+[17 17], 'B_2'};
B3 = {[0 0; 7 0; 7 1; 0 1]+[25 18], 'B_3'};
B4 = {[0 0; 19 0; 19 1; 0 1]+[0 14], 'B_4'};
B5 = {[0 0; 8 0; 8 3; 0 3]+[24 13], 'B_5'};
B6 = {[0 0; 2 0; 2 1; 0 1]+[10 19], 'B_6'};
B7 = {[0 0; 2 0; 2 1; 0 1]+[23 19], 'B_7'};

obstacles = {B0_1, B0_2, B0_3, B0_4, B1, B2, B3, B4, B5, B6, B7};
```

```
%% calc all layers
[matrix_map, maps] = Q2_(robot,obstacles);
maps{1}.Visible = 'on';
matrix_map{1}.Visible = 'on';
maps{8}.Visible = 'on';
matrix_map{8}.Visible = 'on';
maps{16}.Visible = 'on';
matrix_map{16}.Visible = 'on';
maps{32}.Visible = 'on';
matrix_map{32}.Visible = 'on';

%% show and save relevant layers
saveas(maps{1},'graphs\Q2\map1.jpg');
saveas(matrix_map{1},'graphs\Q2\matrix_map1.jpg');
saveas(maps{8},'graphs\Q2\map8.jpg');
saveas(matrix_map{8},'graphs\Q2\matrix_map8.jpg');
saveas(maps{16},'graphs\Q2\map16.jpg');
saveas(matrix_map{1},'graphs\Q2\matrix_map16.jpg');
saveas(maps{32},'graphs\Q2\map32.jpg');
saveas(matrix_map{32},'graphs\Q2\matrix_map32.jpg');

%%
function [matrix_map, maps] = Q2_(robot, obstacles)
dt= 360/32;
maps = cell(1,32);
matrix_map = cell(1,32);
i=0;

% initilaize map object
obj = Map(robot, 0, obstacles);

% calc CB for each theta and get the discrete and continues maps
for theta=0:dt:360-dt
    i = i + 1;
    obj.robot.set_theta(theta);
    obj.calc_c_obstacles;
    maps{i} = obj.get_map('off');
    matrix_map{i} = get_map_matrix(obj, 'off');
end

end
```