MINI PROJECT – PART 1

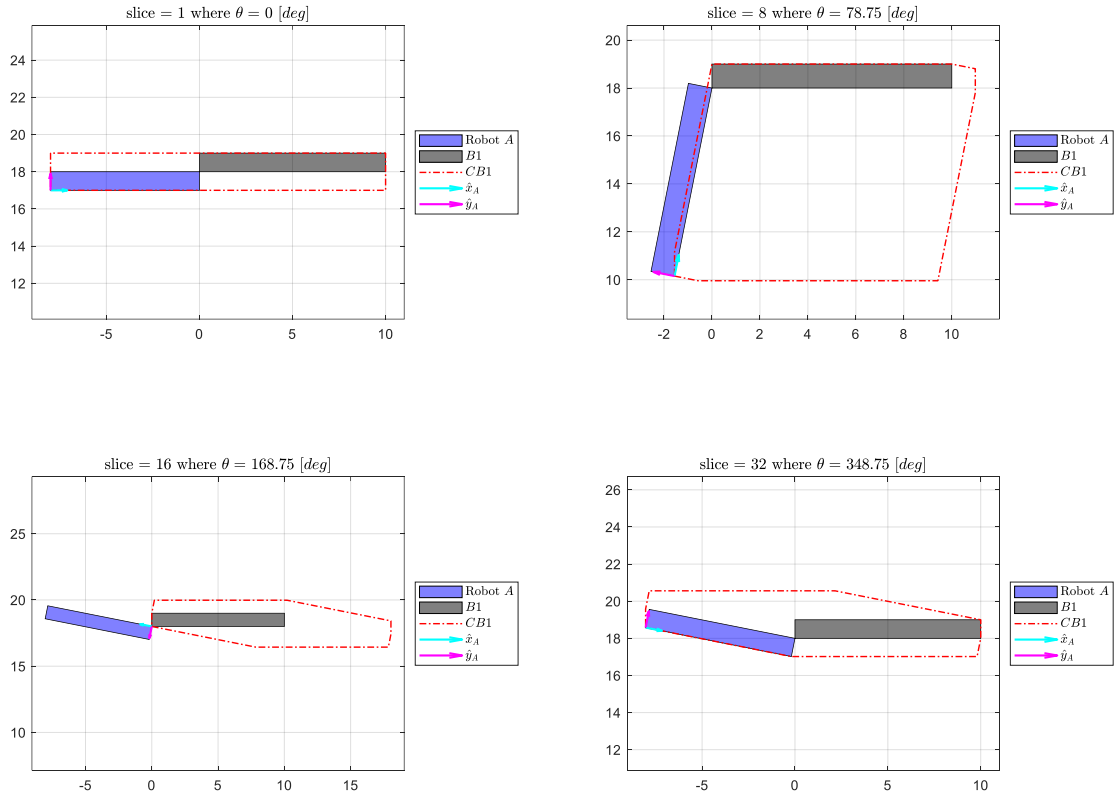
Robot Navigation

Table of Contents

1 C-space obstacle program

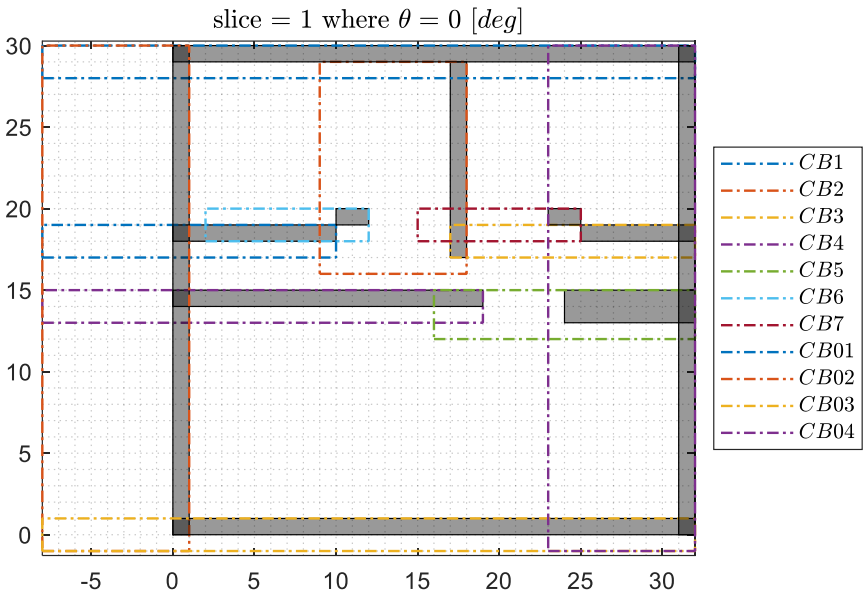
We used the Lozano Perez algorithm mentioned in Latombe Ch. 3. We also used a built-in convex hulls function in MATLAB to reorder the indices of the C-space obstacle to easily plot it. Since the resulting CB is also a polygon, this does not change the results. The selected slices are given below.

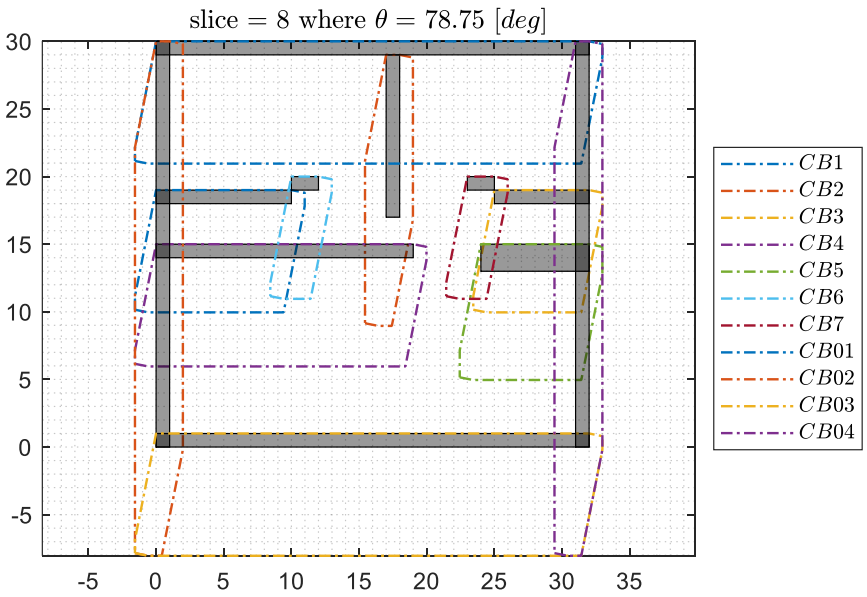
We plotted the robot A with its frame in some position on CB to illustrate its meaning.

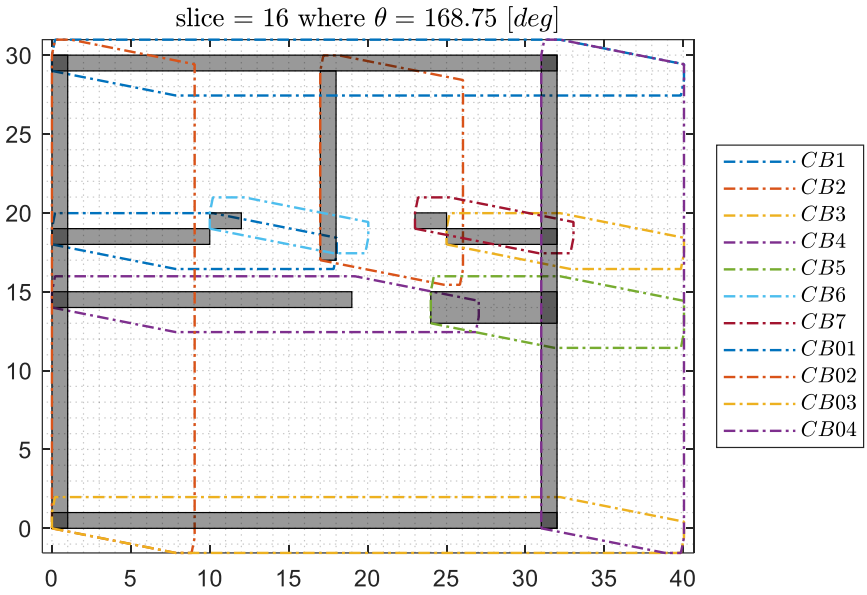


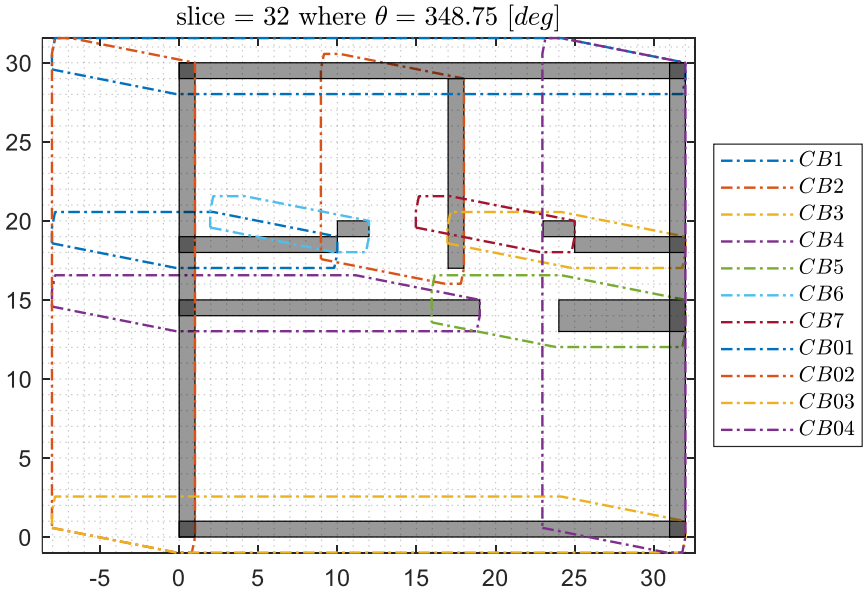
2 Extension to multiple obstacles

We extended the program to show an entire apartment consisting of multiple obstacles. The resulting plots are given below for selected slices. Each obstacle is given its own CB as shown in the legend of each plot.


slice = 1 where θ = 0 [deg]
30
25
20
15
10
5
0
-5
0
5
10
15
20
25
30
CB1
CB2
CB3
CB4
CB5
CB6
CB7
CB01
CB02
CB03
CB04

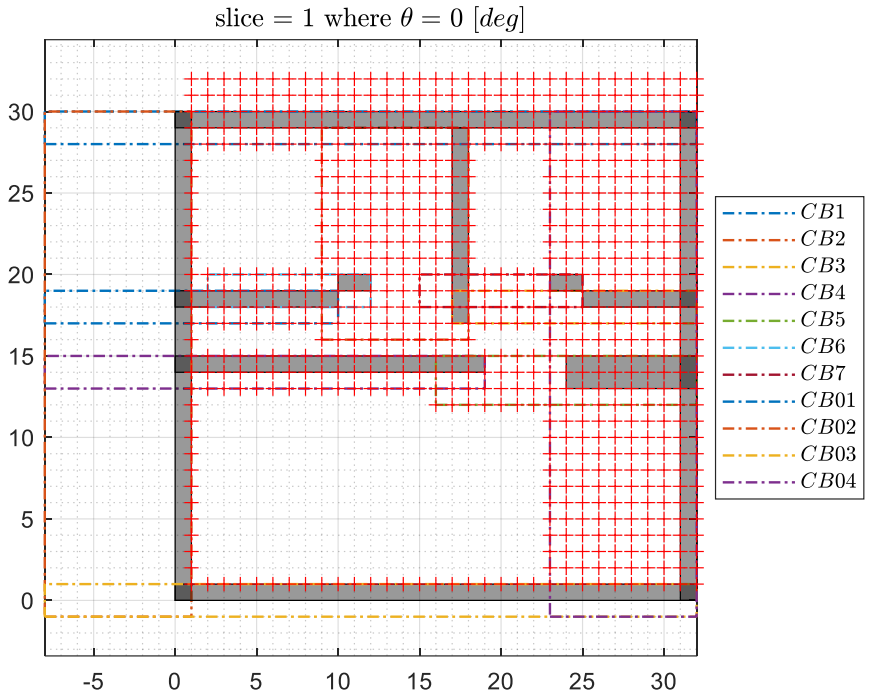

slice = 8 where θ = 78.75 [deg]
30
25
20
15
10
5
0
-5
-5
0
5
10
15
20
25
30
35
CB1
CB2
CB3
CB4
CB5
CB6
CB7
CB01
CB02
CB03
CB04

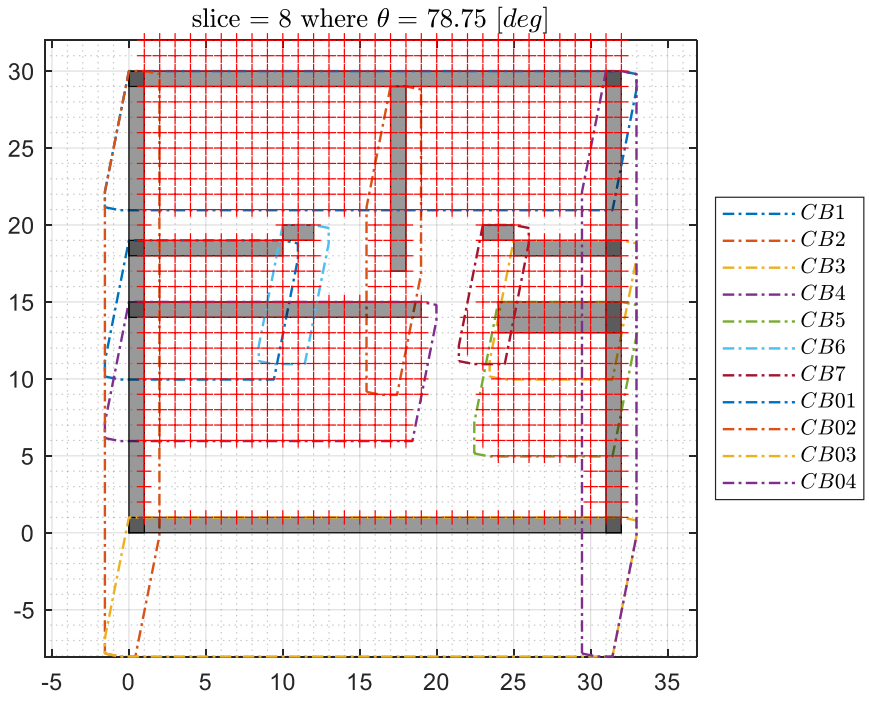

slice = 16 where θ = 168.75 [deg]
CB1
CB2
CB3
CB4
CB5
CB6
CB7
CB01
CB02
CB03
CB04

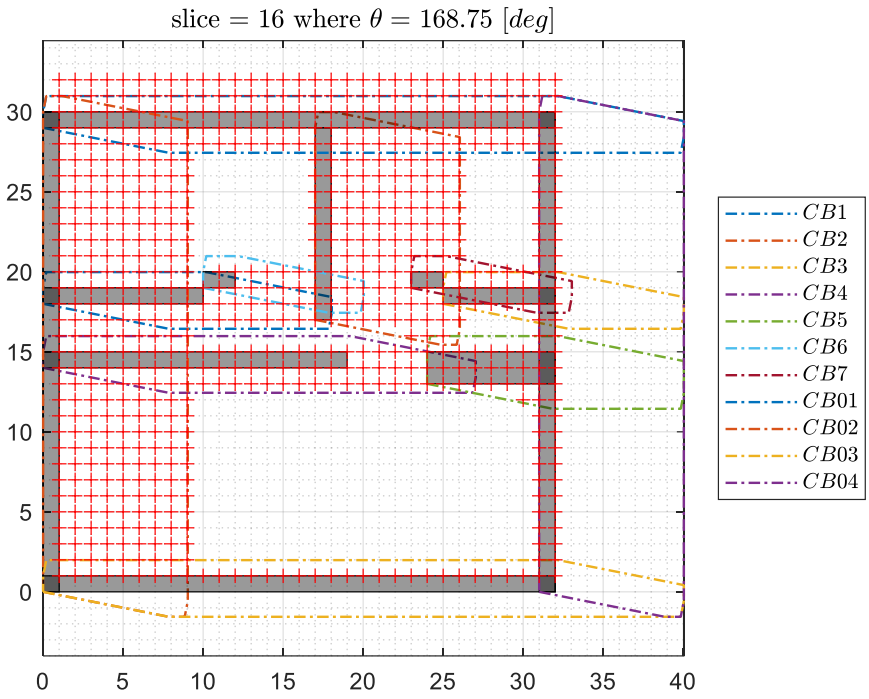

slice = 32 where θ = 348.75 [deg]
CB1
CB2
CB3
CB4
CB5
CB6
CB7
CB01
CB02
CB03
CB04

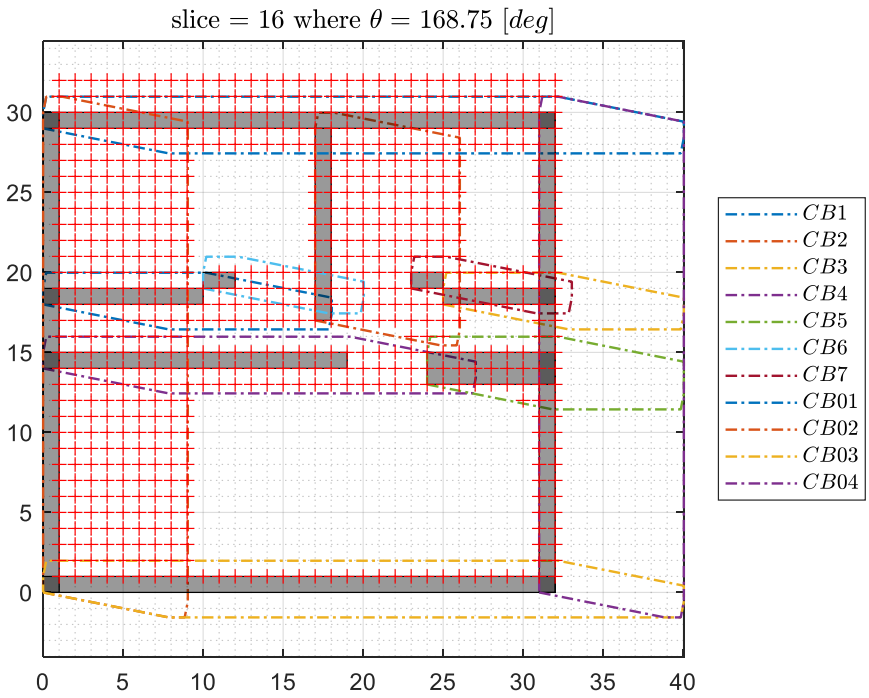
3 Grid preparation

We prepared a grid of (x, y, θ) of dimension $(32x32x32)$, to represent the locations in the given room. For each position in the grid, we checked if the given position was within the C space obstacle and added the prohibited locations to the previous plots of the room. This prepares the ground for path planning in the configuration space.






slice = 16 where θ = 168.75 [deg]
CB1
CB2
CB3
CB4
CB5
CB6
CB7
CB01
CB02
CB03
CB04


slice = 16 where θ = 168.75 [deg]
CB1
CB2
CB3
CB4
CB5
CB6
CB7
CB01
CB02
CB03
CB04

4 Appendix – MATLAB Code

The following code was implemented in MATLAB to solve the question and presented as is.

There is a main file called "Project.m" and functions named "AAPL.m", "createStandardPlot.m", "getCB.m" and "inhull.m". The main file contains answers to all three questions.

1. "AAPL.m" is used to check the applicability condition for point being in CB.
2. "getCB.m" calls AAPL by gathering all points in the CB for a given robot and a boundary.
3. "createStandardPlot.m" is a standard figure creation for all plots throughout the project.
4. "inhull.m" is a function used to check if a given point is within a convex hull, used in Q3.

4.1 Project file (Main program)

```
%% Q1
% ------------------------------------------------------------------------ %
clearvars; clc; close all;
% ------------------------------------------------------------------------ %

% Define polygons via vertices in cc order (original == without rotation)
FrameA  = repmat([4 24], [4, 1]);
FrameB1 = repmat([0 18], [4, 1]);
A_org   = FrameA + [0, 0; 8, 0; 8, 1; 0, 1]; % (x, y) coords, nA = 4 vertices
B1  = FrameB1 + [0, 0; 10, 0; 10, 1; 0, 1]; % nB1 = 4 vertices

% Compute the normal edges of A and B1
vA_org  = [0 -1; 1 0; 0 1; -1 0]; % original normal edges of A -> [vA1; vA2; vA3;
vA4], nA = 4 vertices
vB1 = [0 -1; 1 0; 0 1; -1 0]; % normal edges of B1 -> [vB1_1; vB1_2; vB1_3;
vB1_4],

% apply a rotation of theta only to A because the Boundary is fixed.
thetas = linspace(0, 2*pi - 2*pi/32, 32);
for slice=1:32
    theta=thetas(slice);
    R3d = axang2rotm([0, 0, 1, theta]); % like a bowass
    R = R3d(1:2, 1:2); % go to 2D
    vA = transpose(R*vA_org'); % update/override vA to rotated vA
    A = FrameA + transpose(R*(A_org - FrameA)'); % update/override A to rotated A
    CB = getCB(A, B1, vA, vB1);

    % Plot selected slices
    if (slice==1)||(slice==8)||(slice==16)||(slice==32)
        createStandardPlot(slice, theta); % custom function for standard plotting

        % draw the rotated robot on one of the CB points
        FrameFeature = repmat([CB(1,1), CB(1,2)], [4 1]);
        A_featured = A - FrameA + FrameFeature; % move robot origin to some point
on CB
        Ax_ax = R*[1; 0]; Ay_ax = R*[0; 1];
        fill(A_featured(:,1), A_featured(:,2), 'b', 'FaceAlpha', 0.5,
'DisplayName', 'Robot $A$'); % Polygon A
        fill(B1(:,1), B1(:,2), 'black', 'FaceAlpha', 0.5, 'DisplayName', '$B1$');
% Polygon B
        plot(CB(:,1), CB(:,2), 'r-.', 'DisplayName', '$CB1$', 'LineWidth', 1); %
CB
```

```
        quiver(A_featured(1,1), A_featured(1,2), Ax_ax(1), Ax_ax(2), 'c',
'filled', 'LineWidth', 1.5, ...
            "DisplayName", "$\hat {x}_A$", "MaxHeadSize", 1)
        quiver(A_featured(1,1), A_featured(1,2), Ay_ax(1), Ay_ax(2), 'm',
'filled', 'LineWidth', 1.5, ...
            "DisplayName", "$\hat {y}_A$", "MaxHeadSize", 1)
        maxx = max([CB(:,1); B1(:,1); A_featured(:,1)]); minx = min([CB(:,1);
B1(:,1); A_featured(:,1)]);
        xlim([minx-1, maxx+1]);
        axis equal; grid on; legend("Location","eastoutside")
    end
end


%% Q2
% ------------------------------------------------------------------------ %
clearvars; clc; close all;
% ------------------------------------------------------------------------ %

% redefine obstacles in world frame
B1 = [0 18; 10 18; 10 19; 0 19];
B2 = [17 17; 18 17; 18 29; 17 29];
B3 = [25 18; 32 18; 32 19; 25 19];
B4 = [0 14; 19 14; 19 15; 0 15];
B5 = [24 13; 32 13; 32 15; 24 15];
B6 = [10 19; 12 19; 12 20; 10 20];
B7 = [23 19; 25 19; 25 20; 23 20];
B01 = [0 29; 32 29; 32 30; 0 30];
B02 = [0 0; 1 0; 1 30; 0 30];
B03 = [0 0; 32 0; 32 1; 0 1];
B04 = [31 0; 32 0; 32 30; 31 30];

% redefine robot and his frame
FrameA  = repmat([4 24], [4, 1]);
A_org   = FrameA + [0, 0; 8, 0; 8, 1; 0, 1]; % (x, y) coords, nA = 4 vertices

% Create a cell to include all boundary in single datastructure
B_All = {B1; B2; B3; B4; B5; B6; B7; B01; B02; B03; B04};
v_org = [0 -1; 1 0; 0 1; -1 0]; % all polygons have same noramal edge vectors

% Find all obstacle CB
CB_Combined = num2cell(zeros(length(B_All), 32)); % init by length of B_all x 32
slices

% Loop over all obstacles over all slices
thetas = linspace(0, 2*pi - 2*pi/32, 32);
for obst_inx=1:length(B_All)
    for slice=1:32
        theta=thetas(slice);
        R3d = axang2rotm([0, 0, 1, theta]); % like a bowass
        R = R3d(1:2, 1:2); % go to 2D
        vRobot = transpose(R*v_org'); % update/override v_org to rotated vRobot
        Robot = FrameA + transpose(R*(A_org - FrameA)'); % update/override A to
rotated A
        CB = getCB(Robot, B_All{obst_inx}(:,:), vRobot, v_org);
        CB_Combined{obst_inx, slice} = CB;
    end
end
```

```
% Plot all obstacles for selected slices
% function to convert index of obstacles to their real names
obst_inx_to_cobst =
@(n)(n*(n<=7)+(0.1)*(n==8)+0.2*(n==9)+0.3*(n==10)+0.4*(n==11));
for slice=[1, 8, 16, 32]
    createStandardPlot(slice, thetas(slice)); % custom function for standard
plotting
    for obst_inx=1:length(B_All)
        CB = CB_Combined{obst_inx, slice};
        cobst = replace(string(obst_inx_to_cobst(obst_inx)), ".", ""); % name CB01
instead of CB0.1
        fill(B_All{obst_inx}(:,1), B_All{obst_inx}(:,2), 'black', 'FaceAlpha',
0.4, 'HandleVisibility','off') % 'DisplayName', 'B' + string(obst_inx)); % Polygon
B
        plot(CB(:,1), CB(:,2), 'DisplayName', '$CB$'+ cobst, 'LineWidth', 1,
'LineStyle','-.'); % CB
    end
    axis equal; grid minor; legend("Location","eastoutside")
end

% % add the robot to the plot in some configuration
% CB_Featured = CB_Combined{6, slice};
% FrameFeature = repmat([CB_Featured(1,1), CB_Featured(1,2)], [4 1]);
% Robot_featured = Robot - FrameA + FrameFeature; % move robot origin to some
point on CB
% Robotx_ax = R*[1; 0]; Roboty_ax = R*[0; 1];
% fill(Robot_featured(:,1), Robot_featured(:,2), 'b', 'FaceAlpha', 0.5,
'DisplayName', 'Robot $A$'); % Polygon A
% quiver(Robot_featured(1,1), Robot_featured(1,2), Robotx_ax(1), Robotx_ax(2),
'c', 'filled', 'LineWidth', 1.5, ...
%     "DisplayName", "$\hat {x}_A$", "MaxHeadSize", 1, 'HandleVisibility','off')
% quiver(Robot_featured(1,1), Robot_featured(1,2), Roboty_ax(1), Roboty_ax(2),
'm', 'filled', 'LineWidth', 1.5, ...
%     "DisplayName", "$\hat {y}_A$", "MaxHeadSize", 1, 'HandleVisibility','off')

%% Q3
% ------------------------------------------------------------------------ %
clc; % close all; % to show both Q2 and Q3 on same plots for selected slices
% ------------------------------------------------------------------------ %
space_grid = zeros(32,32,32);

% create a set of points to test for in each CB
test_points = table2array(combinations(linspace(1, 32, 32), linspace(1, 32, 32)));
% ------------------ NOTICE -------------------- %
% the test points start at the (1,1) location and end at the (32,32)
% location. Only these points will be tested, so if a finer resolution is
% needed this is an issu,e because it was assumed that the test_points
% indeces are also the (x, y) locations, so one does not simply change the
% linspace because then it will not be a valid index.
% ---------------------------------------- %
for slice=1:32
    % fill the grid with ones for obstacles and zeros are by default.
    for obst_inx=1:length(B_All)
        % compute for every boundary within given slice all the points in cb
        CB = CB_Combined{obst_inx, slice};
        inx_in_cb = inhull(test_points, CB, convhulln(CB), 0); % find the points
contained in the CB convex hull of a polygon.
        xy_in_cb = test_points(inx_in_cb , :);
        for x=1:32
```

```
            for y=1:32
                if space_grid(x, y, slice)==0
                   space_grid(x, y, slice) = 1 * ismember([x, y], xy_in_cb,
"rows");
                end
            end
        end

    end
end

% Mannual override
space_grid(:, 31:32, :) = 1; % because it is out of the wall for all thetas.
space_grid(:, 1, :) = 1; % wall
space_grid(:, 30, :) = 1; % wall
space_grid(1, :, :) = 1; % wall
space_grid(32, :, :) = 1; % wall

% Plotting
for slice_to_plot = [1 8 16 32]
    % add boundaries to exsiting figures
    createStandardPlot(slice_to_plot, thetas(slice_to_plot))
    [x,y] = find(space_grid(:,:,slice_to_plot) == 1);
    plot(x,y,'r+', 'HandleVisibility','off')
    axis equal; grid on; % legend("off") if in separate plots
end
```

4.2 Function AAPL

```
function apply = AAPL(type, V, Polygon, i, j)
%AAPL returns true or false for appilcability condition.
% type - "A" or "B" type applicability condition
% V - the edge vectors of the "type"
% Poly - the vertices of the polygon of the other than "type"
% i - for A vertices, j - for B vertices

% extend the Polygon and edge Vector to access i,j -+ 1 elements.
ExPoly = [Polygon(end, :); Polygon; Polygon(1, :)];

if type=="A"
    vA_i = V(i,:);
    % take j-1 and j, but make sure there is no index error in j-1=0
    Db_jp = ExPoly(j, :) - Polygon(j, :); % j-1 is like taking 4 1 2 3 by order of
1 2 3 4
    % take j+1 and j, but make sure there is no error in j+1=5
    Db_jm = ExPoly(j+2, :) - Polygon(j, :); % j+1 is like taking 2 3 4 1 by order
of 1 2 3 4
    apply=false; % by default unless found otherwise
    if dot(vA_i, Db_jp)>=0
        if dot(vA_i, Db_jm)>=0
            apply=true;
        end
    end

elseif type=="B"
    vB_j = V(j,:);
    Da_ip = ExPoly(i, :) - Polygon(i, :);
```

```
    Da_jm = ExPoly(i+2, :) - Polygon(i, :);
    apply=false;
    if dot(vB_j, Da_ip)>=0
        if dot(vB_j, Da_jm)>=0
            apply=true;
        end
    end
end
end
```

4.3 Function getCB

```
function [CB] = getCB(Robot, Obstacle, vRobot, vObstacle)
%GETCB find the C-space obstacle using the Lozano Perez algorithm mentioned in
Latombe ch. 3
% The idea is to scan the vectors -vA_i(theta) and vB_j in cc order
% Then construct all (nA + nB) vertices.

    CB = [NaN, NaN]; % just to init... will be removed later
    ExRobot = [Robot(end, :); Robot; Robot(1, :)]; % to access i-1 and i+1 points
extend the matrix
    ExObstacle = [Obstacle(end, :); Obstacle; Obstacle(1, :)]; % to access j-1 and
j+1 points extend the matrix

    for i = 1:4
        for j = 1:4
            % ignore in the case of contact of faces
            if -vRobot(i, :)==vObstacle(j, :)
                continue
            end
            if AAPL("A", vRobot, Obstacle, i, j)
                % so (bj-ai) and (bj-ai+1) are vertices of CB
                CB = union(CB, Obstacle(j, :)-Robot(i, :), 'rows'); % union: add
to CB only points that are not there yet, over rows axis
                CB = union(CB, Obstacle(j, :)-ExRobot(i+2, :), 'rows');
            end
            if AAPL("B", vObstacle, Robot, i, j)
                % so (bj-ai) and (bj+1-ai) are vertices of CB
                CB = union(CB, Obstacle(j, :) - Robot(i, :), 'rows');
                CB = union(CB, ExObstacle(j+2, :) - Robot(i, :), 'rows');
            end
        end
    end
    CB = CB(~any(isnan(CB), 2), :); % clean the NaN values used to init.
    k = convhull(CB(:, 1), CB(:, 2)); % use convex hull to reorder points
    CB = CB(k, :) + repmat(Robot(1,:), [length(k),1]); % get the CB to origin with
the boundary
end
```

4.4 Function createStandardPlot

```
function createStandardPlot(slice, theta)
    figure(slice); hold on; box on; axis equal; legend('show');
    set(groot, 'defaultTextInterpreter', 'latex');
    set(groot, 'defaultLegendInterpreter', 'latex');
```

```
    set(gcf, "Color", "w")
    title("slice = " + string(slice) +" where " + "$\theta =$ " +
string(rad2deg(theta)) + " $[deg]$")
end
```

4.5 Function inhull

```
function in = inhull(testpts,xyz,tess,tol)
% inhull: tests if a set of points are inside a convex hull
% usage: in = inhull(testpts,xyz)
% usage: in = inhull(testpts,xyz,tess)
% usage: in = inhull(testpts,xyz,tess,tol)
%
% arguments: (input)
%  testpts - nxp array to test, n data points, in p dimensions
%       If you have many points to test, it is most efficient to
%       call this function once with the entire set.
%
%  xyz - mxp array of vertices of the convex hull, as used by
%       convhulln.
%
%  tess - tessellation (or triangulation) generated by convhulln
%       If tess is left empty or not supplied, then it will be
%       generated.
%
%  tol - (OPTIONAL) tolerance on the tests for inclusion in the
%       convex hull. You can think of tol as the distance a point
%       may possibly lie outside the hull, and still be perceived
%       as on the surface of the hull. Because of numerical slop
%       nothing can ever be done exactly here. I might guess a
%       semi-intelligent value of tol to be
%
%         tol = 1.e-13*mean(abs(xyz(:)))
%
%       In higher dimensions, the numerical issues of floating
%       point arithmetic will probably suggest a larger value
%       of tol.
%
%       DEFAULT: tol = 0
%
% arguments: (output)
%  in  - nx1 logical vector
%        in(i) == 1 --> the i'th point was inside the convex hull.
%
% Example usage: The first point should be inside, the second out
%
%  xy = randn(20,2);
%  tess = convhulln(xy);
%  testpoints = [ 0 0; 10 10];
%  in = inhull(testpoints,xy,tess)
%
% in =
%      1
%      0
%
% A non-zero count of the number of degenerate simplexes in the hull
% will generate a warning (in 4 or more dimensions.) This warning
```

```
% may be disabled off with the command:
%
%   warning('off','inhull:degeneracy')
%
% See also: convhull, convhulln, delaunay, delaunayn, tsearch, tsearchn
%
% Author: John D'Errico
% e-mail: woodchips@rochester.rr.com
% Release: 3.0
% Release date: 10/26/06
% get array sizes
% m points, p dimensions
p = size(xyz,2);
[n,c] = size(testpts);
if p ~= c
  error 'testpts and xyz must have the same number of columns'
end
if p < 2
  error 'Points must lie in at least a 2-d space.'
end
% was the convex hull supplied?
if (nargin<3) || isempty(tess)
  tess = convhulln(xyz);
end
[nt,c] = size(tess);
if c ~= p
  error 'tess array is incompatible with a dimension p space'
end
% was tol supplied?
if (nargin<4) || isempty(tol)
  tol = 0;
end
% build normal vectors
switch p
  case 2
    % really simple for 2-d
    nrmls = (xyz(tess(:,1),:) - xyz(tess(:,2),:)) * [0 1;-1 0];

    % Any degenerate edges?
    del = sqrt(sum(nrmls.^2,2));
    degenflag = (del<(max(del)*10*eps));
    if sum(degenflag)>0
      warning('inhull:degeneracy',[num2str(sum(degenflag)), ...
        ' degenerate edges identified in the convex hull'])

      % we need to delete those degenerate normal vectors
      nrmls(degenflag,:) = [];
      nt = size(nrmls,1);
    end
  case 3
    % use vectorized cross product for 3-d
    ab = xyz(tess(:,1),:) - xyz(tess(:,2),:);
    ac = xyz(tess(:,1),:) - xyz(tess(:,3),:);
    nrmls = cross(ab,ac,2);
    degenflag = false(nt,1);
  otherwise
    % slightly more work in higher dimensions,
    nrmls = zeros(nt,p);
    degenflag = false(nt,1);
```

```
    for i = 1:nt
      % just in case of a degeneracy
      % Note that bsxfun COULD be used in this line, but I have chosen to
      % not do so to maintain compatibility. This code is still used by
      % users of older releases.
      %  nullsp = null(bsxfun(@minus,xyz(tess(i,2:end),:),xyz(tess(i,1),:)))';
      nullsp = null(xyz(tess(i,2:end),:) - repmat(xyz(tess(i,1),:),p-1,1))';
      if size(nullsp,1)>1
        degenflag(i) = true;
        nrmls(i,:) = NaN;
      else
        nrmls(i,:) = nullsp;
      end
    end
    if sum(degenflag)>0
      warning('inhull:degeneracy',[num2str(sum(degenflag)), ...
        ' degenerate simplexes identified in the convex hull'])

      % we need to delete those degenerate normal vectors
      nrmls(degenflag,:) = [];
      nt = size(nrmls,1);
    end
end
% scale normal vectors to unit length
nrmllen = sqrt(sum(nrmls.^2,2));
% again, bsxfun COULD be employed here...
%  nrmls = bsxfun(@times,nrmls,1./nrmllen);
nrmls = nrmls.*repmat(1./nrmllen,1,p);
% center point in the hull
center = mean(xyz,1);
% any point in the plane of each simplex in the convex hull
a = xyz(tess(~degenflag,1),:);
% ensure the normals are pointing inwards
% this line too could employ bsxfun...
%  dp = sum(bsxfun(@minus,center,a).*nrmls,2);
dp = sum((repmat(center,nt,1) - a).*nrmls,2);
k = dp<0;
nrmls(k,:) = -nrmls(k,:);
% We want to test if:  dot((x - a),N) >= 0
% If so for all faces of the hull, then x is inside
% the hull. Change this to dot(x,N) >= dot(a,N)
aN = sum(nrmls.*a,2);
% test, be careful in case there are many points
in = false(n,1);
% if n is too large, we need to worry about the
% dot product grabbing huge chunks of memory.
memblock = 1e6;
blocks = max(1,floor(n/(memblock/nt)));
aNr = repmat(aN,1,length(1:blocks:n));
for i = 1:blocks
   j = i:blocks:n;
   if size(aNr,2) ~= length(j)
      aNr = repmat(aN,1,length(j));
   end
   in(j) = all((nrmls*testpts(j,:)' - aNr) >= -tol,1)';
end
```