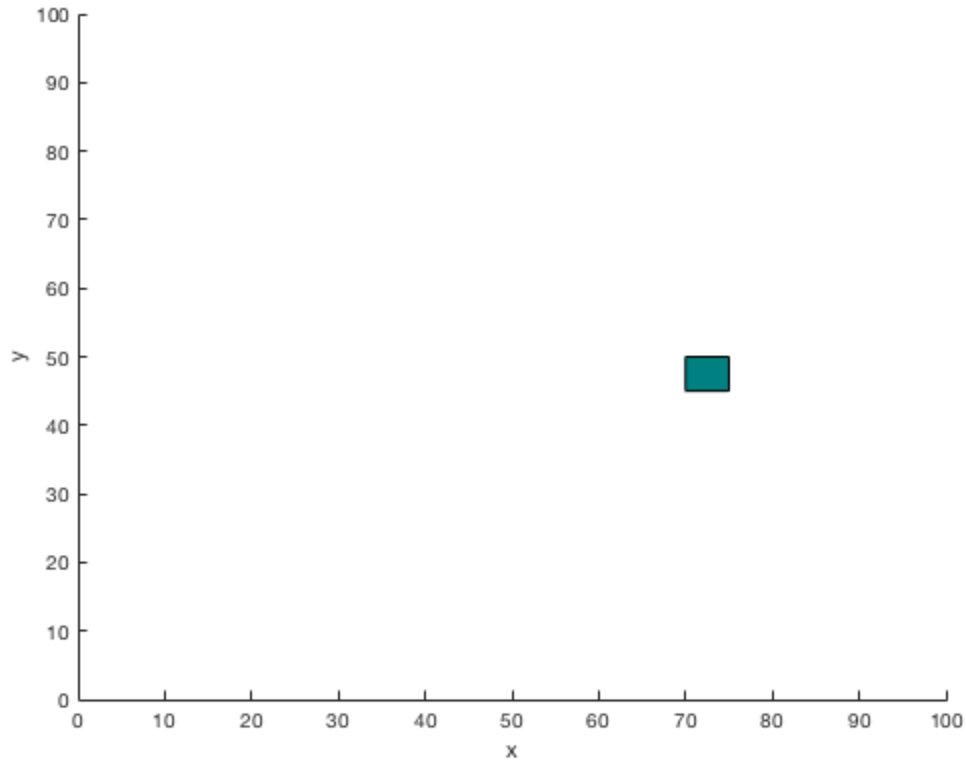
Table of Contents

q1

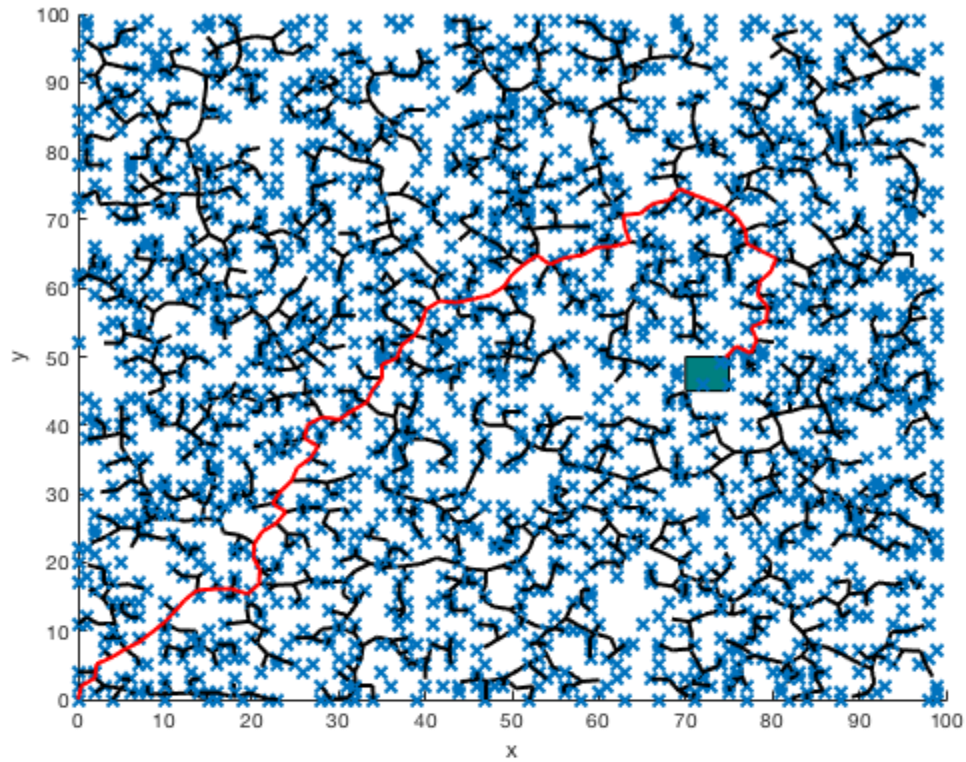
```
clearvars
close all

% x and y axises limit from 0 to x_max and 0 to y_max respectively.
x_max = 100; %;
y_max = 100; %;

% distance of moving at every step
EPS = 2;

% maximum iterations
numNodes = 5000; %;

% attributions of starting point
q_start.coord = [0 0];
q_start.cost = 0;
q_start.parent = 0; %.parent means the index of parent node

% initialize the tree
nodes(1) = q_start;

% plot the goal area
figure(1)
axis([0 x_max 0 y_max])
goal_area = rectangle('Position',[70,45,5,5],'FaceColor',[0 .5 .5]);
xlabel('x')
ylabel('y')
hold on
```



grow the tree

```
for i = 1:1:numNodes

    % generate the random points in the given safe area and plot the
 points
    q_rand = [floor(rand(1)*x_max) floor(rand(1)*y_max)];
    plot(q_rand(1), q_rand(2), 'x', 'Color',  [0 0.4470 0.7410])

    % Find the nearest point existing on the tree to the random point
    ndist = [];
    for j = 1:1:length(nodes)
        n = nodes(j);
        tmp = norm(n.coord - q_rand);
        ndist = [ndist tmp];
    end
    [mini_distance, idx] = min(ndist);
    q_nearest = nodes(idx);

    % move to the random point with distance of eps if distance
 between
    % random point and nearest point is bigger than eps.
    q_new.coord = steer(q_rand, q_nearest.coord, mini_distance, EPS);
    line([q_nearest.coord(1), q_new.coord(1)], [q_nearest.coord(2),
 q_new.coord(2)],...
```

```
        'Color', 'k', 'LineWidth', 2);
    drawnow
    hold on
    q_new.cost = norm(q_new.coord - q_nearest.coord) + q_nearest.cost;
    q_new.parent = idx;

    % Append to nodes
    nodes = [nodes q_new];

    % Break if the link from second to last node to last node
 intersects any of
    % the four edges of the goal area
    if ~noCollision(q_nearest.coord, q_new.coord, [70,45,5,5])
        break
    end
end
q_end = q_new;
num_node_path = 1;
while q_end.parent ~= 0
    start = q_end.parent;
    line([q_end.coord(1), nodes(start).coord(1)], [q_end.coord(2),
 nodes(start).coord(2)],...
        'Color', 'r', 'LineWidth', 2);
    hold on
    q_end = nodes(start);
    num_node_path = num_node_path+1;
end
```



total number of node in the tree

```
num_node_tree = length(nodes)
```

```
num_node_tree =

        2272
```

number of nodes in the sequence that reaches goal area

```
num_node_path
```

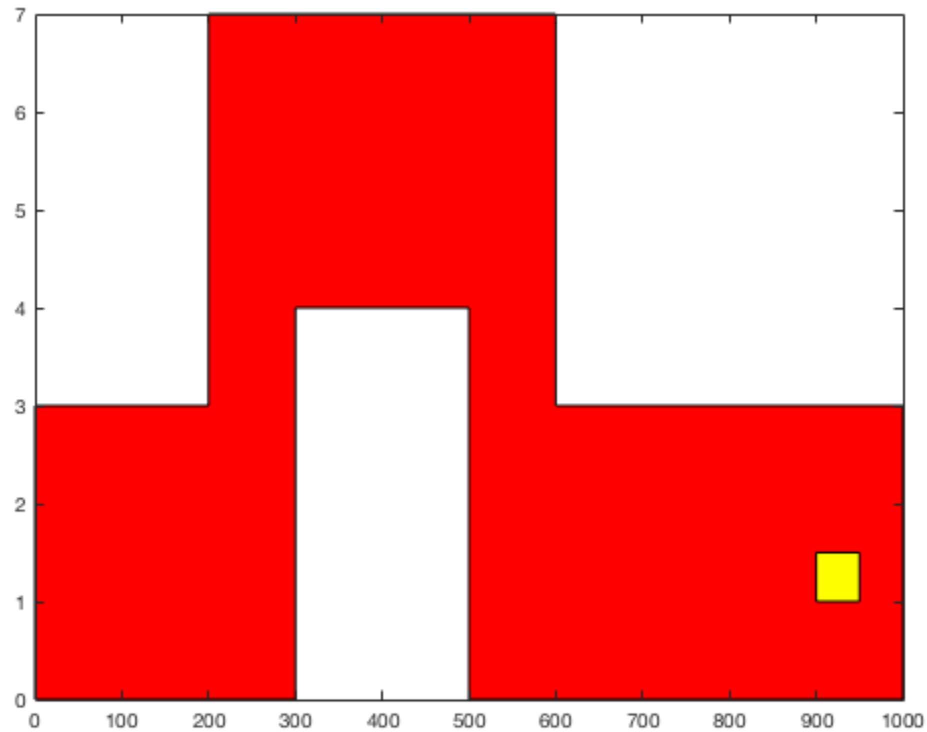
```
num_node_path =

    80
```

Published with MATLAB® R2017b

Table of Contents

Q2

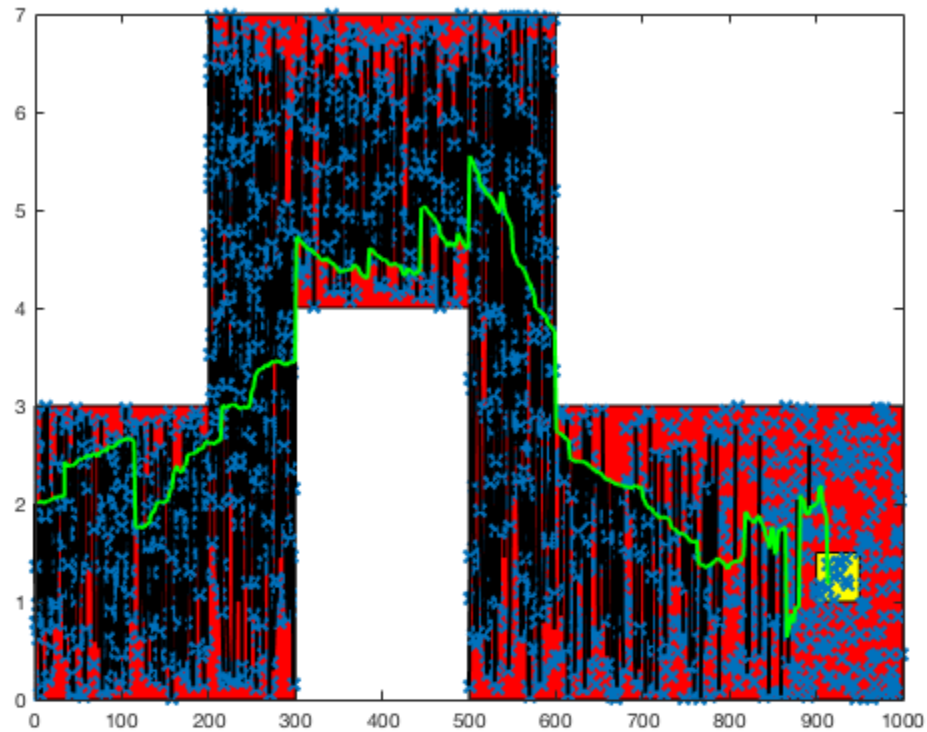
```
clearvars
close all

% x and y axises limit from 0 to x_max and 0 to y_max respectively.
x_max = 1000;
y_max = 7;

% distance of moving at every step
EPS = 2;

% maximum iterations
numNodes = 6000;

% attributions of starting point
q_start.coord = [0 2];
q_start.cost = 0;
q_start.parent = 0; %.parent means the index of parent node

% initialize the tree
nodes(1) = q_start;

% plot the safe area
figure(1)
x=[0 300 300 500 500 1000 1000 600 600 200 200 0]; %x coordinates of
 all the vertices
y=[0 0 4 4 0 0 3 3 7 7 3 3];  %y coordinates of all the vertices
X=[x,x(1)];   %?????????????????????
Y=[y,y(1)];   %??
plot(X,Y,'k')  %?????
fill(x,y,'r')  % fill the safe zone with color
hold on

% plot the goal area
X_goal = [900,900,950,950,900];
Y_goal = [1,1.5,1.5,1,1];
plot(X_goal,Y_goal,'y')
fill(X_goal,Y_goal,'y')
```



grow the tree

```
for i = 1:1:numNodes
    pan = 0;
    % generate the random points in the given safe area and plot the
 points
    while ~pan
        q_rand = [rand*x_max,rand*y_max];
        pan = inpolygon(q_rand(1),q_rand(2),X,Y);
    end
    plot(q_rand(1), q_rand(2), 'x', 'Color',  [0 0.4470 0.7410])
    ndist = [];
    for j = 1:1:length(nodes)
        n = nodes(j);
        tmp = norm(n.coord - q_rand);
        ndist = [ndist tmp];
    end
    [mini_distance, idx] = min(ndist);
    q_nearest = nodes(idx);
    q_new.coord = steer(q_rand, q_nearest.coord, mini_distance, EPS);

    InorOn = inpolygon(q_new.coord(1),q_new.coord(2),X,Y);
    if InorOn
        line([q_nearest.coord(1), q_new.coord(1)],
 [q_nearest.coord(2), q_new.coord(2)],...
```

```
            'Color', 'k', 'LineWidth', 2);
        drawnow
        hold on
        q_new.cost = norm(q_new.coord - q_nearest.coord) +
 q_nearest.cost;
        q_new.parent = idx;

        % Break if the link from second to last node to last node
 intersects any of
        % the four edges of the goal area
        nodes = [nodes q_new];
    end

    % move to the random point with distance of eps if distance
 between
    % random point and nearest point is bigger than eps.

    if ~noCollision(q_nearest.coord, q_new.coord, [900,1,50,0.5])
        break
    end
end

q_end = q_new;
num_node_path = 1;
while q_end.parent ~= 0
    start = q_end.parent;
    line([q_end.coord(1), nodes(start).coord(1)], [q_end.coord(2),
 nodes(start).coord(2)],...
        'Color', 'g', 'LineWidth', 2);
    hold on
    q_end = nodes(start);
    num_node_path = num_node_path+1;
end
```



total number of node in the tree

```
num_node_tree = length(nodes)
```

```
num_node_tree =

        1652
```

number of nodes in the sequence that reaches goal area

```
num_node_path
```

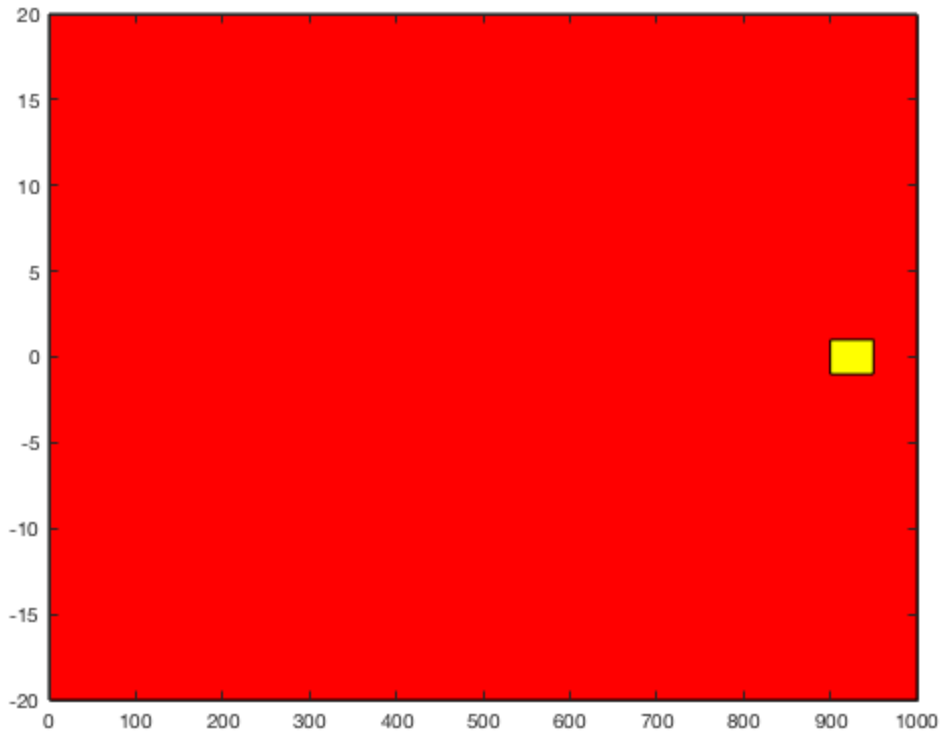
```
num_node_path =

   460
```

Published with MATLAB® R2017b

Table of Contents

Q3

```
clearvars
close all

% x and y axises limit from 0 to x_max and 0 to y_max respectively.
x_min = 0; x_max = 1000;
y_min = -20; y_max = 20;
psi_min = -pi; psi_max = pi;

% distance of moving at every step
EPS = 2;

% maximum iterations
numNodes = 6000;

% attributions of starting point
q_start.coord = [0 0 0];
q_start.cost = 0;
q_start.parent = 0; %.parent means the index of parent node

% initialize the tree
nodes(1) = q_start;

% plot the safe area
figure(1)
x=[0 1000 1000 0]; %x coordinates of all the vertices
y=[-20 -20 20 20];  %y coordinates of all the vertices
X=[x,x(1)];
Y=[y,y(1)];
plot(X,Y,'k')
fill(x,y,'r')  % fill the safe zone with color
hold on

% plot the goal area
X_goal = [900,900,950,950,900];
Y_goal = [-1,1,1,-1,-1];
plot(X_goal,Y_goal,'y')
fill(X_goal,Y_goal,'y')
```

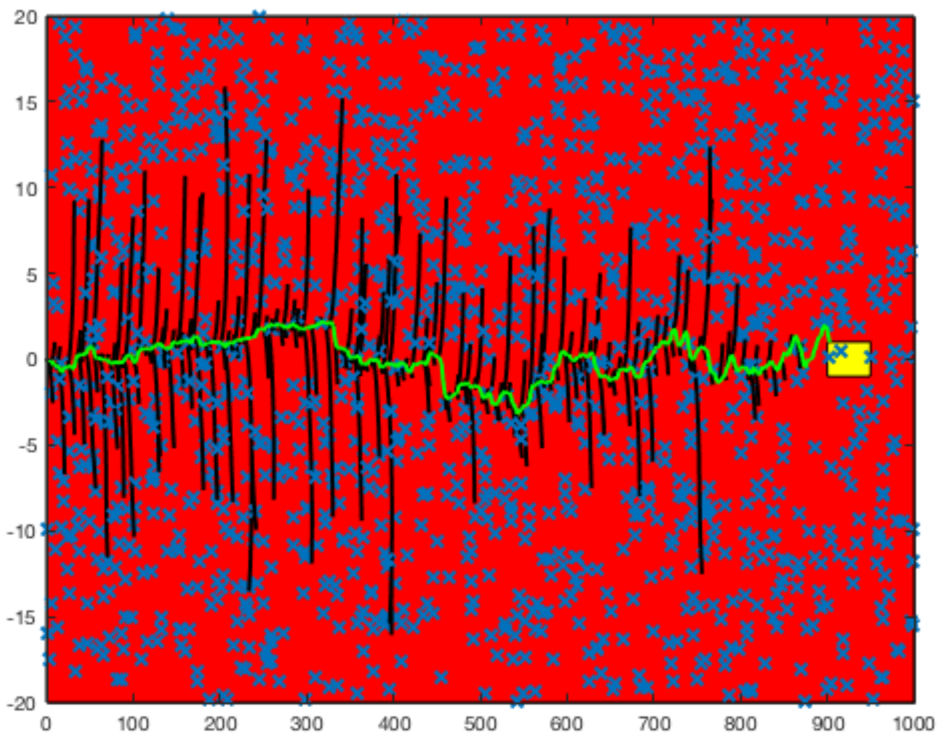


grow the tree

```
for i = 1:1:numNodes
    pan = 0;
    % generate the random points in the given safe area and plot the
 points
    while ~pan
        q_rand = [rand*(x_max-x_min)+x_min,rand*(y_max-
y_min)+y_min,rand*(psi_max-psi_min)+psi_min];
        pan = inpolygon(q_rand(1),q_rand(2),X,Y);
    end
    plot(q_rand(1), q_rand(2), 'x', 'Color',  [0 0.4470 0.7410])
    ndist = [];
    for j = 1:1:length(nodes)
        n = nodes(j);
        tmp = norm(n.coord(1:2) - q_rand(1:2));
        ndist = [ndist tmp];
    end
    [mini_distance, idx] = min(ndist);
    q_nearest = nodes(idx);

    % Get the new Point
    % q_new.coord = steer(q_rand, q_nearest.coord, mini_distance,
 EPS);
```

```
    % Instead of using steer function directly, we apply the brute
 force
    % approach.

    % Parameters
    vx = 30; L = 3;
    dxdt = @(t,x,delta) [vx*cos(x(3));vx*sin(x(3));vx/L*tan(delta)];
    direc_desired = (q_rand(1:2) - q_nearest.coord(1:2))./
norm(q_rand(1:2) - q_nearest.coord(1:2));
    direc = 0;
    flag_fea = false;
    for delta = -20/180*pi:2/180*pi:20/180*pi
        safety = true;
        [t,x] = ode45(@(t,x) dxdt(t,x,delta),[0
 0.1],q_nearest.coord');
        q_fea = x(end,:);
        direc_fea = (q_fea(1:2) - q_nearest.coord(1:2))./
norm(q_fea(1:2) - q_nearest.coord(1:2));
        % to check if any step there is a collison
        stepNum = size(x,1);
        for i = 1:stepNum
            if ~inpolygon(x(i,1),x(i,2),X,Y)
                safety = false;
                break;
            end
        end

        if sum(direc_fea.*direc_desired) > direc && safety
            flag_fea = true;
            direc = sum(direc_fea.*direc_desired);
            q_new.coord = (q_fea - q_nearest.coord)./norm(q_fea -
 q_nearest.coord) * EPS + q_nearest.coord;
        end
    end
    InorOn = inpolygon(q_new.coord(1),q_new.coord(2),X,Y);
    if flag_fea && InorOn
        line([q_nearest.coord(1), q_new.coord(1)],
 [q_nearest.coord(2), q_new.coord(2)],...
            'Color', 'k', 'LineWidth', 2);
        drawnow
        hold on
        q_new.cost = norm(q_new.coord - q_nearest.coord) +
 q_nearest.cost;
        q_new.parent = idx;

        % Break if the link from second to last node to last node
 intersects any of
        % the four edges of the goal area
        nodes = [nodes q_new];
    end

    % move to the random point with distance of eps if distance
 between
    % random point and nearest point is bigger than eps.
```

```
    if ~noCollision(q_nearest.coord(1:2), q_new.coord(1:2),
 [900,-1,50,2])
        break
    end
end

q_end = q_new;
num_node_path = 1;
while q_end.parent ~= 0
    start = q_end.parent;
    line([q_end.coord(1), nodes(start).coord(1)], [q_end.coord(2),
 nodes(start).coord(2)],...
        'Color', 'g', 'LineWidth', 2);
    hold on
    q_end = nodes(start);
    num_node_path = num_node_path+1;
end
```



total number of node in the tree

```
num_node_tree = length(nodes)
```

num_node_tree =

```
        1224
```

number of nodes in the sequence that reaches goal area

```
num_node_path
```

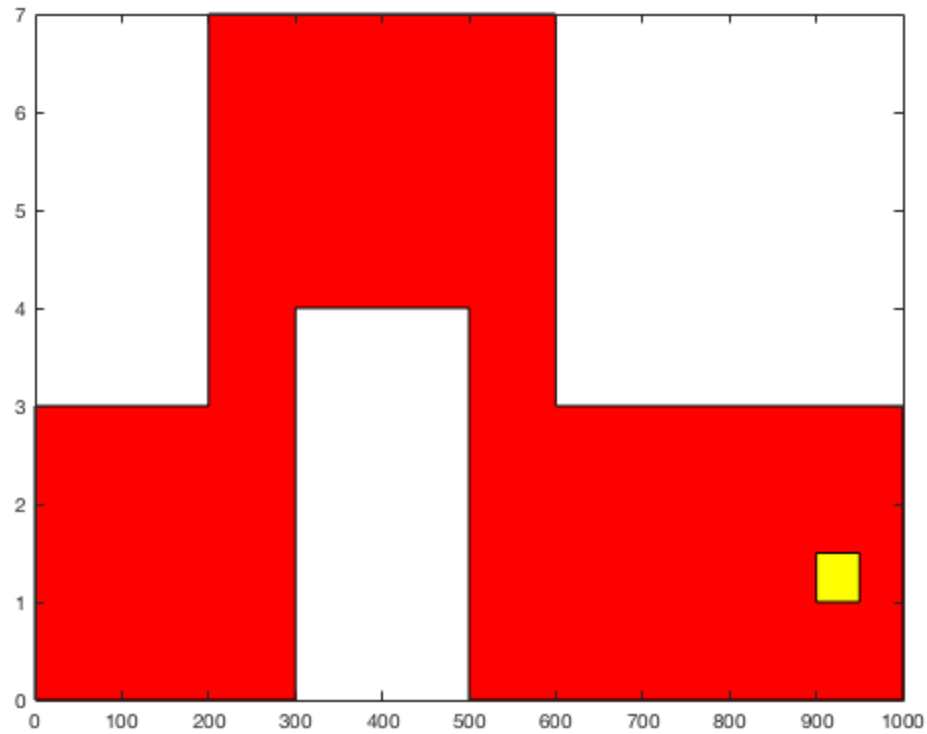
```
num_node_path =

   456
```

Published with MATLAB® R2017b

Table of Contents

Q4

```
clearvars
close all

% x and y axises limit from 0 to x_max and 0 to y_max respectively.
x_min = 0; x_max = 1000;
y_min = -20; y_max = 20;
psi_min = -pi; psi_max = pi;

% distance of moving at every step
EPS = 2;

% maximum iterations
numNodes = 6000;

% attributions of starting point
q_start.coord = [0 0 0];
q_start.cost = 0;
q_start.parent = 0; %.parent means the index of parent node

% initialize the tree
nodes(1) = q_start;

% plot the safe area
figure(1)
x=[0 300 300 500 500 1000 1000 600 600 200 200 0]; %x coordinates of
 all the vertices
y=[0 0 4 4 0 0 3 3 7 7 3 3];  %y coordinates of all the vertices
X=[x,x(1)];
Y=[y,y(1)];
plot(X,Y,'k')
fill(x,y,'r')  % fill the safe zone with color
hold on

% plot the goal area
X_goal = [900,900,950,950,900];
Y_goal = [1,1.5,1.5,1,1];
plot(X_goal,Y_goal,'y')
fill(X_goal,Y_goal,'y')
```

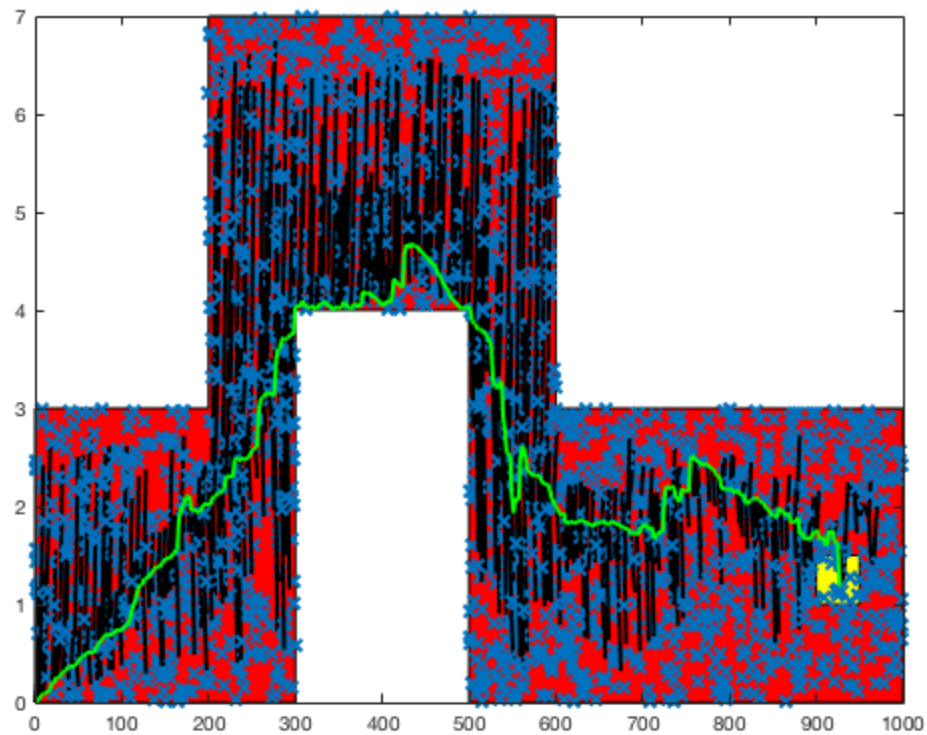


grow the tree

```
tic;
for i = 1:1:numNodes
    pan = 0;
    % generate the random points in the given safe area and plot the
 points
    while ~pan
        q_rand = [rand*(x_max-x_min)+x_min,rand*(y_max-
y_min)+y_min,rand*(psi_max-psi_min)+psi_min];
        pan = inpolygon(q_rand(1),q_rand(2),X,Y);
    end
    plot(q_rand(1), q_rand(2), 'x', 'Color',  [0 0.4470 0.7410])
    ndist = [];
    for j = 1:1:length(nodes)
        n = nodes(j);
        tmp = norm(n.coord(1:2) - q_rand(1:2));
        ndist = [ndist tmp];
    end
    [mini_distance, idx] = min(ndist);
    q_nearest = nodes(idx);

    % Get the new Point
    % q_new.coord = steer(q_rand, q_nearest.coord, mini_distance,
 EPS);
```

```
    % Instead of using steer function directly, we apply the brute
 force
    % approach.

    % Parameters
    vx = 30; L = 3;
    dxdt = @(t,x,delta) [vx*cos(x(3));vx*sin(x(3));vx/L*tan(delta)];
    direc_desired = (q_rand(1:2) - q_nearest.coord(1:2))./
norm(q_rand(1:2) - q_nearest.coord(1:2));
    direc = 0;
    flag_fea = false;
    for delta = -20/180*pi:2/180*pi:20/180*pi
        safety = true;
        [t,x] = ode45(@(t,x) dxdt(t,x,delta),[0
 0.1],q_nearest.coord');
        q_fea = x(end,:);
        direc_fea = (q_fea(1:2) - q_nearest.coord(1:2))./
norm(q_fea(1:2) - q_nearest.coord(1:2));
        % to check if any step there is a collison
        stepNum = size(x,1);
        for i = 1:stepNum
            if ~inpolygon(x(i,1),x(i,2),X,Y)
                safety = false;
                break;
            end
        end

        if sum(direc_fea.*direc_desired) > direc && safety
            flag_fea = true;
            direc = sum(direc_fea.*direc_desired);
            q_new.coord = (q_fea - q_nearest.coord)./norm(q_fea -
 q_nearest.coord) * EPS + q_nearest.coord;
        end
    end
    InorOn = inpolygon(q_new.coord(1),q_new.coord(2),X,Y);
    if flag_fea && InorOn
        line([q_nearest.coord(1), q_new.coord(1)],
 [q_nearest.coord(2), q_new.coord(2)],...
            'Color', 'k', 'LineWidth', 2);
        drawnow
        hold on
        q_new.cost = norm(q_new.coord - q_nearest.coord) +
 q_nearest.cost;
        q_new.parent = idx;

        % Break if the link from second to last node to last node
 intersects any of
        % the four edges of the goal area
        nodes = [nodes q_new];
    end

    % move to the random point with distance of eps if distance
 between
    % random point and nearest point is bigger than eps.
```

```
    if ~noCollision(q_nearest.coord(1:2), q_new.coord(1:2),
 [900,-1,50,2])
        break
    end
end

q_end = q_new;
num_node_path = 1;
while q_end.parent ~= 0
    start = q_end.parent;
    line([q_end.coord(1), nodes(start).coord(1)], [q_end.coord(2),
 nodes(start).coord(2)],...
        'Color', 'g', 'LineWidth', 2);
    hold on
    q_end = nodes(start);
    num_node_path = num_node_path+1;
end
```



total number of node in the tree

```
num_node_tree = length(nodes)
```

num_node_tree =

```
       1463
```

number of nodes in the sequence that reaches goal area

```
num_node_path
```

```
num_node_path =

   465
```

Total Calculation Time

```
toc
```

```
Elapsed time is 262.101255 seconds.
```

Published with MATLAB® R2017b

Comments

Table of Contents

Q5

We found that RRT is not very sensitive to map configuration, moreover when the degree of freedom increases, the complexity increases far more quickly than other methods. In order to imrove the performance, we could use graph search algorithme in the lower dimensional basics (to get path for X and Y) and then use RRT to expand the whole configuration.

Q6

We can use PID or MPC to improve the performance in RRT.

Published with MATLAB® R2017b

Contents

q7

```
clc
clear
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Francesco Borrelli ME C231A 2015
% Kinematic Navigation
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
tic
N=50;
sampling=10;
%Var Defintions
z = sdpvar(2,N);

%Initial and terminal condition
z0 = [0;1];
zT = [850;1];
dzmin=-[20;2];
dzmax=[20;2];
zmin = [0;0];
zmax = [1000;7];

%Obstacle list
i=1;
obs{i}.center=[400;1];
obs{i}.LW=[200;2];
obs{i}.theta=0; %(in radiants)
i=i+1;
obs{i}.center=[800;5];
obs{i}.LW=[400;4];
obs{i}.theta=0; %(in radiants)


% some obtacle postprocessing
for j=1:length(obs)
    t=obs{j}.theta;
    % generate T matrix for each obstacle
    obs{j}.T=[cos(t), -sin(t);sin(t) cos(t)]*diag(obs{j}.LW/2);
    % polyehdral representaion
    obs{j}.poly=obs{j}.T*unitbox(2)+obs{j}.center;
end


%try to remove/add this one


%Constraints
%Setup Optimization Problem
```

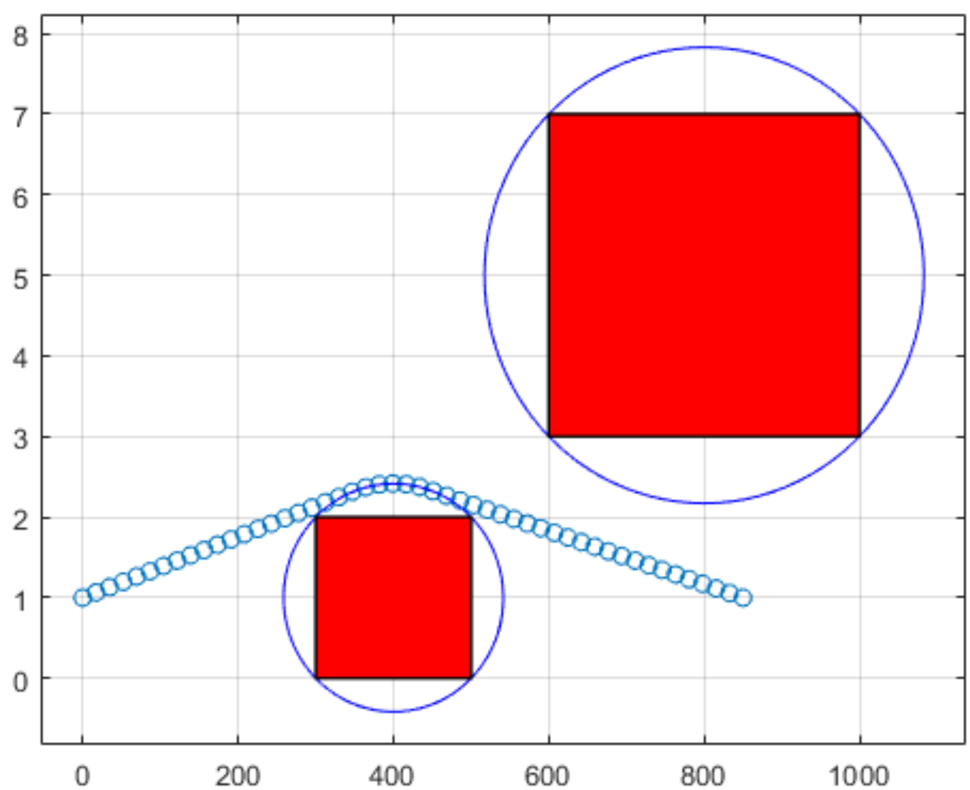
```
cost = 0;
Q=eye(2);
constr = [z(:,1)==z0,z(:,N)==zT];
for t = 2:N
      cost=cost+(z(:,t)-z(:,t-1))'*Q*(z(:,t)-z(:,t-1));
      constr = constr +[dzmin<= z(:,t)-z(:,t-1)<=dzmax];
      constr = constr + [zmin<=z(:,t)<= zmax];
      for k = 0:sampling-1
          for j=1:length(obs)
              xs=z(:,t-1)+k/sampling*(z(:,t)-z(:,t-1));
              constr = constr +[(xs-obs{j}.center)'*inv(obs{j}.T)'*inv(obs{j}.T)*(xs-obs{j
}.center)>=2];
          end
      end
end
options = sdpsettings('solver','ipopt');
%options.ipopt=ipoptset('linear_solver','MUMPS');
solvesdp(constr,cost,options);
z_vec = double(z);


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Plotting Functions % to add title and labels
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
th = 0:pi/50:2*pi;
for j=1:length(obs)
    for l=1:length(th)
        z=[cos(th(l));sin(th(l))]*sqrt(2);
        y=obs{j}.T*z+obs{j}.center;
        xobs{j}(l) = y(1);
        yobs{j}(l) = y(2);
    end
end
```

```
Total number of variables............................:       96
                     variables with only lower bounds:        0
                variables with lower and upper bounds:       96
                     variables with only upper bounds:        0
Total number of equality constraints.................:        0
Total number of inequality constraints...............:     1176
        inequality constraints with only lower bounds:        0
   inequality constraints with lower and upper bounds:        0
        inequality constraints with only upper bounds:     1176


Number of Iterations....: 244

                                   (scaled)                 (unscaled)
Objective...............:   1.4745068842705978e+03    1.4745068842705978e+04
Dual infeasibility......:   9.8279349396820243e-08    9.8279349396820243e-07
Constraint violation....:   0.0000000000000000e+00    0.0000000000000000e+00
Complementarity.........:   1.0000000000000004e-11    1.0000000000000004e-10
Overall NLP error.......:   9.8279349396820243e-08    9.8279349396820243e-07
```

```
Number of objective function evaluations             = 434
Number of objective gradient evaluations             = 245
Number of equality constraint evaluations            = 0
Number of inequality constraint evaluations          = 434
Number of equality constraint Jacobian evaluations   = 0
Number of inequality constraint Jacobian evaluations = 245
Number of Lagrangian Hessian evaluations             = 0
Total CPU secs in IPOPT (w/o function evaluations)   =      0.745
Total CPU secs in NLP function evaluations           =      0.301

EXIT: Optimal Solution Found.
```

plot routine

```
figure
axis([zmin(1) zmax(1) zmin(2) zmax(2)])
plot(z_vec(1,:),z_vec(2,:),'o')
hold on
for j=1:length(obs)
plot(xobs{j}, yobs{j},'b');
plot(obs{j}.T*unitbox(2)+obs{j}.center);
end
```



Comment

```
toc
% the computation time is faster than RRT in most cases.
```

```
Elapsed time is 0.501128 seconds.
```

Published with MATLAB® R2018a

Contents

q8

```
clc
clear
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Francesco Borrelli ME C231A 2015
% Kinematic Navigation
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
tic
N=50;
sampling=10;
%Var Defintions
z = sdpvar(2,N);

%Initial and terminal condition
z0 = [0;1];
zT = [850;1];
dzmin=-[20;2];
dzmax=[20;2];
zmin = [0;0];
zmax = [1000;7];

%Obstacle list
%Obstacle list
i=1;
obs{i}.center=[400;1];
obs{i}.LW=[200;2];
obs{i}.theta=0; %(in radiants)
i=i+1;
obs{i}.center=[800;5];
obs{i}.LW=[400;4];
obs{i}.theta=0; %(in radiants)


% integer variables
d = binvar(4*length(obs),(N-1)*sampling);
% bigM constant
bM=1000;


% some obstacle postprocessing
for j=1:length(obs)
    t=obs{j}.theta;
    % generate T matrix for each obstacle
    obs{j}.T=[cos(t), -sin(t);sin(t) cos(t)]*diag(obs{j}.LW/2);
    % polyehdral representaion
    obs{j}.poly=obs{j}.T*unitbox(2)+obs{j}.center;
end

%try to remove/add this one
```

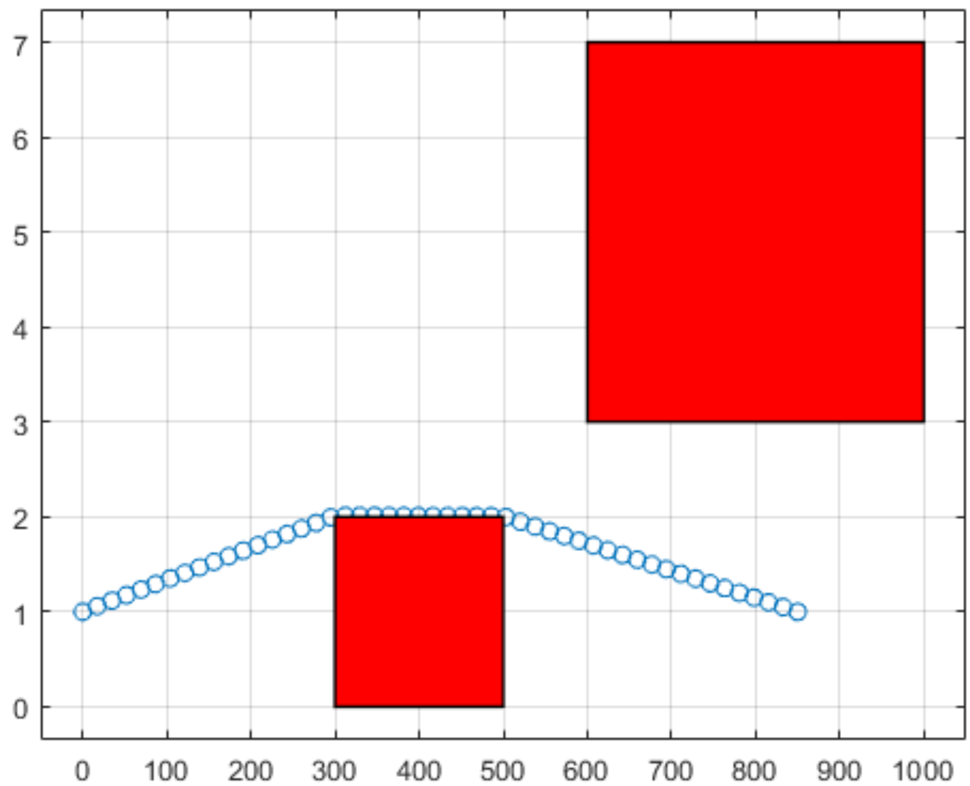
```
%z_obs{4}=[3;7];
%d_obs{4}=8;
%Qobs{4}=diag([1,10]);

%Constraints
%Setup Optimization Problem
cost = 0;
constr = [z(:,1)==z0;z(:,N)==zT];
Q=eye(2);
%constr = [zmin<=z(:,N)<= zmax, z(:,1)==z0,z(:,N)==zT];
for t = 2:N
      cost=cost+(z(:,t)-z(:,t-1))'*Q*(z(:,t)-z(:,t-1));
      constr = constr +[dzmin<= z(:,t)-z(:,t-1)<=dzmax];
      constr = constr +[zmin<= z(:,t)<=zmax];
      for k = 0:sampling-1
          for j=1:length(obs)
              zs=z(:,t-1)+k/sampling*(z(:,t)-z(:,t-1));
              [H,K]=double(obs{j}.poly);
              constr = constr +[H(1,:)*(zs)>=K(1)-(1-d((j-1)*4+1,(t-2)*sampling+k+1))*bM .
..
                                H(2,:)*(zs)>=K(2)-(1-d((j-1)*4+2,(t-2)*sampling+k+1))*bM .
..
                                H(3,:)*(zs)>=K(3)-(1-d((j-1)*4+3,(t-2)*sampling+k+1))*bM .
..
                                H(4,:)*(zs)>=K(4)-(1-d((j-1)*4+4,(t-2)*sampling+k+1))*bM .
..
                                d((j-1)*4+1,(t-2)*sampling+k+1)+d((j-1)*4+2,(t-2)*sampling
+k+1)+d((j-1)*4+3,(t-2)*sampling+k+1)+d((j-1)*4+4,(t-2)*sampling+k+1)>=1];
          end
      end
end
options = sdpsettings('solver','gurobi');
%options.ipopt=ipoptset('linear_solver','MUMPS');
solvesdp(constr,cost,options);
z_vec = double(z);


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Plotting Functions % to add title and labels
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
Academic license - for non-commercial use only
Optimize a model with 5296 rows, 4020 columns and 15880 nonzeros
Model has 198 quadratic objective terms
Variable types: 100 continuous, 3920 integer (3920 binary)
Coefficient statistics:
  Matrix range     [5e-04, 1e+03]
  Objective range  [0e+00, 0e+00]
  QObjective range [2e+00, 4e+00]
  Bounds range     [1e+00, 1e+00]
  RHS range        [1e+00, 1e+03]
Presolve removed 4328 rows and 3388 columns
Presolve time: 0.03s
Presolved: 968 rows, 632 columns, 2893 nonzeros
Presolved model has 190 quadratic objective terms
```

```
Variable types: 96 continuous, 536 integer (536 binary)
Found heuristic solution: objective 15613.676716
Found heuristic solution: objective 14853.329930

Root relaxation: objective 1.474490e+04, 810 iterations, 0.01 seconds

    Nodes    |    Current Node    |     Objective Bounds      |     Work
 Expl Unexpl |  Obj  Depth IntInf | Incumbent    BestBd   Gap | It/Node Time

     0     0 14744.8980    0  328 14853.3299 14744.8980  0.73%     -    0s
     0     0 14744.8980    0  374 14853.3299 14744.8980  0.73%     -    0s
     0     0 14744.8980    0  306 14853.3299 14744.8980  0.73%     -    0s
     0     0 14744.8980    0  306 14853.3299 14744.8980  0.73%     -    0s
     0     0 14744.8980    0  255 14853.3299 14744.8980  0.73%     -    0s
     0     0 14744.8980    0  248 14853.3299 14744.8980  0.73%     -    0s
     0     2 14744.8980    0  248 14853.3299 14744.8980  0.73%     -    0s
*  204    76             106    14853.313802 14744.9943  0.73%  11.7    0s
H  263    23                    14745.018326 14744.9943  0.00%  11.0    0s
H  274    18                    14745.006375 14744.9943  0.00%  10.6    0s

Cutting planes:
  Clique: 311
  MIR: 12

Explored 282 nodes (4730 simplex iterations) in 0.37 seconds
Thread count was 4 (of 4 available processors)

Solution count 5: 14745 14745 14853.3 ... 15613.7

Optimal solution found (tolerance 1.00e-04)
Best objective 1.474500637480e+04, best bound 1.474499433966e+04, gap 0.0001%
```

plot routine

```
figure
plot(z_vec(1,:),z_vec(2,:),'o')
hold on
for j=1:length(obs)
plot(obs{j}.T*unitbox(2)+obs{j}.center);
end
```



The time is bit slower than the NLP.

```
toc
```

```
Elapsed time is 0.741042 seconds.
```

Published with MATLAB® R2018a