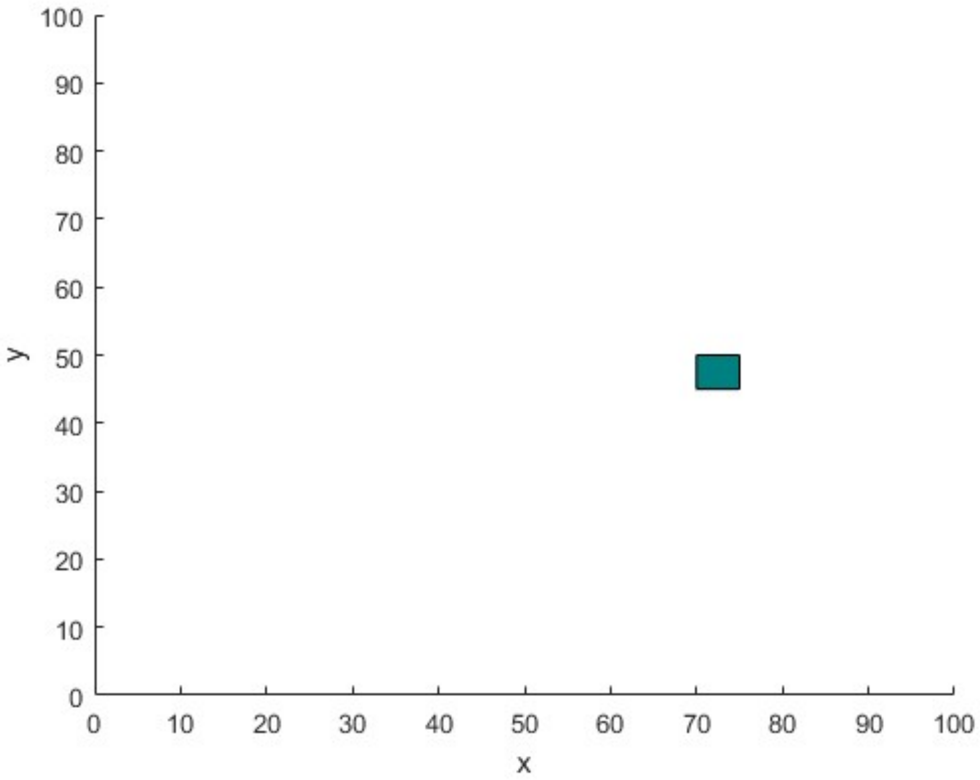
Contents

q1

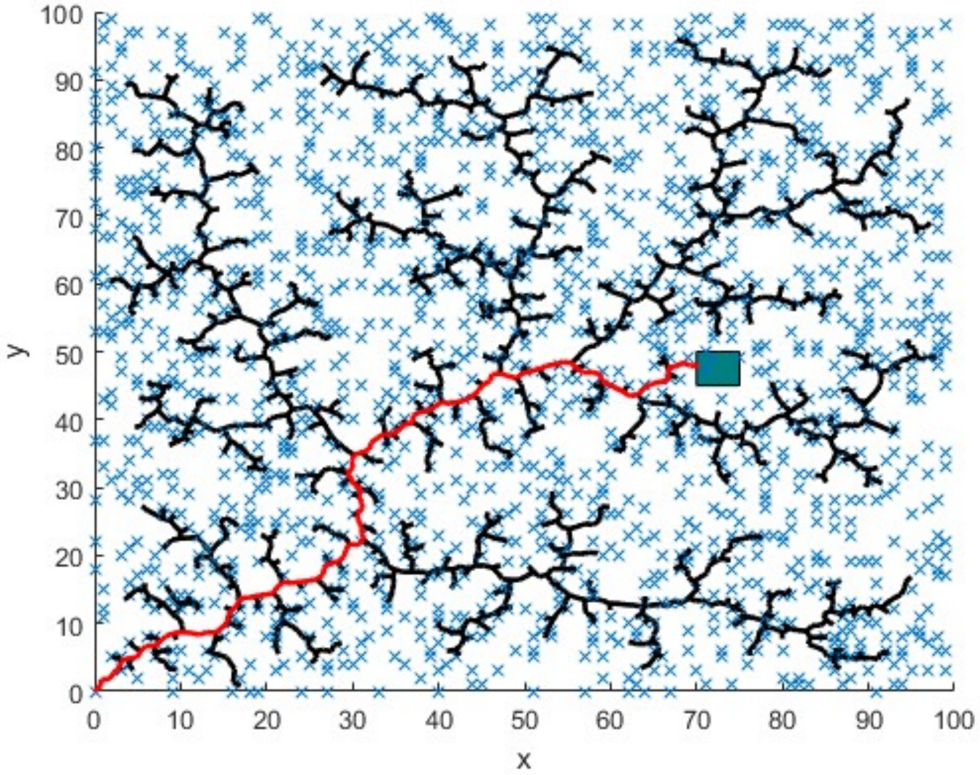
```
clearvars
close all

% 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 = 1;

% maximum iterations
numNodes = 3000;

% 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 = dist(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 = dist(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 =

        1917
```

number of nodes in the sequence that reaches goal area

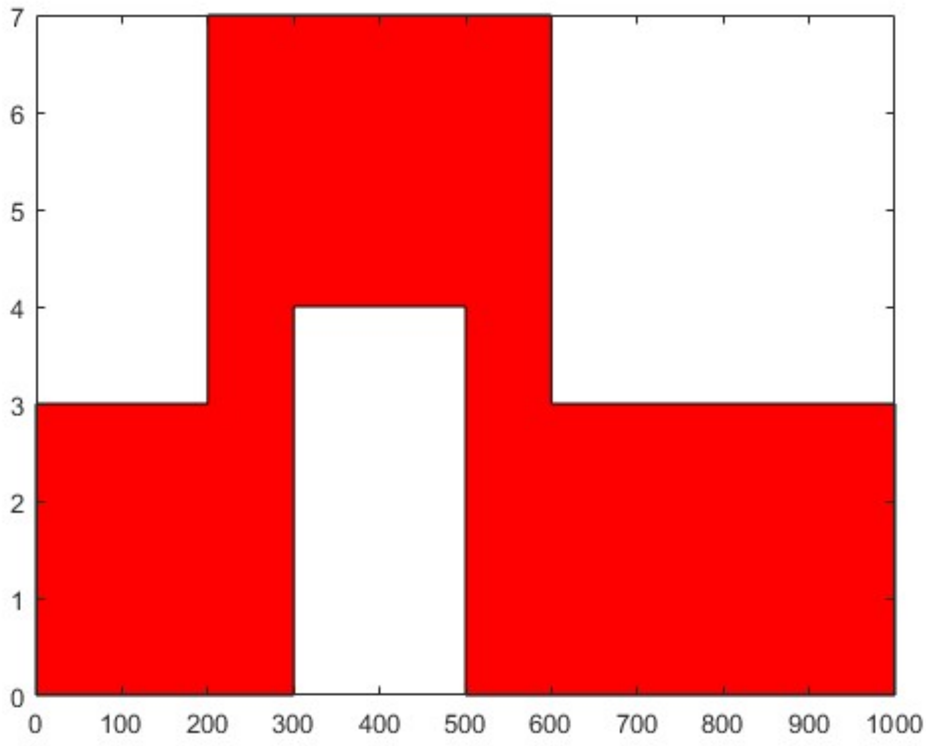
```
num_node_path
```

```
num_node_path =

   111
```

Published with MATLAB® R2017b

Contents


```
%
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 = 1;

% 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
```

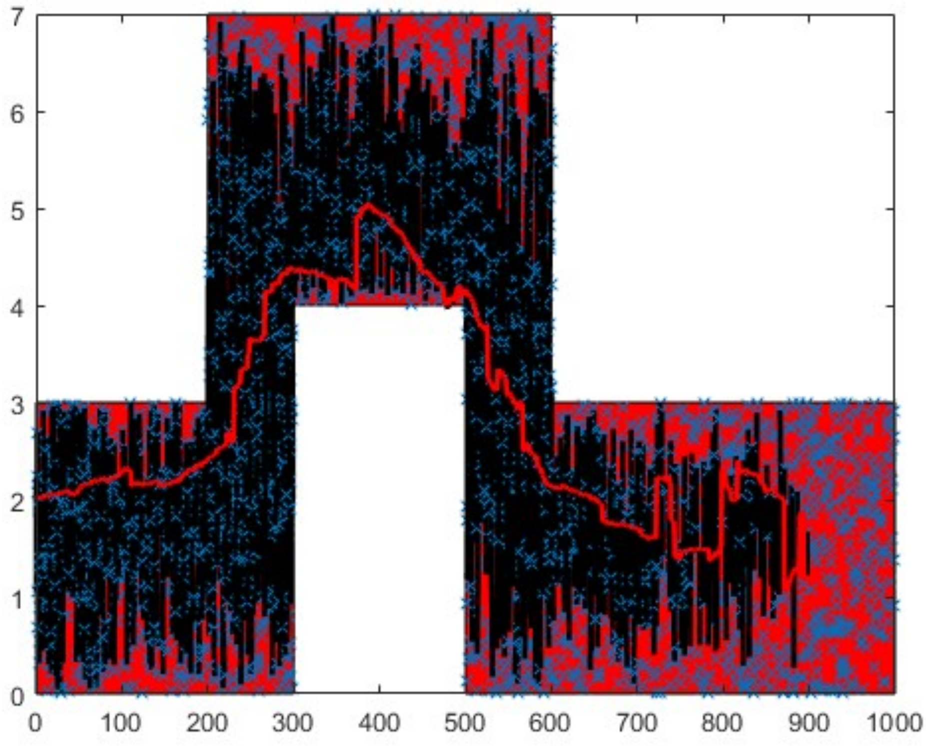



```
...
```

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])

    % Find the nearest point existing on the tree to the random point
    ndist = [];
    for j = 1:1:length(nodes)
        n = nodes(j);
        tmp = dist(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 = dist(q_new.coord, q_nearest.coord) + q_nearest.cost;
    q_new.parent = idx;

    InorOn = inpolygon(q_new.coord(1),q_new.coord(2),X,Y);
    % Append to nodes
    if InorOn == 1
        nodes = [nodes q_new];
    end
    % 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, [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', '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 =

        2827
```

number of nodes in the sequence that reaches goal area

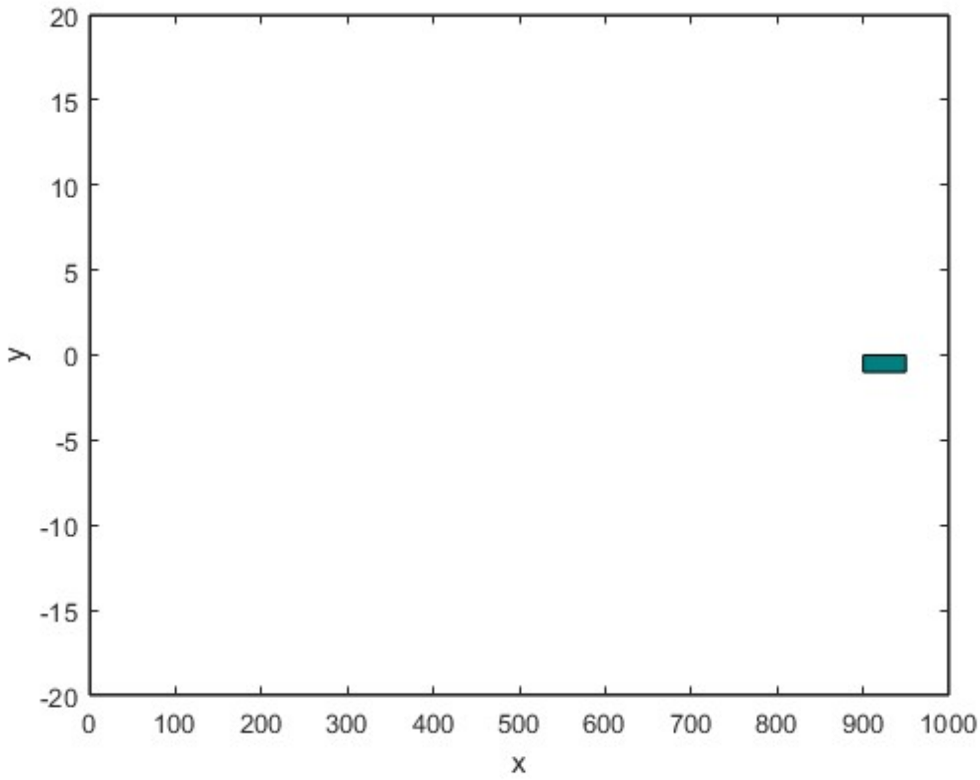
```
num_node_path
```

```
num_node_path =

   906
```

Published with MATLAB® R2017b

Contents


```
%
clearvars
close all

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

% time of moving at every step
EPST = 0.1;

% 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 0 1000 1000]; %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,'w')  % fill the safe zone with color
hold on

% plot the goal area
figure(1)
axis([0 x_max y_min y_max])
goal_area = rectangle('Position',[900,-1,50,1],'FaceColor',[0 .5 .5]);
xlabel('x')
ylabel('y')
hold on
```



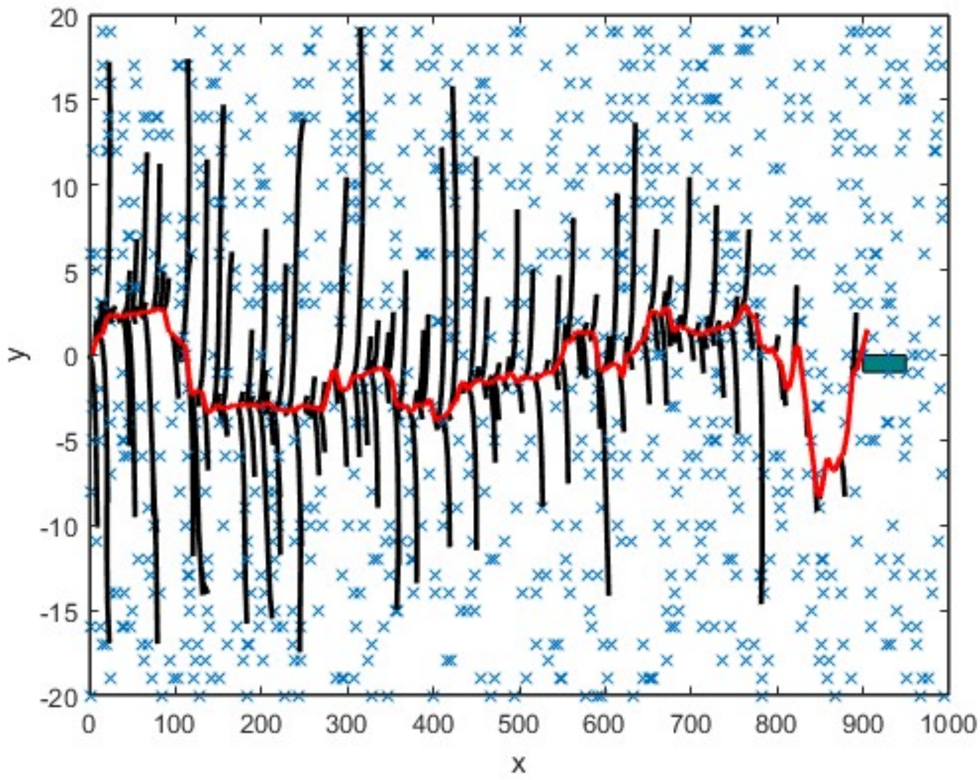
define the vehicle parameters

```
vx = 30;
L = 3;
```

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*2)-y_max floor(rand(1)*2*phi_max)-phi_
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 = dist(n.coord(1:2), q_rand);
        ndist = [ndist tmp];
    end
    [mini_distance, idx] = min(ndist);
    q_nearest = nodes(idx);

    %brute force to check all the possible steering angles, and assign the
    %closet to q_new

    k = 1;
    tempdist = [];
    q_newPossible = [];
    for delta = -20:2:20
```

```
        deltaRad = delta*pi/180;
        dxdt = @(t,x) kinematicsModel(x, deltaRad, vx, L);
        [tsol, xsol] = ode45(dxdt,[0,EPST],q_nearest.coord);

        InorOn = all(inpolygon(xsol(:,1),xsol(:,2),X,Y)) && all(xsol(:,3)<=pi) && all(xsol(:,
3)>=-pi);

        k=k+1;
        if InorOn == 1
            q_newPossible = [q_newPossible; xsol(end,:)];
            tempdist = [tempdist dist(q_newPossible(end,1:2), q_rand)];
        end

    end
    [mini_distance2, idx2] = min(tempdist);

    if isempty(q_newPossible(idx2,:))
        continue;
    end

    q_new.coord = q_newPossible(idx2,:);
    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 = dist(q_new.coord, q_nearest.coord) + q_nearest.cost;
    q_new.parent = idx;

    InorOn = inpolygon(q_new.coord(1),q_new.coord(2),X,Y);
    % Append to nodes
    if InorOn == 1
        nodes = [nodes q_new];
    end
    % 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, [900,1,50,0.5]) && q_new.coord(3)>=-pi/6 &&
 q_new.coord(3)<=pi/6
        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 =

   866
```

number of nodes in the sequence that reaches goal area

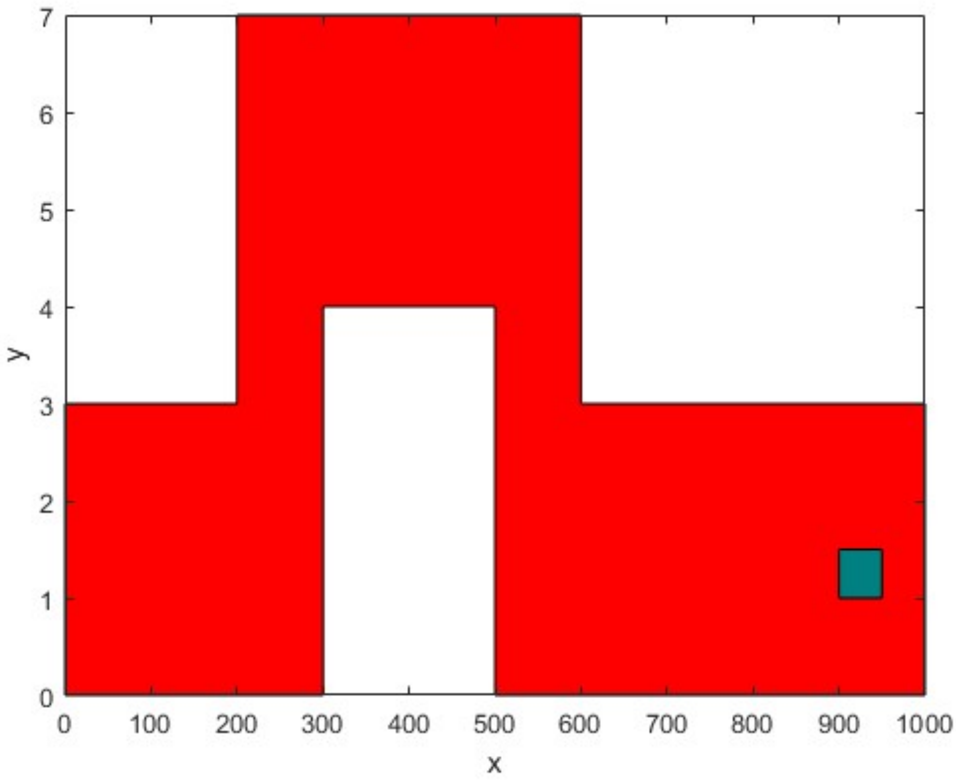
```
num_node_path
```

```
num_node_path =

   306
```

Published with MATLAB® R2017b

Contents


```
%
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;
phi_max = pi;
phi_min = -pi;

% time of moving at every step
EPST = 0.1;

% maximum iterations
numNodes = 6000;

% attributions of starting point
q_start.coord = [0 2 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
figure(1)
axis([0 x_max 0 y_max])
goal_area = rectangle('Position',[900,1,50,0.5],'FaceColor',[0 .5 .5]);
xlabel('x')
ylabel('y')
hold on
```



define the vehicle parameters

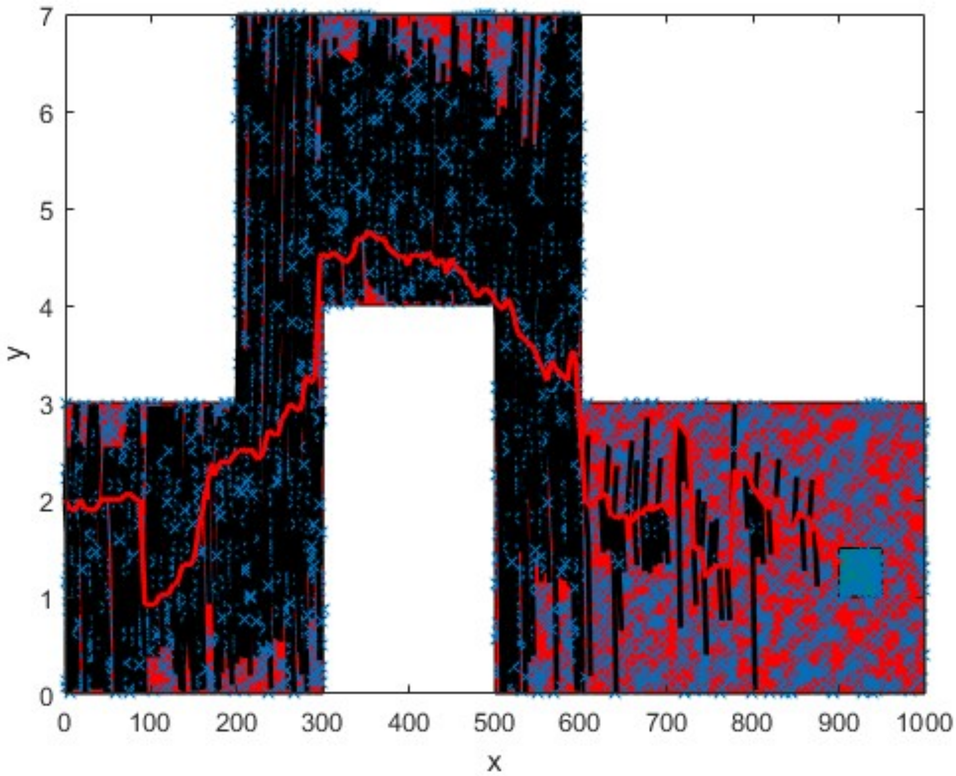
```
vx = 30;
L = 3;
```

grow the tree

```
for i = 1:1:numNodes

    %q_rand = [floor(rand(1)*x_max) floor(rand(1)*y_max*2)-y_max floor(rand(1)*2*phi_max)-phi
_max]';
    %plot(q_rand(1), q_rand(2), 'x', 'Color',  [0 0.4470 0.7410])

    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 floor(rand(1)*2*phi_max)-phi_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])

    % Find the nearest point existing on the tree to the random point
    ndist = [];
    for j = 1:1:length(nodes)
        n = nodes(j);
        tmp = dist(n.coord(1:2), q_rand);
        ndist = [ndist tmp];
    end
    [mini_distance, idx] = min(ndist);
    q_nearest = nodes(idx);
```

```
    %brute force to check all the possible steering angles, and assign the
    %closet to q_new

    k = 1;
    tempdist = [];
    q_newPossible = [];
    for delta = -20:2:20
        deltaRad = delta*pi/180;
        dxdt = @(t,x) kinematicsModel(x, deltaRad, vx, L);
        [tsol, xsol] = ode45(dxdt,[0,EPST],q_nearest.coord);

        InorOn = all(inpolygon(xsol(:,1),xsol(:,2),X,Y)) && all(xsol(:,3)<=pi) && all(xsol(:,
3)>=-pi);

        k=k+1;
        if InorOn == 1
            q_newPossible = [q_newPossible; xsol(end,:)];
            tempdist = [tempdist dist(q_newPossible(end,1:2), q_rand)];
        end

    end
    [mini_distance2, idx2] = min(tempdist);

    if isempty(q_newPossible(idx2,:))
        continue;
    end

    q_new.coord = q_newPossible(idx2,:);
    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 = dist(q_new.coord, q_nearest.coord) + q_nearest.cost;
    q_new.parent = idx;

    InorOn = inpolygon(q_new.coord(1),q_new.coord(2),X,Y);
    % Append to nodes
    if InorOn == 1
        nodes = [nodes q_new];
    end
    % 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(1:2), [900,1,50,0.5]) && q_new.coord(3)>=-pi
/6 && q_new.coord(3)<=pi/6
        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 =

        2345
```

number of nodes in the sequence that reaches goal area

```
num_node_path
```

```
num_node_path =

   302
```

Published with MATLAB® R2017b

Contents


```
clc
clear
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Francesco Borrelli ME C231A 2015
% Kinematic Navigation
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
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];
```

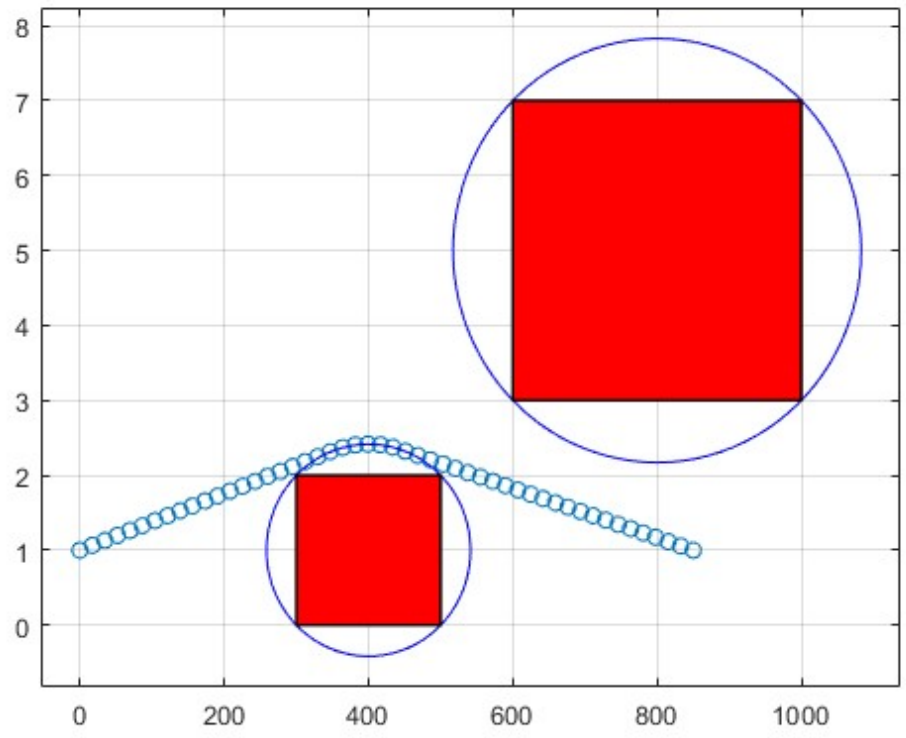
```
      constr = constr + [zmin<=z(:,t)<= zmax];
      for k = 0:sampling-1
          for j=1:length(obs)
              xs=z(:,t-1)+k/sampling*(z(:,t)-z(:,t-1));
              %constr = constr + [norm(inv(obs{j}.T)*(xs - obs{j}.center),2)>=sqrt(2)];
              constr = constr +[(xs-obs{j}.center)'*inv(obs{j}.T)'*inv(obs{j}.T)*(xs-obs{j}.c
enter)>=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
```

```
******************************************************************************
This program contains Ipopt, a library for large-scale nonlinear optimization.
 Ipopt is released as open source code under the Eclipse Public License (EPL).
         For more information visit http://projects.coin-or.org/Ipopt
******************************************************************************

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....: 332

                                   (scaled)                 (unscaled)
Objective...............:   1.4745068842705905e+03    1.4745068842705905e+04
Dual infeasibility......:   3.3529033117656135e-08    3.3529033117656133e-07
Constraint violation....:   0.0000000000000000e+00    0.0000000000000000e+00
Complementarity.........:   1.0000000000000003e-11    1.0000000000000002e-10
Overall NLP error.......:   3.3529033117656135e-08    3.3529033117656133e-07
```

```
Number of objective function evaluations             = 1001
Number of objective gradient evaluations             = 333
Number of equality constraint evaluations            = 0
Number of inequality constraint evaluations          = 1001
Number of equality constraint Jacobian evaluations   = 0
Number of inequality constraint Jacobian evaluations = 333
Number of Lagrangian Hessian evaluations             = 0
Total CPU secs in IPOPT (w/o function evaluations)   =      1.324
Total CPU secs in NLP function evaluations           =      0.736

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
```



Contents


```
clc
clear
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Francesco Borrelli ME C231A 2015
% Kinematic Navigation
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
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
```

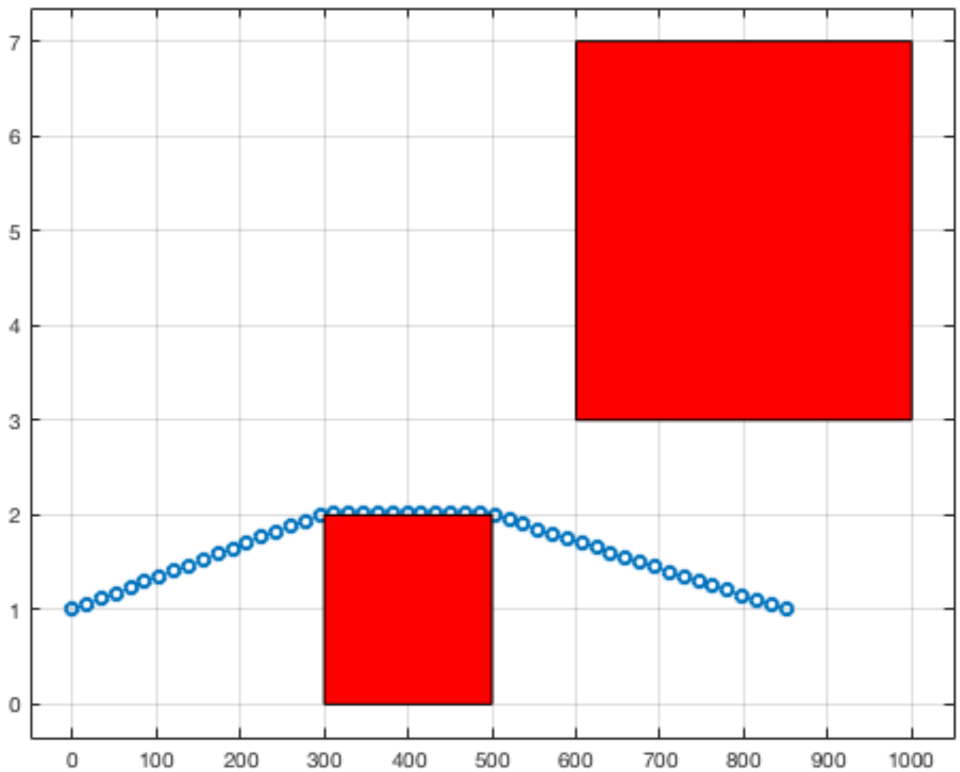
```
%z_obs{4}=[3;7];
%d_obs{4}=8;
%Qobs{4}=diag([1,10]);

%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
*  203    77             106    14853.313802 14744.9943  0.73%  11.7    0s
H  267    22                    14745.018326 14744.9943  0.00%  10.8    0s
H  278    17                    14745.006375 14744.9943  0.00%  10.5    0s

Cutting planes:
  Clique: 311
  MIR: 12

Explored 287 nodes (4744 simplex iterations) in 0.30 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
```



Published with MATLAB® R2017b