Contents

Final Exam - ME C231B Jun Zeng

We also include all seperate files in the './allfiles/' folder for testing

```
addpath('./allfiles/');
```

Vehicle Navigation

1.a

Nothing to hand in

1.b

```
clear all
clear yalmip
```

Car Parameters

```
lr  = 1.7;
lf = 1.1;
L=lf+lr;
```

Ego car G and g, not used in this code

```
width=1; %car width
G = [-1 0; 1 0;0 -1; 0 1]; g = [0; lr+lf; width; width]; % polyhedron Gy<=g as in the paper
```

Input constraints

```
umin(2)=-pi;
delta_max=25*pi/180; % Max steering
umin(1)=(lf+lr)/tan(delta_max);
model.u.min=umin;
```

State constraints

```
model.z.min=-[20;10;2*pi];
model.z.max=[20;20;2*pi];

%Initial, terminal conditions and horizon
z0 = [-10;10;0];
zT = [0;0;-pi/2];
N=4; %navigation  horizon


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Obstacle list
i=1;
obs{i}.center=[-5;0];
obs{i}.LW=[2;8];
obs{i}.theta=0/180; %(in radiants)
i=i+1;
obs{i}.center=[5;0];
obs{i}.LW=[2;8];
obs{i}.theta=0/180; %(in radiants)
i=i+1;
obs{i}.center=[0;14];
obs{i}.LW=[1;8];
obs{i}.theta=0/180; %(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;
    [AA{j},bb{j}]=double(obs{j}.poly);
    lambda{j} = sdpvar(size(AA{j},1),N,'full');
end
```

Setup the Navigation Problem

```
options = sdpsettings('solver','ipopt');
%options = sdpsettings('solver','fmincon','verbose',1);

z = sdpvar(3,N+1);
u = sdpvar(2,N);
constr = [z(:,N+1)==zT, z(:,1) == z0];
cost = 0;
SampleNum = 10;
for k = 1:N
    constr = constr+...
         [z(1,k+1) == z(1,k)-u(1,k)*sin(z(3,k))+u(1,k)*sin(z(3,k)+u(2,k)),...
          z(2,k+1) == z(2,k)+u(1,k)*cos(z(3,k))-u(1,k)*cos(z(3,k)+u(2,k)),...
          z(3,k+1) == z(3,k)+u(2,k),...
          model.u.min(1) <= u(1,k),model.u.min(2) <= u(2,k),u(2,k) <= -model.u.min(2),...
          model.z.min <= z(:,k+1),z(:,k+1)<=model.z.max];
    cost = cost + (u(2,k)*u(1,k))^2;
    for p = 1:SampleNum-1
        for q = 1:size(obs,2)
            zs = z(:,k)+p/SampleNum*(z(:,k+1)-z(:,k));
            A = AA{q}; b = bb{q};
            constr = constr + [(AA{q}*zs(1:2)-bb{q})'*lambda{q}(:,k) >= 0];
            constr = constr + [lambda{q}(:,k)'*AA{q}*AA{q}'*lambda{q}(:,k)<=1];
            constr = constr + [lambda{q}(:,k) >= 0];
        end
    end
end
```

Compute Navigation Solution

```
optimize(constr,cost,options)
zdata = double(z);
udata = double(u);
```

```
******************************************************************************
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
******************************************************************************
```

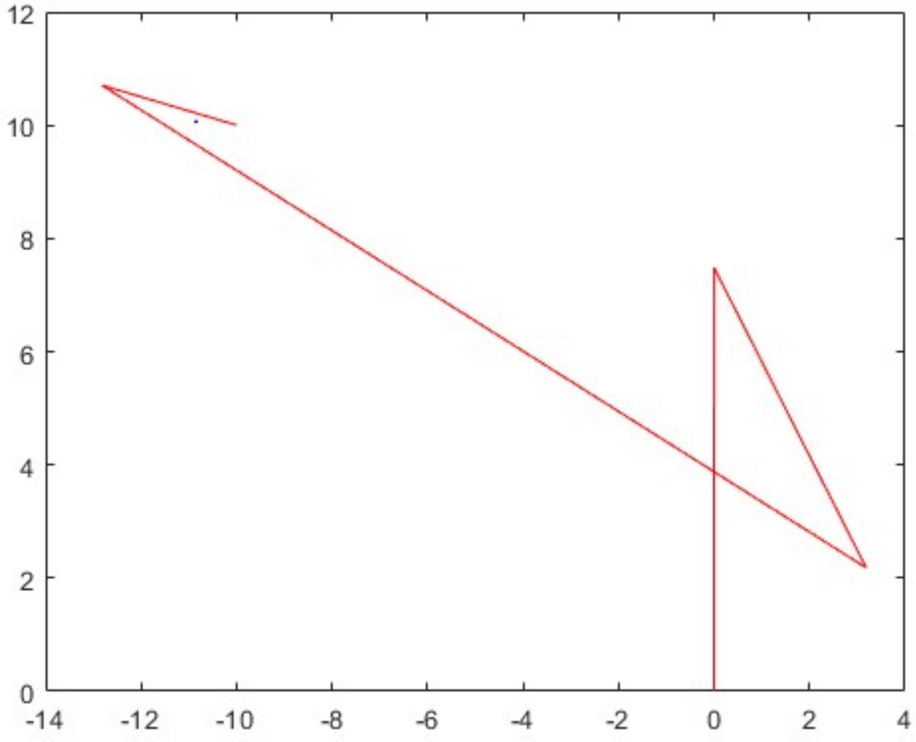
```
Total number of variables............................:       73
                     variables with only lower bounds:       52
                variables with lower and upper bounds:       21
                     variables with only upper bounds:        0
Total number of equality constraints.................:       20
Total number of inequality constraints...............:      216
        inequality constraints with only lower bounds:        0
   inequality constraints with lower and upper bounds:        0
        inequality constraints with only upper bounds:      216


Number of Iterations....: 1500

                                   (scaled)                 (unscaled)
Objective...............:   4.3585652874278037e+02    4.3585652874278037e+02
Dual infeasibility......:   5.4327122108688286e-04    5.4327122108688286e-04
Constraint violation....:   3.9999576983973384e-09    3.9999576983973384e-09
Complementarity.........:   1.0006851544561848e-11    1.0006851544561848e-11
Overall NLP error.......:   1.6276300928888377e-05    5.4327122108688286e-04


Number of objective function evaluations             = 1951
Number of objective gradient evaluations             = 1501
Number of equality constraint evaluations            = 1951
Number of inequality constraint evaluations          = 1951
Number of equality constraint Jacobian evaluations   = 1501
Number of inequality constraint Jacobian evaluations = 1501
Number of Lagrangian Hessian evaluations             = 0
Total CPU secs in IPOPT (w/o function evaluations)   =      2.592
Total CPU secs in NLP function evaluations           =      9.215

EXIT: Maximum Number of Iterations Exceeded.

ans =

  struct with fields:

    yalmiptime: 2.0176
    solvertime: 11.8184
          info: 'Maximum iterations or time limit exceeded (IPOPT)'
       problem: 3
```

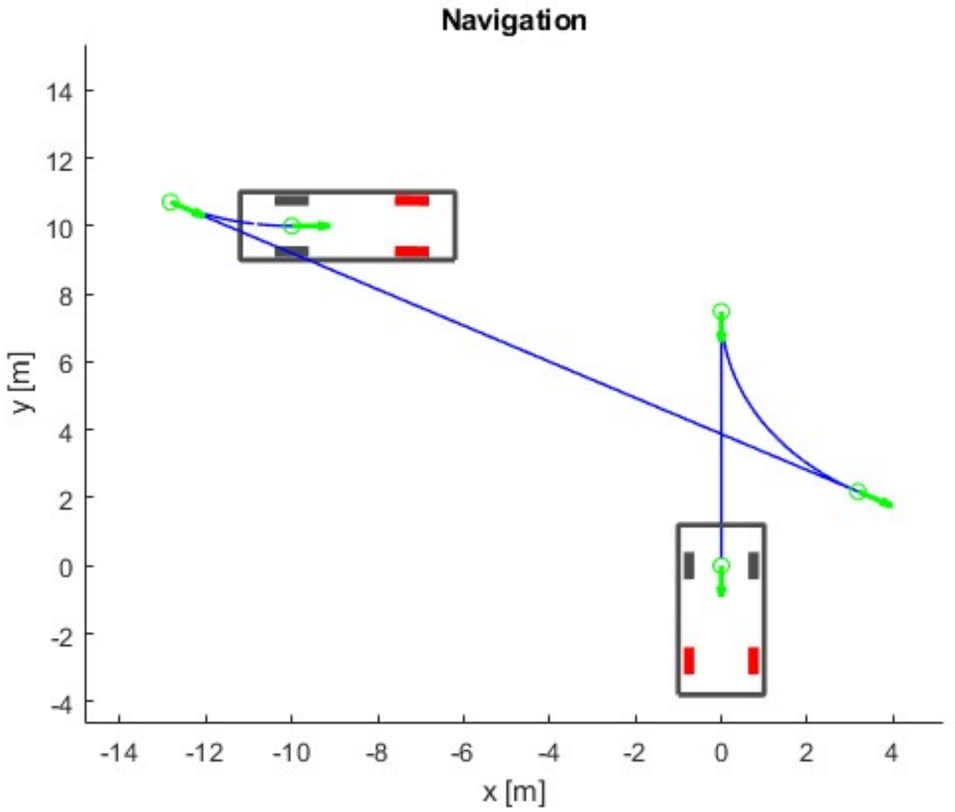
Plot Solution

```
figure
plot(zdata(1,:),zdata(2,:),'r')
hold on
car_plot(double(u),z0)
```

```
auto =

  struct with fields:
```

```
     w: 2
    db: 1.2000
    df: 1
     l: 5


fig = 

  Figure (2) with properties:

      Number: 2
        Name: ''
       Color: [0.9400 0.9400 0.9400]
    Position: [360 502 560 420]
       Units: 'pixels'

  Use GET to show all properties
```

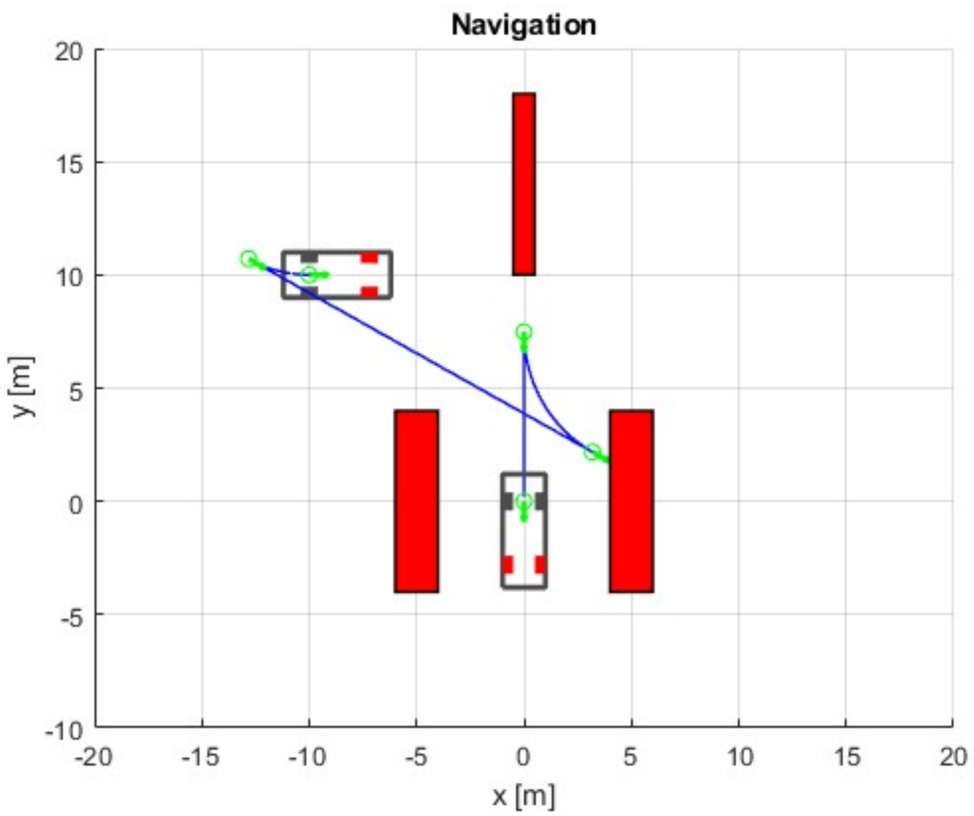




Plot obstacles

```
for j=1:length(obs)
plot(polytope(AA{j},bb{j}));
```

```
end
axis([model.z.min(1) model.z.max(1) model.z.min(2) model.z.max(2)])
```



1.c

```
clear all
clear yalmip
```

Car Parameters

```
lr  = 1.7;
lf = 1.1;
L=lf+lr;
```

Ego car G and g, not used in this code

```
width=1; %car width
G = [-1 0; 1 0;0 -1; 0 1]; g = [0; lr+lf; width; width]; % polyhedron Gy<=g as in the paper
```

Input constraints

```
umin(2)=-pi;
delta_max=25*pi/180; % Max steering
umin(1)=(lf+lr)/tan(delta_max);
model.u.min=umin;
```

State constraints

```
model.z.min=-[20;10;2*pi];
model.z.max=[20;20;2*pi];

%Initial, terminal conditions and horizon
z0 = [-10;10;0];
zT = [0;0;-pi/2];
N=6; %navigation  horizon


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Obstacle list
i=1;
obs{i}.center=[-5;0];
obs{i}.LW=[2;8];
obs{i}.theta=0/180; %(in radiants)
i=i+1;
obs{i}.center=[5;0];
obs{i}.LW=[2;8];
obs{i}.theta=0/180; %(in radiants)
i=i+1;
obs{i}.center=[0;14];
obs{i}.LW=[1;8];
obs{i}.theta=0/180; %(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;
    [AA{j},bb{j}]=double(obs{j}.poly);
    lambda{j} = sdpvar(size(AA{j},1),N,'full');
end
```

Setup the Navigation Problem

```
options = sdpsettings('solver','ipopt');
%options = sdpsettings('solver','fmincon','verbose',1);

z = sdpvar(3,N+1);
u = sdpvar(2,N);
constr = [z(:,N+1)==zT, z(:,1) == z0];
cost = 0;
SampleNum = 10;
for k = 1:N
    constr = constr+...
         [z(1,k+1) == z(1,k)-u(1,k)*sin(z(3,k))+u(1,k)*sin(z(3,k)+u(2,k)),...
          z(2,k+1) == z(2,k)+u(1,k)*cos(z(3,k))-u(1,k)*cos(z(3,k)+u(2,k)),...
          z(3,k+1) == z(3,k)+u(2,k),...
          model.u.min(1) <= u(1,k),model.u.min(2) <= u(2,k),u(2,k) <= -model.u.min(2),...
```

```
        model.z.min <= z(:,k+1),z(:,k+1)<=model.z.max];
    cost = cost + (u(2,k)*u(1,k))^2;
    for p = 1:SampleNum-1
        for q = 1:size(obs,2)
            zs = z(:,k)+p/SampleNum*(z(:,k+1)-z(:,k));
            A = AA{q}; b = bb{q};
            constr = constr + [(AA{q}*zs(1:2)-bb{q})'*lambda{q}(:,k) >= 0];
            constr = constr + [lambda{q}(:,k)'*AA{q}*AA{q}'*lambda{q}(:,k)<=1];
            constr = constr + [lambda{q}(:,k) >= 0];
        end
    end
end
```

Compute Navigation Solution

```
optimize(constr,cost,options)
zdata = double(z);
udata = double(u);
```

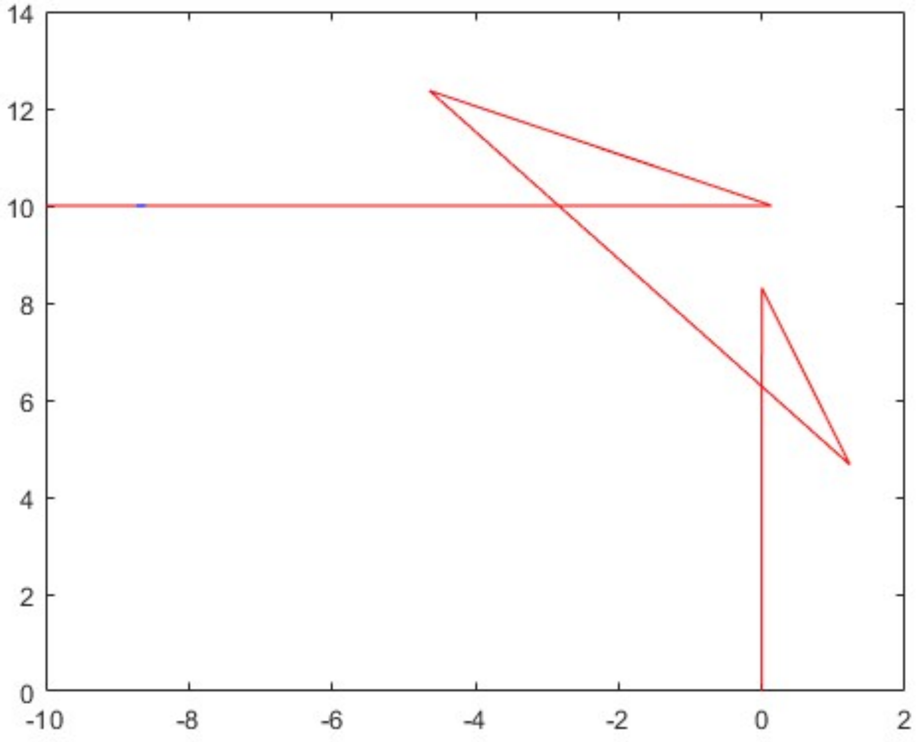
```
******************************************************************************
This program contains Ipopt, a library for large-scale nonlinear optimization.
 Ipopt is released as open source code under the Eclipse Public License (EPL).
         For more information visit http://projects.coin-or.org/Ipopt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

Total number of variables............................:      111
                     variables with only lower bounds:       78
                variables with lower and upper bounds:       33
                     variables with only upper bounds:        0
Total number of equality constraints.................:       30
Total number of inequality constraints...............:      324
        inequality constraints with only lower bounds:        0
   inequality constraints with lower and upper bounds:        0
        inequality constraints with only upper bounds:      324


Number of Iterations....: 1500

                                   (scaled)                 (unscaled)
Objective...............:   2.6000440382522561e+02    2.6000440382522561e+02
Dual infeasibility......:   7.9872023746613759e-06    7.9872023746613759e-06
Constraint violation....:   7.2759576141834259e-12    7.2759576141834259e-12
Complementarity.........:   1.5679231827368443e-09    1.5679231827368443e-09
Overall NLP error.......:   2.6529681864100602e-07    7.9872023746613759e-06


Number of objective function evaluations             = 2484
Number of objective gradient evaluations             = 1496
Number of equality constraint evaluations            = 2484
Number of inequality constraint evaluations          = 2484
Number of equality constraint Jacobian evaluations   = 1506
Number of inequality constraint Jacobian evaluations = 1506
Number of Lagrangian Hessian evaluations             = 0
Total CPU secs in IPOPT (w/o function evaluations)   =      3.388
```

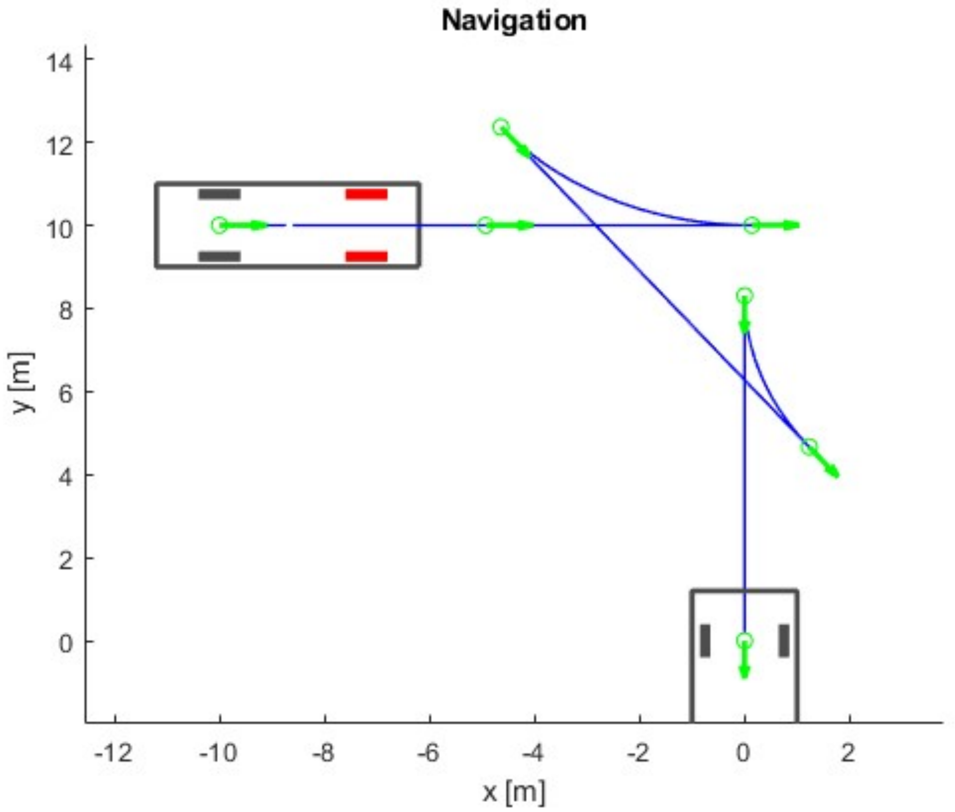
```
Total CPU secs in NLP function evaluations           =     16.905

EXIT: Maximum Number of Iterations Exceeded.

ans =

  struct with fields:

    yalmiptime: 2.4003
    solvertime: 20.3037
          info: 'Maximum iterations or time limit exceeded (IPOPT)'
       problem: 3
```

Plot Solution

```
figure
plot(zdata(1,:),zdata(2,:),'r')
hold on
car_plot(double(u),z0)
```

```
auto =

  struct with fields:

     w: 2
    db: 1.2000
    df: 1
     l: 5


fig =

  Figure (4) with properties:

      Number: 4
        Name: ''
       Color: [0.9400 0.9400 0.9400]
    Position: [360 502 560 420]
       Units: 'pixels'

  Use GET to show all properties
```

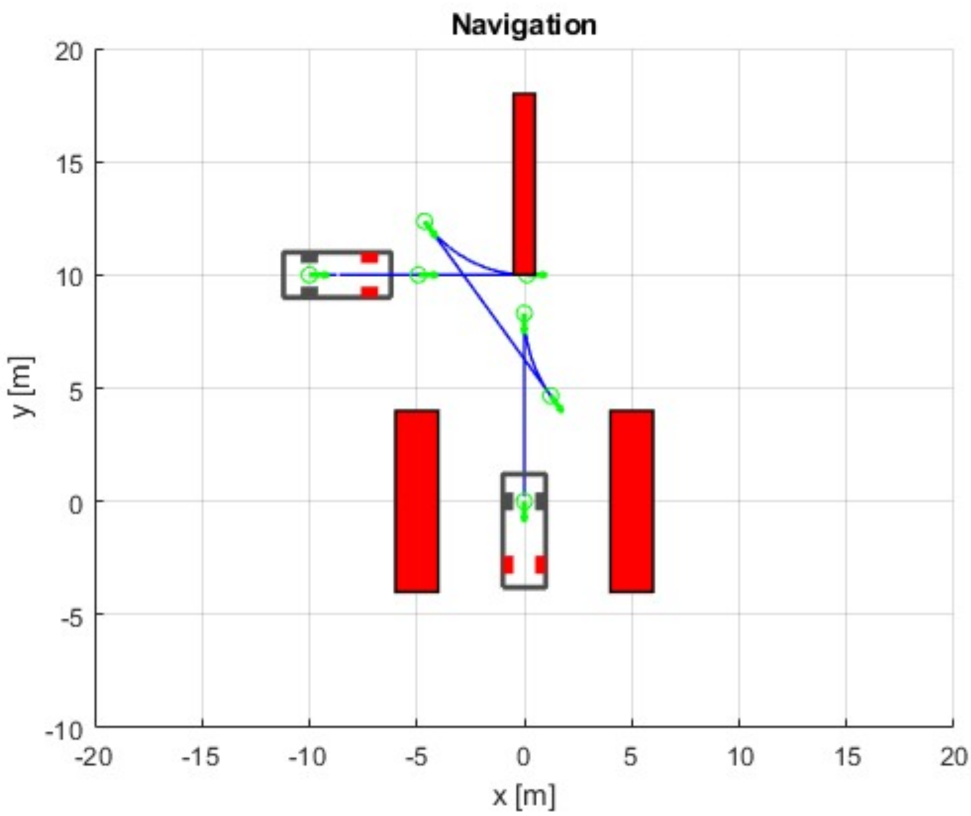




Plot obstacles

```
for j=1:length(obs)
plot(polytope(AA{j},bb{j}));
```

```
end
axis([model.z.min(1) model.z.max(1) model.z.min(2) model.z.max(2)])
```



1.d

```
clear all
clear yalmip
```

Car Parameters

```
lr  = 1.7;
lf = 1.1;
L=lf+lr;
```

Ego car G and g, not used in this code

```
width=1; %car width
G = [-1 0; 1 0;0 -1; 0 1]; g = [0; lr+lf; width; width]; % polyhedron Gy<=g as in the paper
```

Input constraints

```
umin(2)=-pi;
delta_max=25*pi/180; % Max steering
umin(1)=(lf+lr)/tan(delta_max);
model.u.min=umin;
```

State constraints

```
model.z.min=-[20;10;2*pi];
model.z.max=[20;20;2*pi];

%Initial, terminal conditions and horizon
z0 = [-10;10;0];
zT = [0;0;-pi/2];
N=6; %navigation  horizon


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Obstacle list
i=1;
obs{i}.center=[-5;0];
obs{i}.LW=[2;8];
obs{i}.theta=0/180; %(in radiants)
i=i+1;
obs{i}.center=[5;0];
obs{i}.LW=[2;8];
obs{i}.theta=0/180; %(in radiants)
i=i+1;
obs{i}.center=[0;14];
obs{i}.LW=[1;8];
obs{i}.theta=0/180; %(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;
    [AA{j},bb{j}]=double(obs{j}.poly);
    lambda{j} = sdpvar(size(AA{j},1),N,'full');
    s{j} = sdpvar(1,N);
end
```

Setup the Navigation Problem

```
options = sdpsettings('solver','ipopt');
%options = sdpsettings('solver','fmincon','verbose',1);

z = sdpvar(3,N+1);
u = sdpvar(2,N);
constr = [z(:,N+1)==zT, z(:,1) == z0];
cost = 0;
SampleNum = 10;
wf = 1000;
for k = 1:N
    tempCost = 0;
    constr = constr+...
         [z(1,k+1) == z(1,k)-u(1,k)*sin(z(3,k))+u(1,k)*sin(z(3,k)+u(2,k)),...
```

```
        z(2,k+1) == z(2,k)+u(1,k)*cos(z(3,k))-u(1,k)*cos(z(3,k)+u(2,k)),...
        z(3,k+1) == z(3,k)+u(2,k),...
        model.u.min(1) <= u(1,k),model.u.min(2) <= u(2,k),u(2,k) <= -model.u.min(2),...
        model.z.min <= z(:,k+1),z(:,k+1)<=model.z.max];
        cost = cost + (u(2,k)*u(1,k))^2;

    for q = 1:length(obs)
        for p = 1:SampleNum-1
            zs = z(:,k)+p/SampleNum*(z(:,k+1)-z(:,k));
            A = AA{q}; b = bb{q};
            constr = constr + [(AA{q}*zs(1:2)-bb{q})'*lambda{q}(:,k) >= -s{q}(k)];
            % avoid unnecessary warning of inequality
            tempCost = tempCost + s{q}(k);
        end
        constr = constr + [lambda{q}(:,k)'*AA{q}*AA{q}'*lambda{q}(:,k)==1];
        constr = constr + [lambda{q}(:,k) >= 0];
        constr = constr + [s{q}(k) >= 0];
    end
    cost = cost + wf*tempCost;
end
```

Compute Navigation Solution

```
optimize(constr,cost,options)
zdata = double(z);
udata = double(u);
```

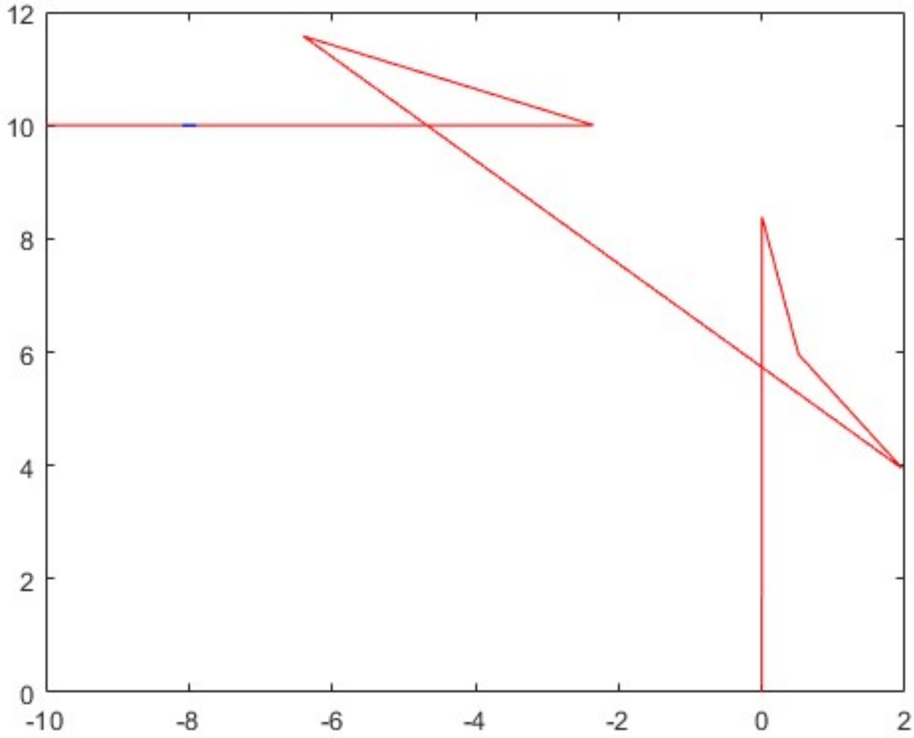
```
******************************************************************************
This program contains Ipopt, a library for large-scale nonlinear optimization.
 Ipopt is released as open source code under the Eclipse Public License (EPL).
         For more information visit http://projects.coin-or.org/Ipopt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

Total number of variables............................:      129
                     variables with only lower bounds:       96
                variables with lower and upper bounds:       33
                     variables with only upper bounds:        0
Total number of equality constraints.................:       48
Total number of inequality constraints...............:      162
        inequality constraints with only lower bounds:        0
   inequality constraints with lower and upper bounds:        0
        inequality constraints with only upper bounds:      162


Number of Iterations....: 1500

                                   (scaled)                 (unscaled)
Objective...............:   3.2131583832581865e+00    2.8918425449323678e+02
Dual infeasibility......:   1.7510834928432477e+00    1.5759751435589229e+02
Constraint violation....:   5.8501061326463155e-02    5.8501061326463155e-02
Complementarity.........:   2.4246556278715663e-06    2.1821900650844097e-04
Overall NLP error.......:   1.7510834928432477e+00    1.5759751435589229e+02
```

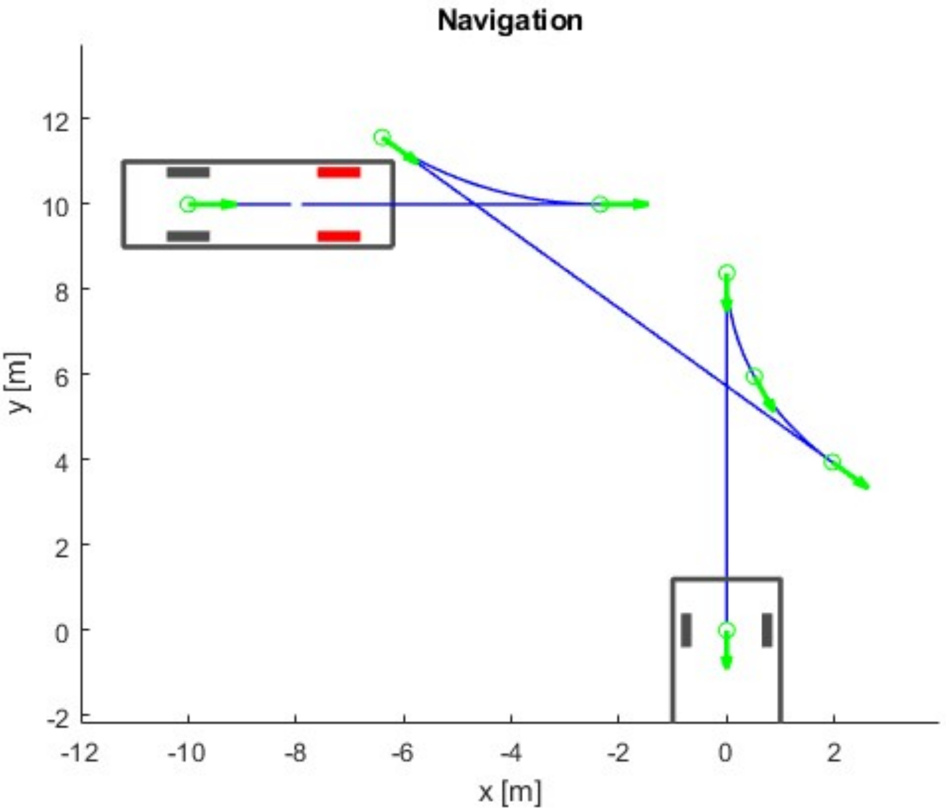
```
Number of objective function evaluations             = 3044
Number of objective gradient evaluations             = 1404
Number of equality constraint evaluations            = 3048
Number of inequality constraint evaluations          = 3048
Number of equality constraint Jacobian evaluations   = 1507
Number of inequality constraint Jacobian evaluations = 1507
Number of Lagrangian Hessian evaluations             = 0
Total CPU secs in IPOPT (w/o function evaluations)   =      2.989
Total CPU secs in NLP function evaluations           =     29.950

EXIT: Maximum Number of Iterations Exceeded.

ans = 

  struct with fields:

    yalmiptime: 1.9030
    solvertime: 32.9500
          info: 'Maximum iterations or time limit exceeded (IPOPT)'
       problem: 3
```

Plot Solution

```
figure
plot(zdata(1,:),zdata(2,:),'r')
hold on
car_plot(double(u),z0)
```

```
auto = 

  struct with fields:

     w: 2
    db: 1.2000
    df: 1
     l: 5


fig = 

  Figure (6) with properties:

      Number: 6
        Name: ''
       Color: [0.9400 0.9400 0.9400]
    Position: [360 502 560 420]
       Units: 'pixels'

  Use GET to show all properties
```

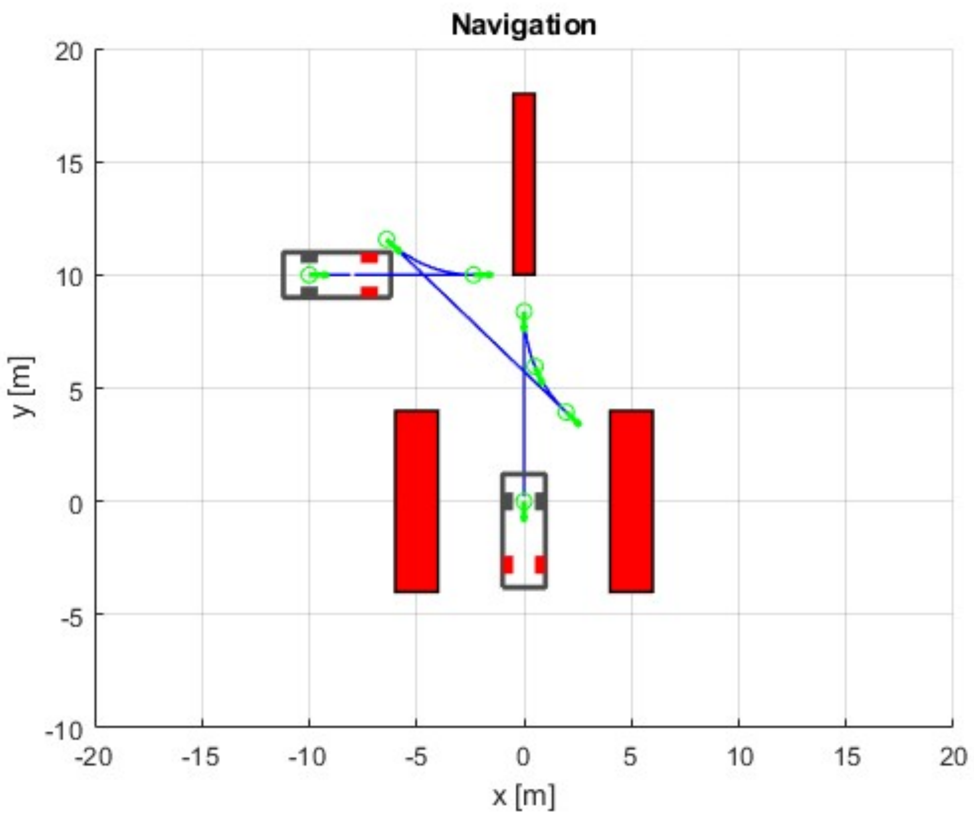




Plot obstacles

```
for j=1:length(obs)
plot(polytope(AA{j},bb{j}));
```

```
end
axis([model.z.min(1) model.z.max(1) model.z.min(2) model.z.max(2)])
```



1.e

```
clear all
clear yalmip
```

Car Parameters

```
lr  = 1.7;
lf = 1.1;
L=lf+lr;
```

Ego car G and g, not used in this code

```
width=1; %car width
G = [-1 0; 1 0;0 -1; 0 1]; g = [0; lr+lf; width; width]; % polyhedron Gy<=g as in the paper
```

Input constraints

```
umin(2)=-pi;
delta_max=25*pi/180; % Max steering
umin(1)=(lf+lr)/tan(delta_max);
model.u.min=umin;
```

State constraints

```
model.z.min=-[20;10;2*pi];
model.z.max=[20;20;2*pi];

%Initial, terminal conditions and horizon
z0 = [-10;10;0];
zT = [0;0;-pi/2];
N=4; %navigation  horizon


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Obstacle list
i=1;
obs{i}.center=[-5;0];
obs{i}.LW=[2;8];
obs{i}.theta=0/180; %(in radiants)
i=i+1;
obs{i}.center=[5;0];
obs{i}.LW=[2;8];
obs{i}.theta=0/180; %(in radiants)
i=i+1;
obs{i}.center=[0;14];
obs{i}.LW=[1;8];
obs{i}.theta=0/180; %(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;
    [AA{j},bb{j}]=double(obs{j}.poly);
    lambda{j} = sdpvar(size(AA{j},1),N,'full');
end
```

Setup the Navigation Problem

```
options = sdpsettings('solver','ipopt');
%options = sdpsettings('solver','fmincon','verbose',1);

z = sdpvar(3,N+1);
u = sdpvar(2,N);
constr = [z(:,N+1)==zT, z(:,1) == z0];
cost = 0;
SampleNum = 10;
slack = 10000;
for k = 1:N
    constr = constr+...
         [z(1,k+1) == z(1,k)-u(1,k)*sin(z(3,k))+u(1,k)*sin(z(3,k)+u(2,k)),...
          z(2,k+1) == z(2,k)+u(1,k)*cos(z(3,k))-u(1,k)*cos(z(3,k)+u(2,k)),...
          z(3,k+1) == z(3,k)+u(2,k),...
```

```
        model.u.min(2) <= u(2,k),u(2,k) <= -model.u.min(2),...
        model.z.min <= z(:,k+1),z(:,k+1)<=model.z.max]; %model.u.min(1) <= u(1,k),... don't
 need anymore
    cost = cost + (u(2,k)*u(1,k))^2;
    % Here we introduce a slack variable for the radius of turning which we
    % have seen in ME231A to solve this problem
    cost = cost + slack*(1/(u(1,k)^2)-1/(model.u.min(1)^2));
    for p = 1:SampleNum-1
        for q = 1:size(obs,2)
            zs = z(:,k)+p/SampleNum*(z(:,k+1)-z(:,k));
            A = AA{q}; b = bb{q};
            constr = constr + [(AA{q}*zs(1:2)-bb{q})'*lambda{q}(:,k) >= 0];
            constr = constr + [lambda{q}(:,k)'*AA{q}*AA{q}'*lambda{q}(:,k)<=1];
            constr = constr + [lambda{q}(:,k) >= 0];
        end
    end
end
```

Compute Navigation Solution

```
optimize(constr,cost,options)
zdata = double(z);
udata = double(u);
```

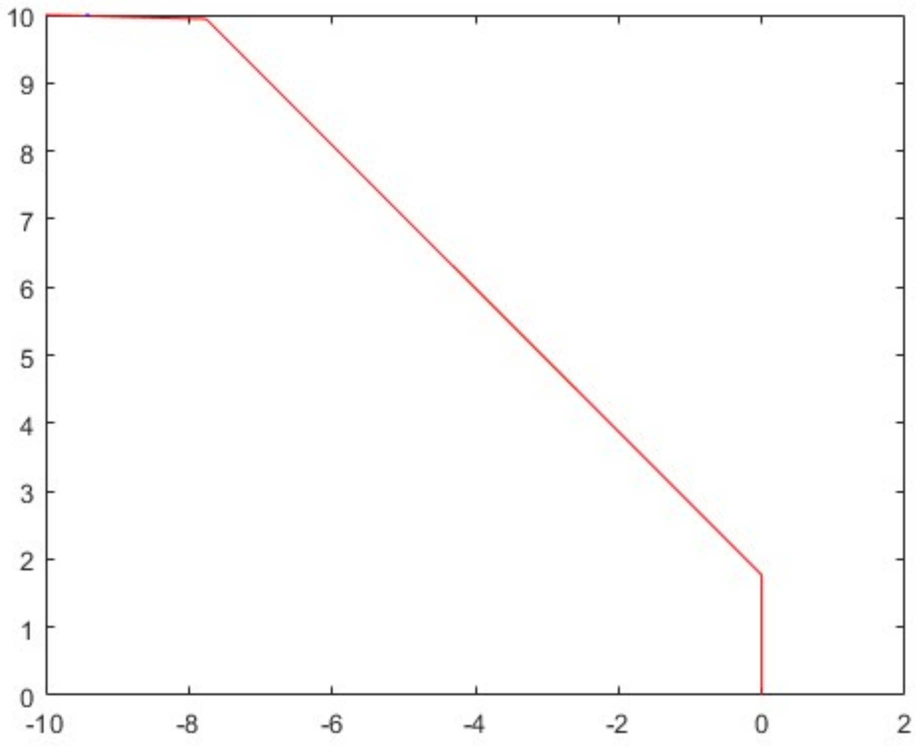
```
******************************************************************************
This program contains Ipopt, a library for large-scale nonlinear optimization.
 Ipopt is released as open source code under the Eclipse Public License (EPL).
         For more information visit http://projects.coin-or.org/Ipopt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

Total number of variables............................:       73
                     variables with only lower bounds:       48
                variables with lower and upper bounds:       21
                     variables with only upper bounds:        0
Total number of equality constraints.................:       20
Total number of inequality constraints...............:      216
        inequality constraints with only lower bounds:        0
   inequality constraints with lower and upper bounds:        0
        inequality constraints with only upper bounds:      216


Number of Iterations....: 295

                                   (scaled)                 (unscaled)
Objective...............:  -3.9680909940123716e+00   -7.9361819880247435e+02
Dual infeasibility......:   9.7253351605915218e-08    1.9450670321183043e-05
Constraint violation....:   7.9291684329518830e-11    7.9291684329518830e-11
Complementarity.........:   1.0000000000000003e-11    2.0000000000000005e-09
Overall NLP error.......:   9.7253351605915218e-08    1.9450670321183043e-05


Number of objective function evaluations             = 554
Number of objective gradient evaluations             = 292
Number of equality constraint evaluations            = 554
```

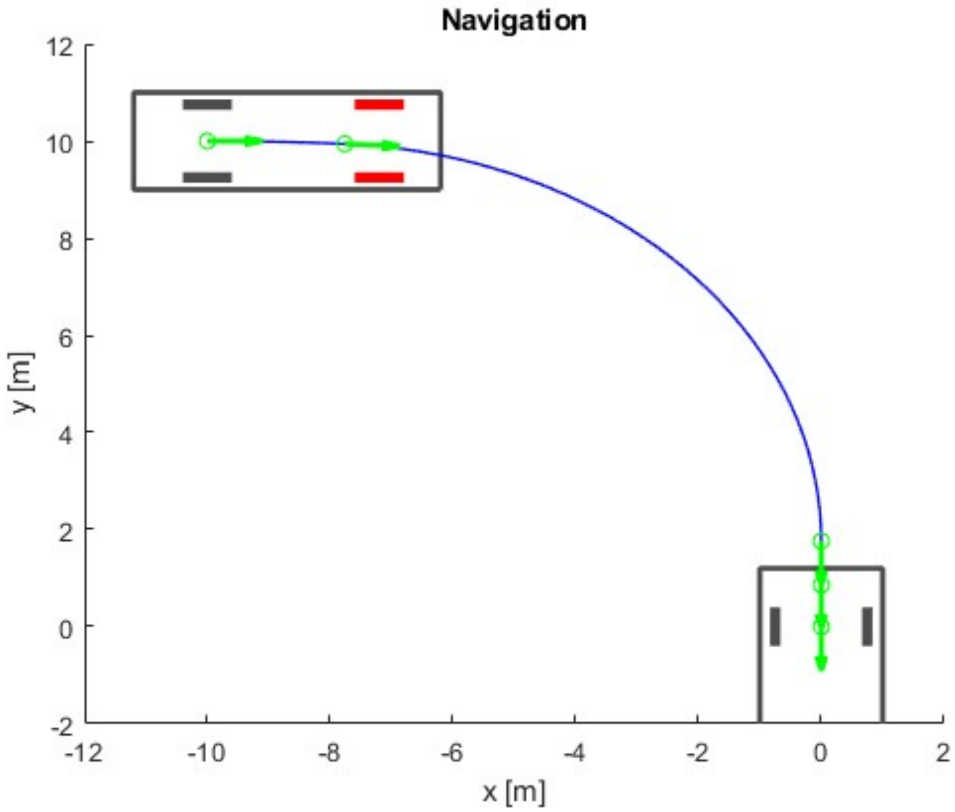
```
Number of inequality constraint evaluations          = 554
Number of equality constraint Jacobian evaluations   = 298
Number of inequality constraint Jacobian evaluations = 298
Number of Lagrangian Hessian evaluations             = 0
Total CPU secs in IPOPT (w/o function evaluations)   =      0.601
Total CPU secs in NLP function evaluations           =      2.631

EXIT: Optimal Solution Found.

ans = 

  struct with fields:

    yalmiptime: 1.9962
    solvertime: 3.2428
          info: 'Successfully solved (IPOPT)'
       problem: 0
```

Plot Solution

```
figure
plot(zdata(1,:),zdata(2,:),'r')
hold on
car_plot(double(u),z0)
```

```
auto = 

  struct with fields:

     w: 2
    db: 1.2000
    df: 1
     l: 5


fig = 

  Figure (8) with properties:

      Number: 8
        Name: ''
       Color: [0.9400 0.9400 0.9400]
    Position: [360 502 560 420]
       Units: 'pixels'

  Use GET to show all properties
```

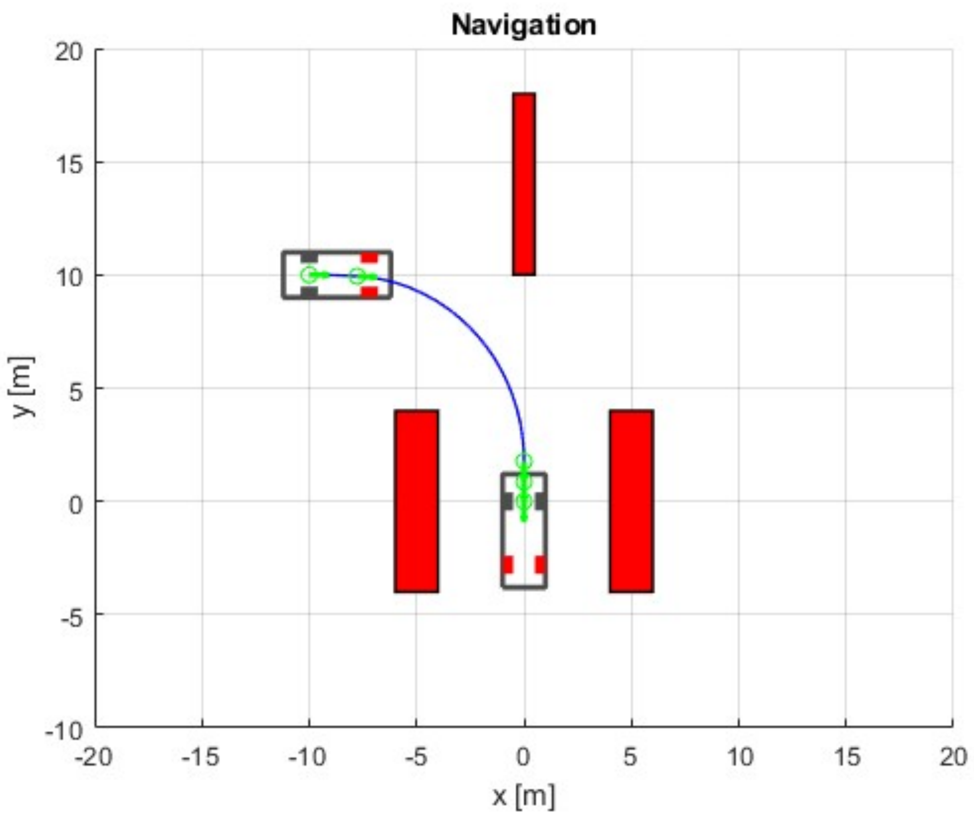




Plot obstacles

```
for j=1:length(obs)
plot(polytope(AA{j},bb{j}));
```

```
end
axis([model.z.min(1) model.z.max(1) model.z.min(2) model.z.max(2)])
```



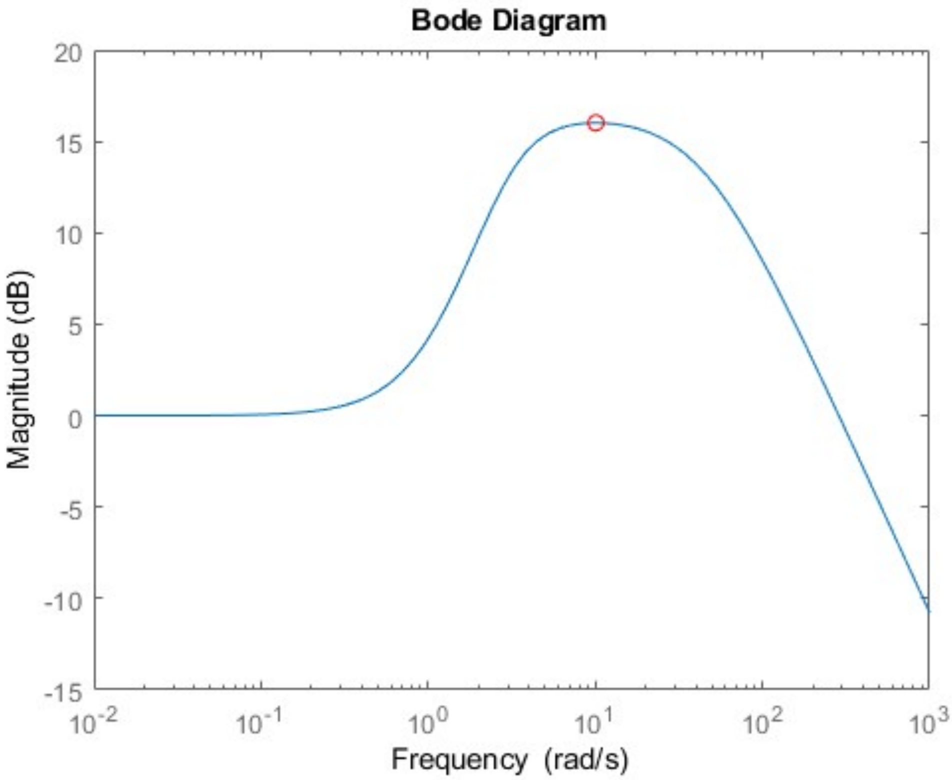
Robust Control

2

At the line 3, we use norm calculator with a setting of accuracy to get the maximum frequency response and its corresponded frequency, thus in the line 4, we have A = abs(freqresp(G,B)). At the line 5, we use frd to compare the system G with a newly defined system, where at the frequency B, the frequency response is exactly the maximum frequency response of G. Finally, we see the superposition of bodegram of two these system at frequnecy B.

```
close all
clear
q2;
```

```
ans =

    6.3223    6.3223
```

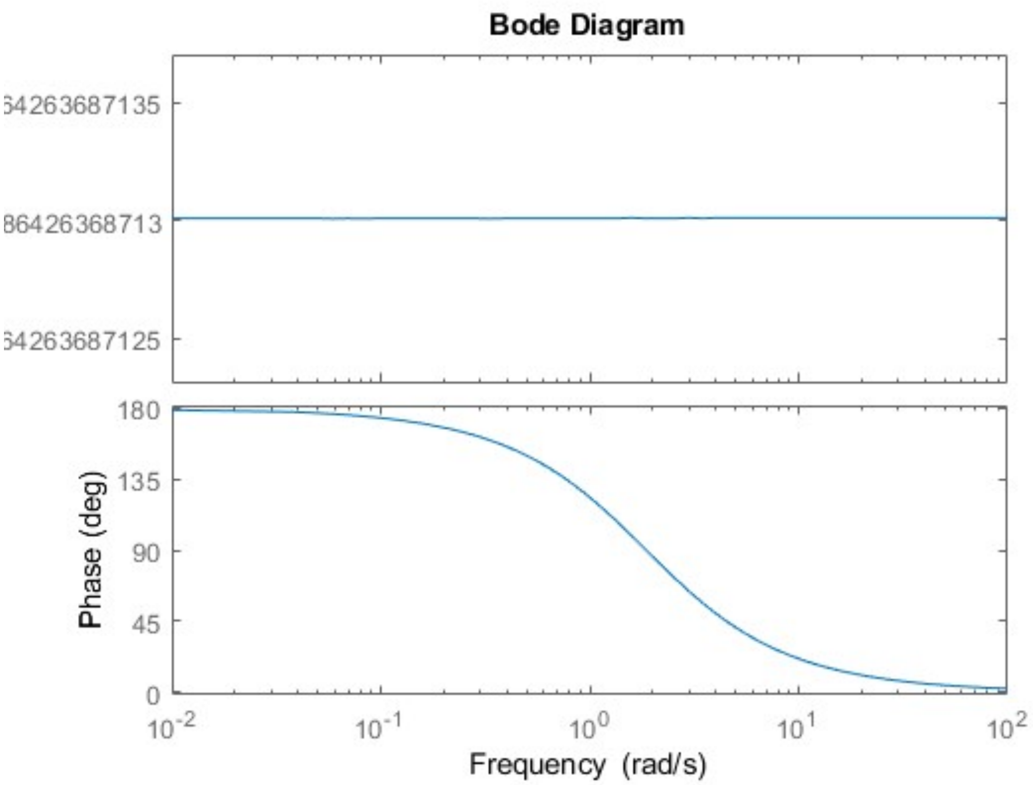


3

At the line 4, we use random complex number delta to generate D. We go to the file cnum2sys.m, as we have seen in the homework and the slide, the behavior of cnumsys.m is that the infinity norm of this system is the same as the complex number generated and the associated frequency is wBar. At the line 5, we calculate the frequency response of D and to compare it with delta (absolutely the same!). At the line 6, we verify that wBar is indeed the frequency. At the line 7, we generate the bode plot of D, we find out the gain is constant and equals to 20*log(delta).

```
q3;
```

```
ans =

   1.4126 + 1.5024i   1.4126 + 1.5024i


ans =

    2.0622    2.0622
```



q.4

4.a

```
P = tf(1,[1,-1]);
C = tf([5.8 9],[0.04 1 0]);
isstable(feedback(P,C))
```

```
ans =

  logical

   1
```

4.b

```
G = -C/(1+P*C);
```

4.c

```
normTol = 0.001;
fprintf('The smallest absolute value of Delta calculated by the small gain thm\n')
[InfNorm,freq] = norm(G,inf,normTol);
bound = 1/InfNorm
%verification: to get the smallest delta
M = freqresp(G,freq);
[U,S,V]=svd(M);
```

```
deltamin = -1/S(1,1)*V(:,1)*U(:,1)';
pole(feedback(P-deltamin,C))
%one pure imaginary pool appear at this critical value
```

```
The smallest absolute value of Delta calculated by the small gain thm

bound =

    0.1454


ans =

  -1.3912 -13.6340i
  -0.0000 + 7.8064i
  -2.3928 - 0.1569i
```

4.d

We see that it can tolerate up to 100% of the modeled uncertainty, the results found here is eactly as we have seen in 4.c

```
deltamin = ucomplex('delta',0,'Radius',0.1454);
%we use the smallest value found in 4.c
[stabmarg,destabunc,report] = robuststab(feedback(P+deltamin,C))
```

```
stabmarg =

  struct with fields:

              LowerBound: 1.0000
              UpperBound: 1.0000
    DestabilizingFrequency: 7.7805


destabunc =

  struct with fields:

    delta: -0.1393 + 0.0418i


report =

  6×87 char array

    'System is robustly stable for the modeled uncertainty.                                 '
    ' -- It can tolerate up to 100% of the modeled uncertainty.                             '
    ' -- There is a destabilizing perturbation amounting to 100% of the modeled uncertainty.'
    ' -- This perturbation causes an instability at the frequency 7.78 rad/seconds.        '
    ' -- Sensitivity with respect to each uncertain element is:                             '
```

```
'      100% for delta. Increasing delta by 25% decreases the margin by 25%.        '
```

4.e

```
deltamin = -1/S(1,1)*V(:,1)*U(:,1)';
deltanew = cnum2sys(deltamin,freq);
pole(feedback(P+deltanew, C))
%We can see clearly that there are two poles on the imaginary
% axis, thus the dynamic system generated in 4.e is unstable.
```

```
ans =

 -26.8942 +40.5961i
 -26.8942 -40.5961i
  -1.5000 + 1.4554i
  -1.5000 - 1.4554i
```

4.f

```
deltanew = ultidyn('delta', [1 1], 'Bound', norm(deltamin));
[stabmarg,destabunc,report] = robuststab(feedback(P+deltanew,C))
```

```
stabmarg =

  struct with fields:

               LowerBound: 1.0000
               UpperBound: 1.0000
    DestabilizingFrequency: 7.8176


destabunc =

  struct with fields:

    delta: [1×1 ss]


report =

  6×87 char array

    'System is robustly stable for the modeled uncertainty.                                 '
    ' -- It can tolerate up to 100% of the modeled uncertainty.                             '
    ' -- There is a destabilizing perturbation amounting to 100% of the modeled uncertainty.'
    ' -- This perturbation causes an instability at the frequency 7.82 rad/seconds.         '
    ' -- Sensitivity with respect to each uncertain element is:                             '
    '      100% for delta. Increasing delta by 25% decreases the margin by 25%.             '
```

4.g

G is the sensitivity function of the negative feedback system of P,C

```
P = tf(1,[1 -1]);
C = tf([5.8 9],[0.04 1 0]);
G_new = -P*C/(1+P*C);
```

4.h

```
[Inf_norm, freq] = norm(G_new, inf, normTol);
M = freqresp(G_new, freq);
[U,S,V] = svd(M);
deltamin = -1/S(1,1)*V(:,1)*U(:,1)';
fprintf('norm of delta\n')
disp(norm(deltamin))
fprintf('norm of delta calculated by small gain thm\n')
disp(1/Inf_norm)
P_tilde = P*(1-deltamin);
pole(feedback(P_tilde, C))
```

```
norm of delta
    0.6802

norm of delta calculated by small gain thm
    0.6802


ans =

 -22.9601 - 1.7435i
  -0.0000 + 2.9391i
  -1.0399 - 1.1956i
```

4.i

```
deltanew = ultidyn('delta', [1 1], 'Bound', norm(deltamin));
uncertainSys = feedback((P*(1-deltanew)),C);
[stabmarg,destabunc,report] = robuststab(uncertainSys)
% By refering to the output in stabmarg and destabunc, the
% system is marginally unstable, which meets the result found in 4.h
```

```
stabmarg =

  struct with fields:

              LowerBound: 1.0000
              UpperBound: 1.0000
    DestabilizingFrequency: 2.9326
```

```
destabunc = 

  struct with fields:

    delta: [1×1 ss]


report = 

  6×87 char array

    'System is robustly stable for the modeled uncertainty.                                 '
    ' -- It can tolerate up to 100% of the modeled uncertainty.                             '
    ' -- There is a destabilizing perturbation amounting to 100% of the modeled uncertainty.'
    ' -- This perturbation causes an instability at the frequency 2.93 rad/seconds.         '
    ' -- Sensitivity with respect to each uncertain element is:                             '
    '      100% for delta. Increasing delta by 25% decreases the margin by 25%.             '
```

q5

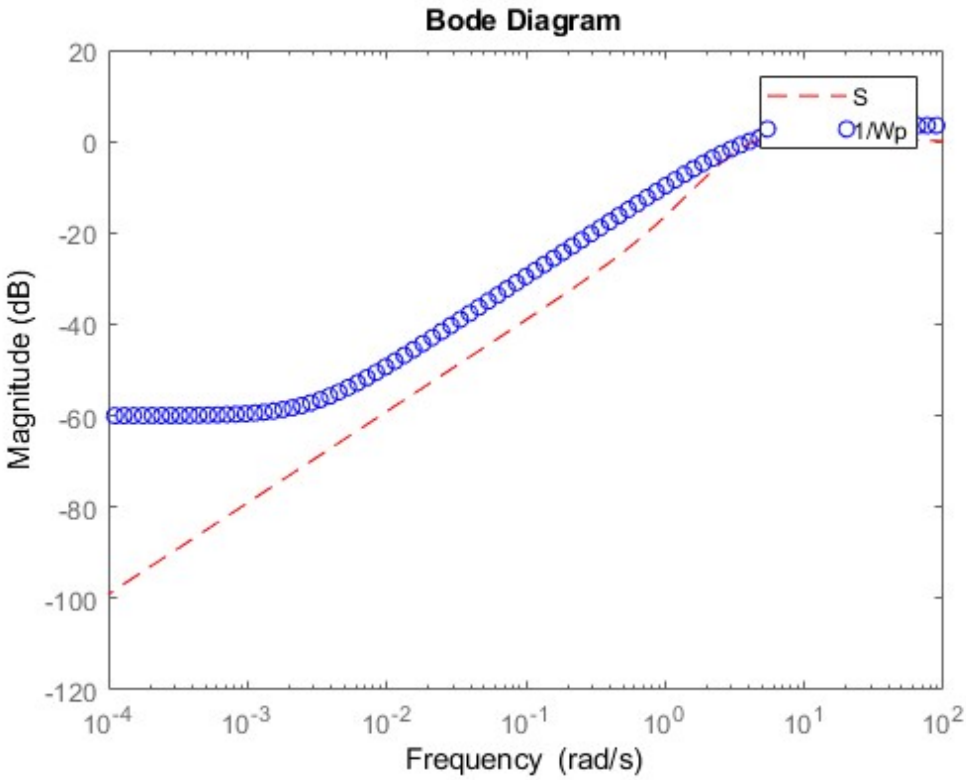
5.a

```
P = tf(1,[1 -1]);
C = tf([5.8 9],[0.04 1 0]);
S = feedback(1,P*C);
Wp = tf([0.667 3],[1 0.003]);
normTol = 0.001;
norm(Wp*S,inf,normTol)<=1
```

```
ans = 

  logical

   1
```

5.b

```
figure
bodemag(S,'r--',1/Wp,'bo')
legend('S','1/Wp')
```



5.c

Based on the results of 4.g, the system is stable, moreover we can verify it by using the function robstab.

```
delta = ultidyn('delta',[1 1],'bound',1);
Pu = P*(1+0.4*delta);
System1 = feedback(C,Pu);
[stabmarg,wcu,report] = robuststab(System1)
```

```
stabmarg = 

  struct with fields:

               LowerBound: 1.7019
               UpperBound: 1.7019
    DestabilizingFrequency: 2.8380


wcu = 

  struct with fields:

    delta: [1×1 ss]


report = 

  6×87 char array
```

```
'System is robustly stable for the modeled uncertainty.                                '
' -- It can tolerate up to 170% of the modeled uncertainty.                            '
' -- There is a destabilizing perturbation amounting to 170% of the modeled uncertainty.'
' -- This perturbation causes an instability at the frequency 2.84 rad/seconds.        '
' -- Sensitivity with respect to each uncertain element is:                            '
'      100% for delta. Increasing delta by 25% decreases the margin by 25%.            '
```

5.d

```
[wcg,wcu] = wcgain(Wp/(1+Pu*C))
% specific stable linear system
wcu.delta
```

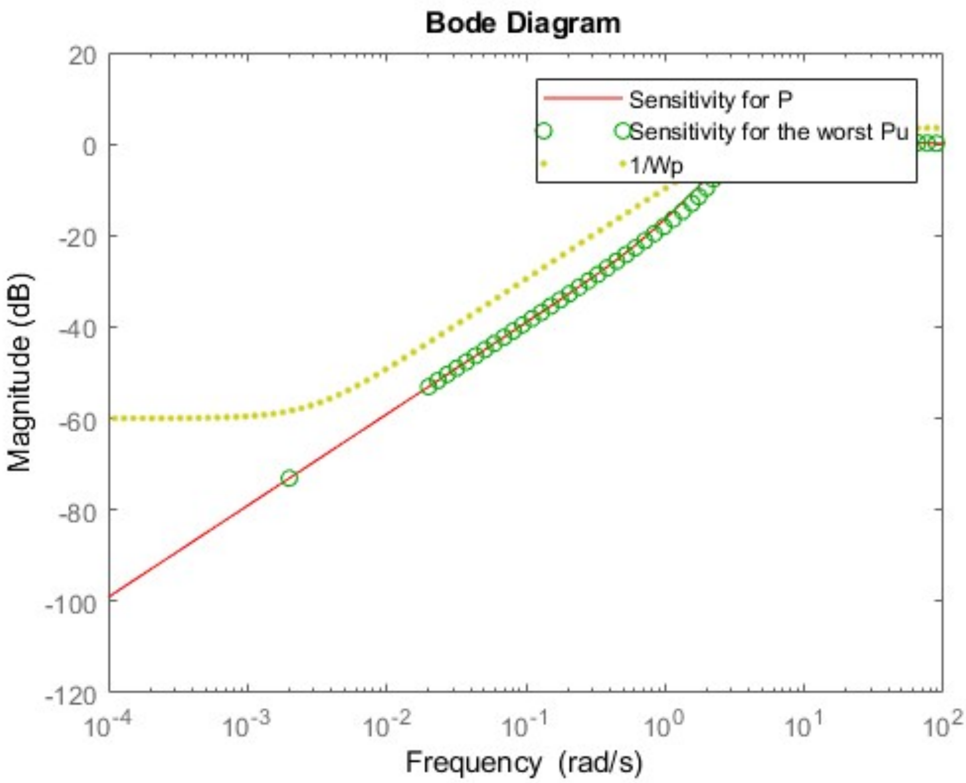
```
wcg =

  struct with fields:

           LowerBound: 2.1645
           UpperBound: 2.1686
    CriticalFrequency: 3.4966


wcu =

  struct with fields:

    delta: [1×1 ss]


ans =

  A =
             x1      x2      x3      x4
   x1    -4.704  -9.407       0    6.82
   x2         0  -4.704       0    6.82
   x3         0       0  -2.472   2.472
   x4         0       0  -2.472  -2.472

  B =
            u1
   x1        0
   x2        0
   x3    2.224
   x4    2.224

  C =
             x1      x2      x3      x4
   y1    -3.067  -3.067       0   2.224

  D =
        u1
   y1    0
```

```
Continuous-time state-space model.
```

5.e

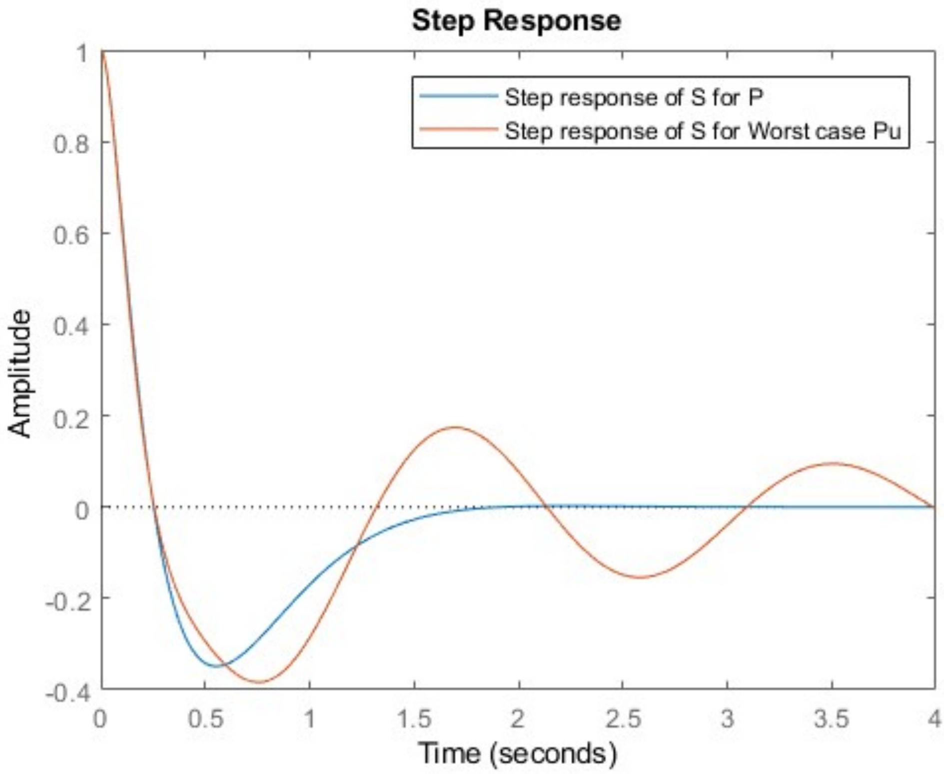
We can see the worst case gain from the intersection of 1/Wp and the sensitivity of the worse case Pu (refer to the small gain theorem), the intersection frequency is larger than 1, which confirm the worse case gain seen in 5.d.

```
Pu_worst = P*(1+0.4*wcu.delta);
Sworst = 1/(1+Pu_worst*C);
figure
bodemag(S,'r-',Sworst,'go',1/Wp,'y.')
legend('Sensitivity for P','Sensitivity for the worst Pu','1/Wp')
```



5.f

```
figure
step(S,Sworst,4)
legend('Step response of S for P','Step response of S for Worst case Pu')
```



5.g Task 1

```
Wu = makeweight(0.4, 20, 400);
delta = ultidyn('delta',[1 1],'bound',1);
PuNew = P*(1+Wu*delta);
robuststab(feedback(PuNew,C))
```

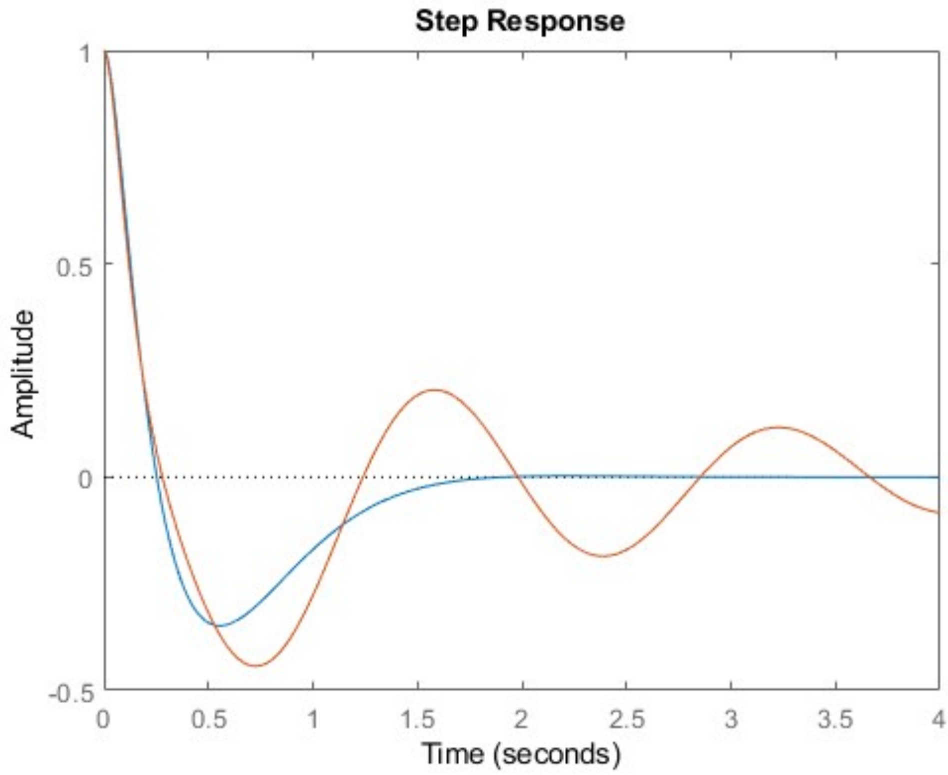
5.g Task 2

```
[wcg,wcu] = wcgain(Wp/(1+PuNew*C))
```

```
wcg =

  struct with fields:

           LowerBound: 2.4469
           UpperBound: 2.4520
    CriticalFrequency: 3.8329


wcu =

  struct with fields:

    delta: [1×1 ss]
```

5.g Task 3

```
figure
Pu_worst = P*(1+Wu*wcu.delta);
step(1/(1+P*C),1/(1+Pu_worst*C),4)
```



Published with MATLAB® R2017b

```
validation code 1:

A =

       39386

validation code 2:

ans =

    73764561
```