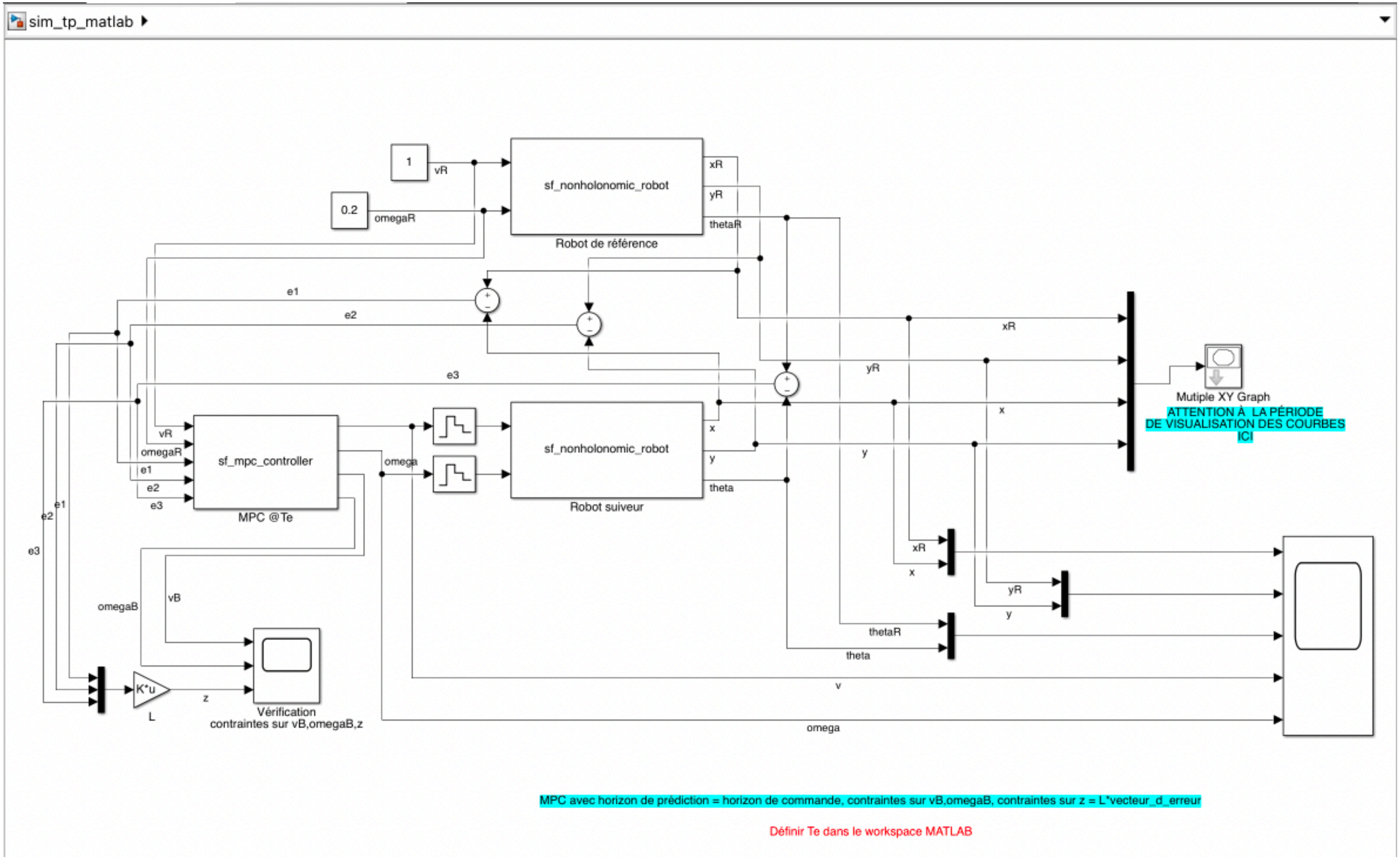
sim_tp_matlab
1
vR
0.2
omegaR
sf_nonholonomic_robot
xR
yR
thetaR
Robot de référence
e1
e2
e3
vR
omegaR
e1
e2
e3
sf_mpc_controller
MPC @Te
omega
sf_nonholonomic_robot
x
y
theta
Robot suiveur
xR
yR
x
y
Mutiple XY Graph
ATTENTION À LA PÉRIODE
DE VISUALISATION DES COURBES
ICI
xR
x
yR
y
thetaR
theta
v
omega
omegaB
vB
K*u
L
z
Vérification
contraintes sur vB,omegaB,z
MPC avec horizon de prédiction = horizon de commande, contraintes sur vB,omegaB, contraintes sur z = L*vecteur_d_erreur
Définir Te dans le workspace MATLAB

```matlab
clear all
close all

A_REMPLIR = nan;

vR = 1, omegaR = 0.2, Te = 0.25

Ac = [0 omegaR 0 ; -omegaR 0 vR ; 0 0 0]; Bc = [-1 0 ; 0 0 ; 0 -1];
Ad = ...
[  cos(Te*omegaR), sin(Te*omegaR), (vR - vR*cos(Te*omegaR))/omegaR;
  -sin(Te*omegaR), cos(Te*omegaR),     (vR*sin(Te*omegaR))/omegaR;
              0,             0,                               1];
Bd = ...
[          -sin(Te*omegaR)/omegaR, (vR*(sin(Te*omegaR) - Te*omegaR))/omegaR^2;
   (2*sin((Te*omegaR)/2)^2)/omegaR,    -(2*vR*sin((Te*omegaR)/2)^2)/omegaR^2;
                              0,                                    -Te];
Gc = ss(Ac,Bc,eye(3),0); clear Ac Bc
B0Gd = ss(Ad,Bd,eye(3),0,Te); B0Gd_bis = c2d(Gc,Te,'zoh'); clear Ad Bd

%% Section IV - DLQ Riccati
fprintf('= Section IV - DLQ Riccati =====================================\n');
N = 100; % Attention, horizon de taille N => indices de 1 à N+1 !
nb_cas = 3;
S = {100*eye(3),100*eye(3),diag([1 1 100])};
Q = {100*eye(3),1*eye(3),diag([100 100 1])};
R = {1*eye(2),100*eye(2),diag([1 10])};
TIME = [0:N]*Te;

%
% Résolution de l'équation récurrente rétrograde de Riccati
P = {nan(3,3,N+1),nan(3,3,N+1),nan(3,3,N+1)};
for i_cas = 1:nb_cas
    P{i_cas}(:,:,N+1) = A_REMPLIR; % remplissage de P[N]
    for i_time = N+1:-1:2
        % depuis les instants k=N jusqu'à k=1 afin de remplir P[N-1] à P[0]
        P{i_cas}(:,:,i_time-1) = ...
            A_REMPLIR;
    end
end
%
% Gain de retour d'état u[k] = -K[k] x[k] depuis k=N-1 jusqu'à k=0
K = {nan(2,3,N+1),nan(2,3,N+1),nan(2,3,N+1)}; % dernière page = NaN's
for i_cas = 1:nb_cas
    for i_time = 1:N
        K{i_cas}(:,:,i_time) = A_REMPLIR;
    end
end
%
% Simulation des états depuis k=0 jusqu'à k=N
U = {nan(2,1,N+1),nan(2,1,N+1),nan(2,1,N+1)};
X = {nan(3,1,N+1),nan(3,1,N+1),nan(3,1,N+1)};
for i_cas = 1:nb_cas
    X{i_cas}(:,:,1) = A_REMPLIR;
    for i_time = 1:N
        U{i_cas}(:,:,i_time) = ...
            A_REMPLIR;
        X{i_cas}(:,:,i_time+1) = ...
            A_REMPLIR;
    end
end
%
% Plot
for i_cas = 1:nb_cas
    plotX{i_cas}=squeeze(X{i_cas});
    plotU{i_cas}=squeeze(U{i_cas});
    figure(i_cas); title(sprintf('== DLQR -- Cas %d ==',i_cas));
    subplot(2,1,1); title('x');
    plot(TIME,plotX{i_cas}(1,:),'o-',...
         TIME,plotX{i_cas}(2,:),'o-.',...
         TIME,plotX{i_cas}(3,:),'o:');
    legend('x1','x2','x3');
    subplot(2,1,2); title('u');
    plot(TIME,plotU{i_cas}(1,:),'o-',...
         TIME,plotU{i_cas}(2,:),'o-.');
    legend('u1','u2');
end

%% Section VI - DLQR
fprintf('= Section VI - DLQR =====================================\n');
for i_cas = 1:nb_cas
    fprintf('Cas %d\n',i_cas);
    fprintf('--> solutions DLQR\n');
    P_DLQR{i_cas} = A_REMPLIR;
    K_DLQR{i_cas} = A_REMPLIR;
    P_DLQR{i_cas}, K_DLQR{i_cas}
    fprintf('--> "régimes permanents" P et K du problème DLQ\n');
    A_REMPLIR;
end

%% Section V - DLQ par Programmation Quadratique
fprintf('= Section V - DLQ par Programmation Quadratique ==============================
========\n');
for i_cas = 1:nb_cas
    barQ{i_cas} = A_REMPLIR;
    barR{i_cas} = A_REMPLIR;
    %
    barS = A_REMPLIR; barT = A_REMPLIR;
    for i_barST = 1:N
        barS = A_REMPLIR;
        barT = A_REMPLIR;
    end
    H = A_REMPLIR;
    f = A_REMPLIR;
    Y = A_REMPLIR;
    %
    xiU{i_cas} = A_REMPLIR;
    clear barS barT H f Y
    fprintf('Cas %d\n',i_cas);
    fprintf('--> u_B^*[0:N-1] solution DLQ Riccati\n');
    plotU{i_cas}(:,1:(end-1))
    fprintf('--> u_B^*[0:N-1] solution Prog Quad\n');
    reshape(xiU{i_cas},2,N)
    fprintf('--> Écart absolu maxi\n');
    max(max(abs(plotU{i_cas}(:,1:(end-1))-reshape(xiU{i_cas},2,N))))
end

%% Section VII - DLQ avec contraintes par Programmation Quadratique
fprintf('= Section V - DLQ avec contraintes par Programmation Quadratique ============
========================\n');
i_cas = 1; % On choisit le Cas 1
%
% Critère 1/2 xi^T H xi où xi = [E^T U^T]^T
% avec E = [e[1];...;e[N]] et U = [uB[1];...;uB[N-1]]
H = A_REMPLIR;
```

```
f = A_REMPLIR;
%
% Contraintes égalités (indépendantes du cas considéré)
Ae = A_REMPLIR;
Be = A_REMPLIR;
Ae1 = A_REMPLIR;
Ae2 = A_REMPLIR;
Ae = [Ae1 Ae2];
clear Ae1 Ae2
Be(1:3,:) = A_REMPLIR;
%
% Contraintes inégalités -- on considèrera nb_cas_ineg cas différents
nb_cas_ineg = 1; % on peut aller jusqu'à 3 par exemple
xiEU = cell(nb_cas_ineg);
fval = cell(nb_cas_ineg);
exitflag = cell(nb_cas_ineg);
Ai = cell(nb_cas_ineg);
Bi = cell(nb_cas_ineg);
uMIN = cell(nb_cas_ineg);
uMAX = cell(nb_cas_ineg);
L = cell(nb_cas_ineg); % sorties critiques z = L e
zMIN = cell(nb_cas_ineg);
zMAX = cell(nb_cas_ineg);
uMIN{1} = [-3;-0.5]; uMAX{1} = [3;2];
zMIN{1} = -0.7; zMAX{1} = pi/3; L{1} = [0 0 1]; %contrainte sur e3
for i_cas_ineg = 1:nb_cas_ineg
    Ai{i_cas_ineg} = zeros(2*N+4*N,3*N+2*N);
    Bi{i_cas_ineg} = zeros(2*N+4*N,1);
    Ai{i_cas_ineg} = A_REMPLIR;
    Bi{i_cas_ineg} = A_REMPLIR;
    [xiEU{i_cas_ineg},fval{i_cas_ineg},exitflag{i_cas_ineg}] = ...
        quadprog(H,f,Ai{i_cas_ineg},Bi{i_cas_ineg},Ae,Be);
    if (exitflag{i_cas_ineg} ~= 1), error('Revoir programme optim'); end
    %
    U_in_xiEU = xiEU{i_cas_ineg}((3*N+1):(5*N),:);
    plotUcontraint{i_cas_ineg} = reshape(U_in_xiEU,2,N); clear U_in_xiEU
    plotUcontraint{i_cas_ineg} = [plotUcontraint{i_cas_ineg} nan(2,1)];
    %
    X_in_xiEU = xiEU{i_cas_ineg}(1:(3*N),:);
    plotXcontraint{i_cas_ineg} = reshape(X_in_xiEU,3,N); clear X_in_xiEU
    plotXcontraint{i_cas_ineg} = [X{i_cas}(:,:,1) plotXcontraint{i_cas_ineg}];
end
%
% Plot
for i_cas_ineg = 1:nb_cas_ineg
    figure(10+i_cas_ineg);
    title(sprintf('== DLQR CONTRAINT -- Cas %d -- Cas\\_ineg %d ==',i_cas,i_cas_ineg))
;
    subplot(2,1,1); title('x');
    plot(TIME,plotXcontraint{i_cas}(1,:),'o-',...
         TIME,plotXcontraint{i_cas}(2,:),'o-.',...
         TIME,plotXcontraint{i_cas}(3,:),'o:');
    legend('x1','x2','x3');
    subplot(2,1,2); title('u');
    plot(TIME,plotUcontraint{i_cas}(1,:),'o-',...
         TIME,plotUcontraint{i_cas}(2,:),'o-.');
    legend('u1','u2');
end
```

```
function sf_nonholonomic_robot(s)
setup(s);

function setup(s)
s.NumDialogPrms = 1;

s.NumInputPorts  = 2;
s.NumOutputPorts = 3;

s.SetPreCompInpPortInfoToDynamic;
s.SetPreCompOutPortInfoToDynamic;

s.InputPort(1).Dimensions = 1;
s.InputPort(1).DirectFeedthrough = false;
s.InputPort(2).Dimensions = 1;
s.InputPort(2).DirectFeedthrough = false;

s.OutputPort(1).Dimensions = 1;
s.OutputPort(2).Dimensions = 1;
s.OutputPort(3).Dimensions = 1;

s.SampleTimes = [0 0];

s.NumContStates = 3;

s.SimStateCompliance = 'DefaultSimState';

s.RegBlockMethod('SetInputPortSamplingMode', @SetInpPortFrameData);

s.RegBlockMethod('InitializeConditions', @InitializeConditions);
s.RegBlockMethod('Outputs', @Outputs);     % Required
s.RegBlockMethod('Derivatives', @Derivatives);

function SetInpPortFrameData(block, idx, fd)
  block.InputPort(idx).SamplingMode = fd;
  block.OutputPort(1).SamplingMode  = fd;
  block.OutputPort(2).SamplingMode  = fd;
  block.OutputPort(3).SamplingMode  = fd;

function InitializeConditions(s)
s.ContStates.Data = s.DialogPrm(1).Data;

function Outputs(s)
s.OutputPort(1).Data = s.ContStates.Data(1);
s.OutputPort(2).Data = s.ContStates.Data(2);
s.OutputPort(3).Data = s.ContStates.Data(3);

function Derivatives(s)
v = s.InputPort(1).Data;
omega = s.InputPort(2).Data;
x = s.ContStates.Data(1);
y = s.ContStates.Data(2);
theta = s.ContStates.Data(3);
s.Derivatives.Data = [v*cos(theta);v*sin(theta);omega];
```

```matlab
function sf_mpc_controller(s)
setup(s);

function setup(s)
s.NumDialogPrms = 4;

s.NumInputPorts  = 5;
s.NumOutputPorts = 4;

s.SetPreCompInpPortInfoToDynamic;
s.SetPreCompOutPortInfoToDynamic;

s.InputPort(1).Dimensions = 1;
s.InputPort(1).DirectFeedthrough = true;
s.InputPort(2).Dimensions = 1;
s.InputPort(2).DirectFeedthrough = true;
s.InputPort(3).Dimensions = 1;
s.InputPort(3).DirectFeedthrough = true;
s.InputPort(4).Dimensions = 1;
s.InputPort(4).DirectFeedthrough = true;
s.InputPort(5).Dimensions = 1;
s.InputPort(5).DirectFeedthrough = true;

s.OutputPort(1).Dimensions = 1;
s.OutputPort(2).Dimensions = 1;
s.OutputPort(3).Dimensions = 1;
s.OutputPort(4).Dimensions = 1;

s.SampleTimes = [s.DialogPrm(1).Data 0];
s.SimStateCompliance = 'DefaultSimState';
s.RegBlockMethod('SetInputPortSamplingMode', @SetInpPortFrameData);
s.RegBlockMethod('Outputs', @Outputs);

function SetInpPortFrameData(block, idx, fd)
  block.InputPort(idx).SamplingMode = fd;
  block.OutputPort(1).SamplingMode  = fd;
  block.OutputPort(2).SamplingMode  = fd;
  block.OutputPort(3).SamplingMode  = fd;
  block.OutputPort(4).SamplingMode  = fd;

function Outputs(s)
Te = s.DialogPrm(1).Data;
Np = s.DialogPrm(2).Data; Nu = s.DialogPrm(3).Data;
L = s.DialogPrm(4).Data;

vR = s.InputPort(1).Data; omegaR = s.InputPort(2).Data;
e0 = [s.InputPort(3).Data;s.InputPort(4).Data;s.InputPort(5).Data];
Ad = ...
[  cos(Te*omegaR), sin(Te*omegaR), (vR - vR*cos(Te*omegaR))/omegaR;
  -sin(Te*omegaR), cos(Te*omegaR),      (vR*sin(Te*omegaR))/omegaR;
              0,             0,                            1];
Bd = ...
[          -sin(Te*omegaR)/omegaR, (vR*(sin(Te*omegaR) - Te*omegaR))/omegaR^2;
   (2*sin((Te*omegaR)/2)^2)/omegaR,    -(2*vR*sin((Te*omegaR)/2)^2)/omegaR^2;
                             0,                                   -Te];

Q = {100*eye(3),1*eye(3),diag([100 100 1]),eye(3)};
R = {1*eye(2),100*eye(2),diag([1 10]),eye(2)};

i_cas = 1;
% Utiliser Q{i_cas}, R{i_cas}
A_COMPLETER = nan;

S{i_cas} = A_COMPLETER;

% ...

% Critère 1/2 xi^T H xi où xi = [E^T U^T]^T
% avec E = [e[1];...;e[N]] et U = [uB[1];...;uB[N-1]]
H = A_COMPLETER;
f = A_COMPLETER;

%
% Contraintes égalités
Ae = A_COMPLETER;
Be = A_COMPLETER;

%
% Contraintes inégalités
uMIN = [-10;-10]; uMAX = [10;10];
zMIN = -pi; zMAX = pi;

Ai = A_COMPLETER;
Bi = A_COMPLETER;

%
% ENSUITE DECOMMENTER : [xiEU,fval,exitflag] = quadprog(H,f,Ai,Bi,Ae,Be);
% if (exitflag ~= 1), exitflag, error('Revoir programme optim'); end

uB = [vR;omegaR]; %MODIFIER BIEN SUR...
uF = [0;0]; %A_COMPLETER;

s.OutputPort(1).Data = uB(1)+uF(1);
s.OutputPort(2).Data = uB(2)+uF(2);
s.OutputPort(3).Data = uB(1);
s.OutputPort(4).Data = uB(2);
```