Commande articulaire d'un robot manipulateur de type RP

Table of Contents

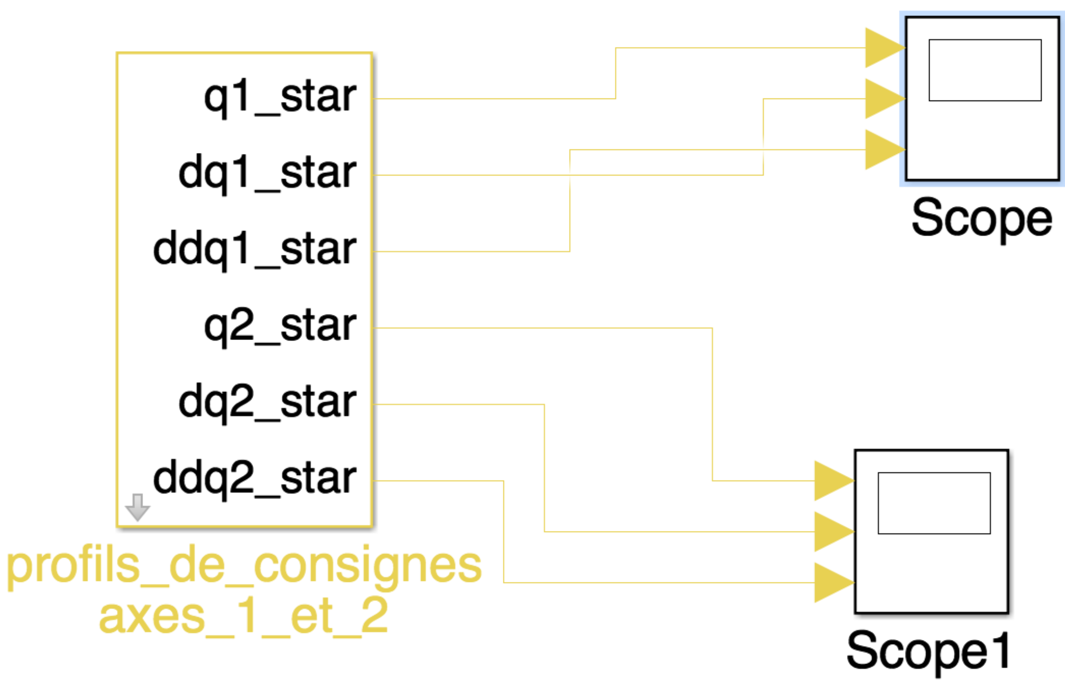
TP réalisé par Benjamin CHAMAND

Cette manipulation concerne l'étude simulée, sous Matlab, de diverses stratégies pour la commande articulaire en position d'un bras manipulateur élémentaire de type RP.

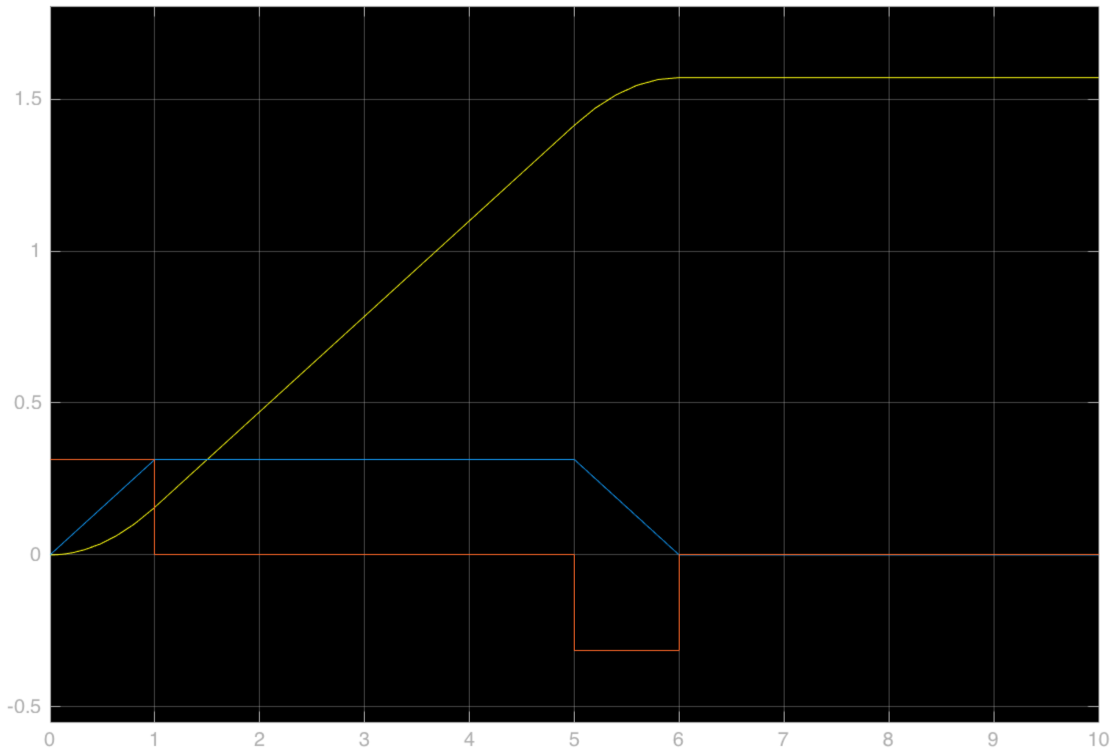
Prise en main de l'outil de simulation et calculs préliminaires

Question 1 : évolution temporelle des profils de consigne.

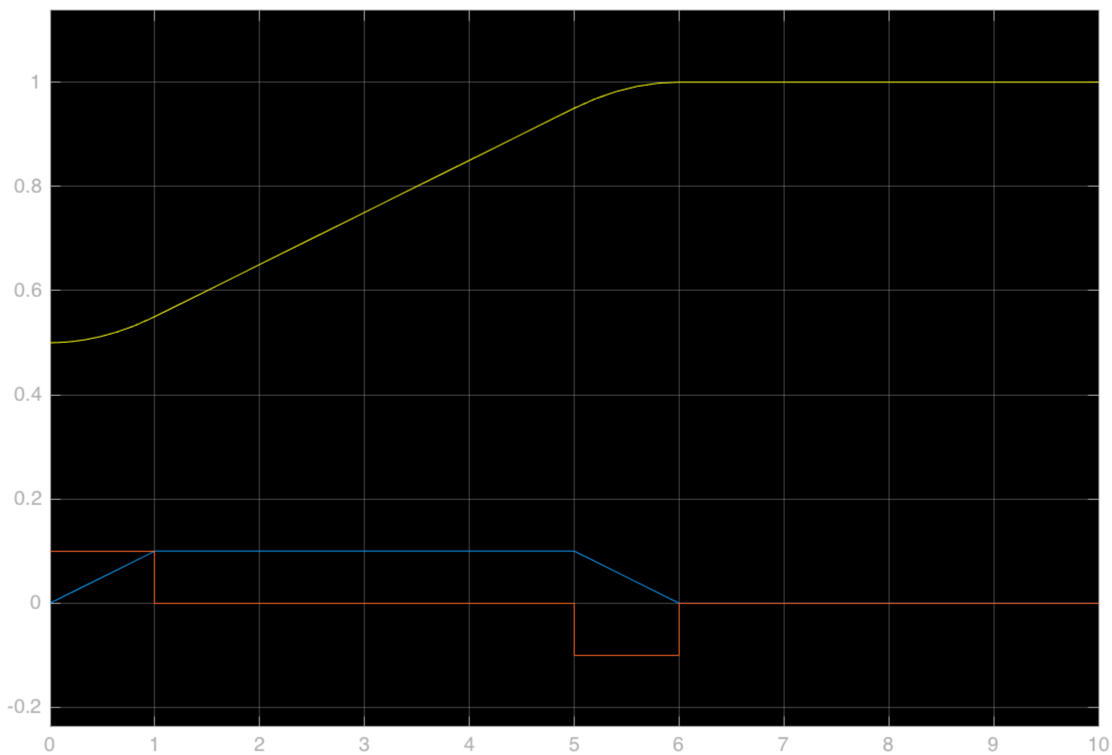
En simulant le bloc "profils_de_consigne_axes_1_et_2" avec 2 oscilloscopes à la sortie du bloc on voit le profil de position, de vitesse et d'accélération de q1 et q2.



Scope 1: montre l'évolution temporelle de la position, vitesse et accélération de q1



Scope 2: montre l'évolution temporelle de la position, vitesse et accélération de q2



Question 2 : Mise en place des constantes

```
% Rapport du gain en vitesse du moteur et de la résistance de l'induit
Km_R = 0.3;

% Coefficient de frottement efficace de l'ensemble moteur-réducteur
Beff = 1/80;

% Inertie ensemble moteur-réducteur
Jm = 1/100;

% Rapports de réduction (3 cas différents)
R = [1/200 1/30 1];
% On considère un cas du rapport de réduction parmi les 3 précédents
r = R(1);

% masse m = 15kg
m = 15;

% Energie efficace
Jeff = Jm + r^2*m;

% Fonction de transfert de la partie mécanique du robot
G = tf(1,[Jeff Beff 0]);
```

Commande en vitesse de type PD

Exercice 3 : On doit établir l'expression analytique des coefficients K et K_D de la loi de commande proportionnelle dérivée.

$$\epsilon_{vit} = \frac{(B_{eff} + K_d \cdot \frac{K_m}{R})\dot{\theta}_1 + r \cdot d0}{K \frac{K_m}{R}}$$

On a $d_i(t) \equiv 0$, par suite :

$$\epsilon_{vit} = \frac{(B_{eff} + K_d \cdot \frac{K_m}{R})\dot{\theta}_1}{K \frac{K_m}{R}}$$

De plus, avec $\zeta = 1$, on a :

$$w_n = \sqrt{\frac{K \cdot \frac{K_m}{R}}{J}}$$

$$2 \cdot \zeta \cdot w_n = \frac{B_{eff} + K_d \cdot \frac{K_m}{R}}{J} \Rightarrow 2 \cdot w_n = \frac{B_{eff} + K_d \cdot \frac{K_m}{R}}{J}$$

En égalisant les 2 dernières équations, on trouve :

$$K = \frac{B_{eff} + k_d \cdot \frac{K_m}{R}}{4 \cdot J_{eff} \cdot \frac{K_m}{R}}$$

Question 4 :

(a) : calcul des valeurs numériques des coefficients K et K_d de telle sorte que l'erreur en vitesse sur la consigne rampe = 1/2.

Avec toutes les équations précédentes, on a :

$$K = \frac{2 \cdot (B_{eff} + K_d \cdot \frac{K_m}{R}}{K \cdot \frac{K_m}{R}} = \frac{B_{eff} + k_d \cdot \frac{K_m}{R}}{4 \cdot J_{eff} \cdot \frac{K_m}{R}}$$

$$\Rightarrow K_d = \frac{8 \cdot J_{eff} - B_{eff}}{\frac{K_m}{R}}$$

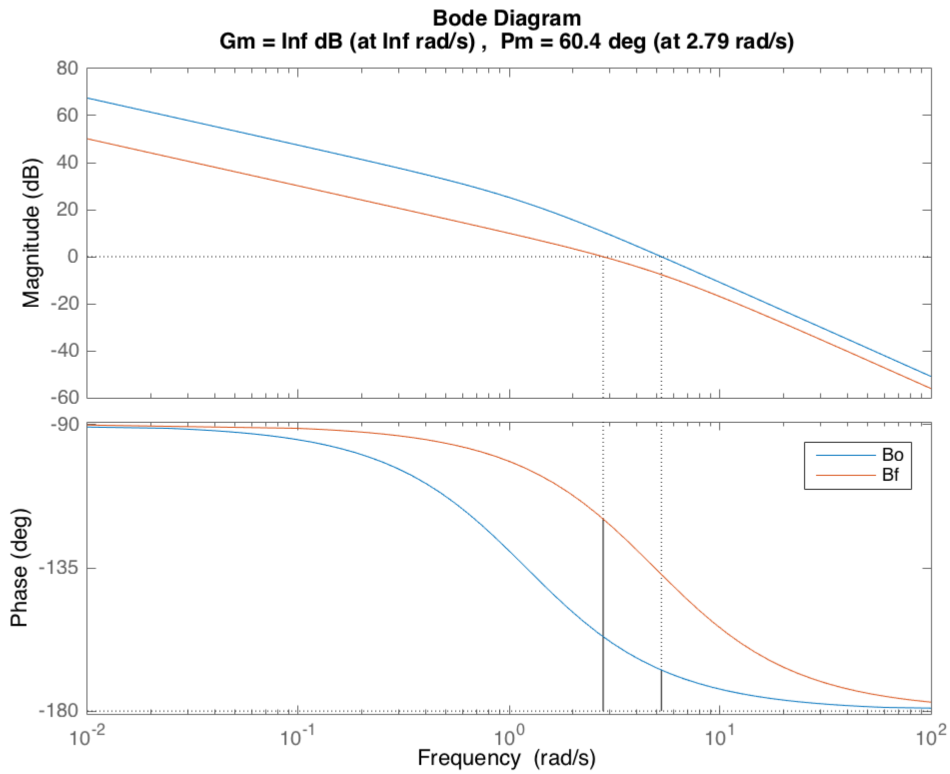
On trouve alors K en utilisant K_d :

$$K = \frac{2 \cdot (B_{eff} + 8 \cdot J_{eff} - B)}{\frac{K_m}{R}} = \frac{16 \cdot J_{eff}}{\frac{K_m}{R}}$$

```
K = (16*Jeff) / Km_R;
Kd = (8*Jeff - Beff) / Km_R;
```

(b) : Lieux de transfert de la boucle ouverte :

```
d = 1;
Bo = series((Km_R-r*d),G);
Bf = feedback(K*Bo,tf([Kd 0],1));
margin(Bo);
title('Marge de phase et marge de gain');
hold on;
margin(Bf);
legend('Bo', 'Bf');
hold off;
```

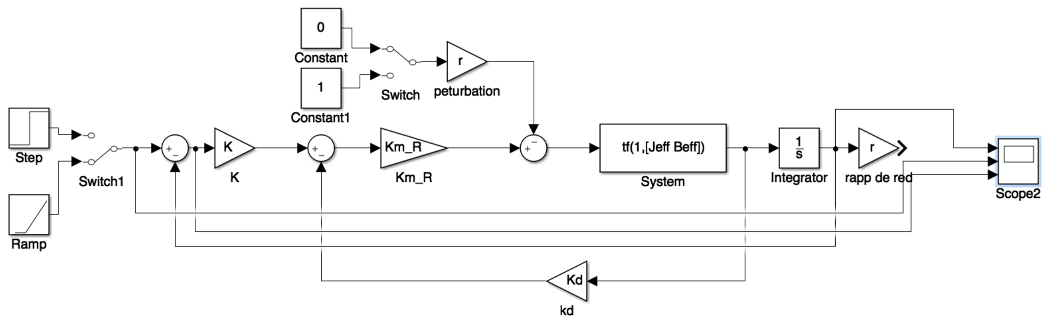


Avant correction, la marge de phase est faible proche de -180° Apres correction, la marge de phase est plus grande au point critique.

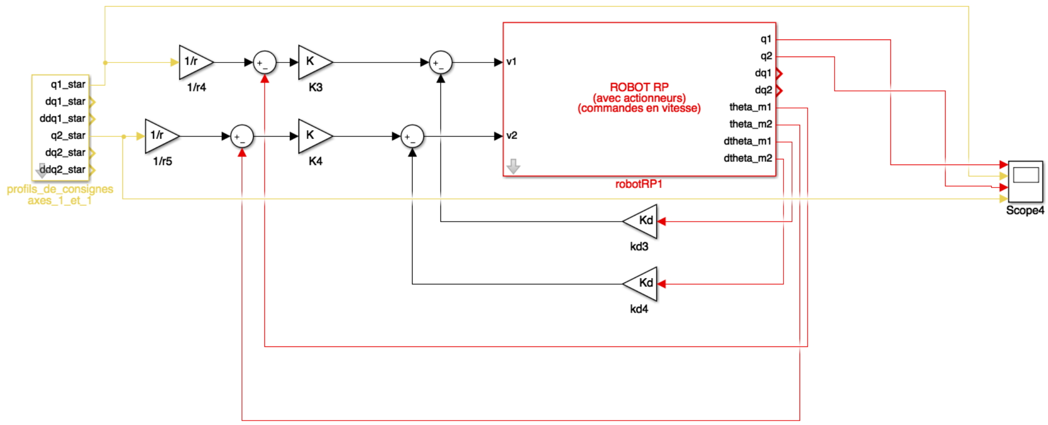
(c) : Simulink En l'absence de pertubation, l'erreur de position est nulle et l'erreur de vitesse est bien de 1/2.

Avec la perturbation, le système n'est pas capable de corriger la perturbation et cette dernière s'ajouter à notre erreur de position ainsi que de vitesse.

Modèle linéaire :



Modèle non linéaire :



Question 5 :

On a un modèle linéaire valable pour un rapport de réduction r élevé. S'il n'y a pas de de perturbations et pas de gravité, le système est correctement asservi

Le modèle met plus de temps à se stabiliser dans le cas non lineaire.

Commande en vitesse de type PID

On se place dans l'hypothèse où r=1/200

Question 6 : calcul de T_I :

```
Ti = 20000;
BFPID = Bf * tf([Ti 1], [Ti 0]);
```

Il faut retoucher le Ti pour l'axe 2 afin que la correction soit satisfaisante sur q2

```
Ti2 = 12;
```

Dans la partie simulink nous remarquons que les sorties du système n'ont pas d'erreur en regime permanent alors qu'elles ont un faible retard ainsi qu'un dépassement selon la valeur de Ti.

Retour sur la modélisation

Question 7 : Expression de Jeffi(q) et rdi(q) fonctions des coordonnées articulaires du robot.

Actionneur #k

$$J_m \cdot \ddot{\theta}_m + B_{eff} \cdot \dot{\theta}_m = \frac{K_m}{R} \cdot v_k - r \cdot d_k$$

$$\begin{aligned} r \cdot d_k &= r \cdot (D \cdot \ddot{q} + B + G) \\ &= r \cdot \begin{pmatrix} m \cdot q_2^2 & 0 \\ 0 & m \end{pmatrix} \cdot \begin{pmatrix} \ddot{q}_1 \\ \ddot{q}_2 \end{pmatrix} + r \cdot \begin{pmatrix} 2 \cdot m \cdot q_2 \cdot \dot{q}_1 \cdot \dot{q}_2 \\ -m \cdot q_2 \cdot \dot{q}_1^2 \end{pmatrix} - r \begin{pmatrix} m \cdot q_2 \cdot g_y \cdot \cos(q_1) \\ m \cdot g_y \cdot \sin(q_1) \end{pmatrix} \\ &= r \cdot D \cdot \begin{pmatrix} \ddot{q}_1 \\ \ddot{q}_2 \end{pmatrix} + r \cdot \begin{pmatrix} d_1 \\ d_2 \end{pmatrix} \end{aligned}$$

Avec :

$$\begin{aligned} d_k &= B + G \\ &= \begin{pmatrix} 2 \cdot m \cdot q_2 \cdot \dot{q}_1 \cdot \dot{q}_2 \\ -m \cdot q_2 \cdot \dot{q}_1^2 \end{pmatrix} - \begin{pmatrix} m \cdot q_2 \cdot g_y \cdot \cos(q_1) \\ m \cdot g_y \cdot \sin(q_1) \end{pmatrix} \\ &= \begin{pmatrix} 2 \cdot m \cdot q_2 \cdot \dot{q}_1 \cdot \dot{q}_2 - m \cdot q_2 \cdot g_y \cdot \cos(q_1) \\ -m \cdot q_2 \cdot \dot{q}_1^2 - m \cdot g_y \cdot \sin(q_1) \end{pmatrix} \end{aligned}$$

Avec toutes les écritures précédentes, on peut écrire :

$$(J_m + r^2 \cdot D) \cdot \ddot{\theta}_m + B_{eff} \cdot \dot{\theta}_m = \frac{K_m}{R} \cdot v_k - r \cdot (B + G)$$

On peut donc approximé une inertie efficace par :

$$J_{eff} = valeur\ pire\ cas\ de\ (J_m + r^2 \cdot D)$$

Question 8 : Pour chaque axe i, on va calculer à l'aide de Matlab les inerties efficaces extrêmes.

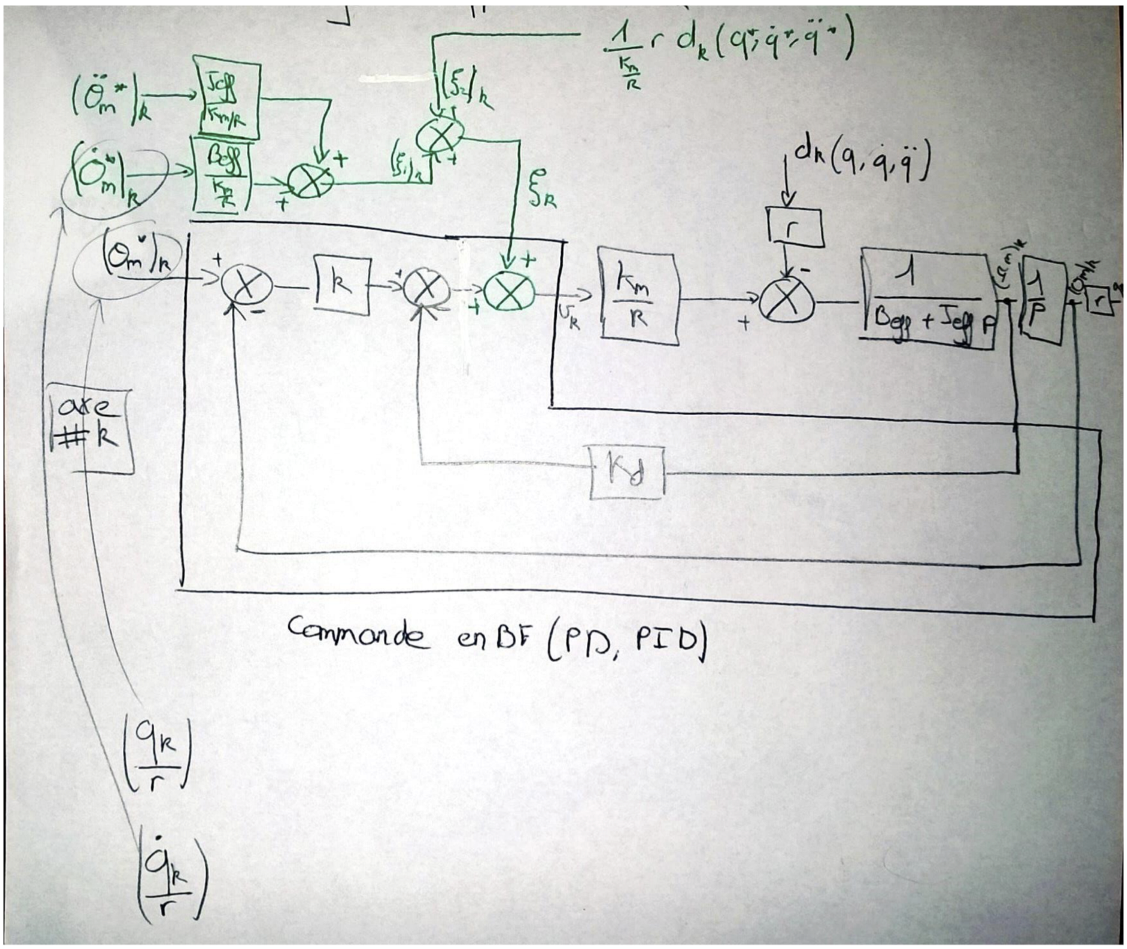
```
q1min = 0;
q1min = pi/2;
q2min = 0.5;
q2max = 1;

Dmin = [m*q2min^2 0 ; 0 m];
Dmax = [m*q2max^2 0 ; 0 m];
d11min = Dmin(1,1);
d22min = Dmin(2,2);
d11max = Dmax(1,1);
d22max = Dmax(2,2);

Jeff1min =  Jm + r*d11min
Jeff1max =  Jm + r*d11max
Jeff2min =  Jm + r*d22min
Jeff2max =  Jm + r*d22max


Jeff1min =

    0.0287


Jeff1max =

    0.0850


Jeff2min =

    0.0850


Jeff2max =
```

```
0.0850
```

Plus un système a de l'inertie, plus il est dur à contrôler, donc on prend les valeurs dont l'inertie est la plus faible, c'est à dire : Jeff1min et Jeff2min

Commande non linéaire centralisée par anticipation

Implémentation du feedforward :

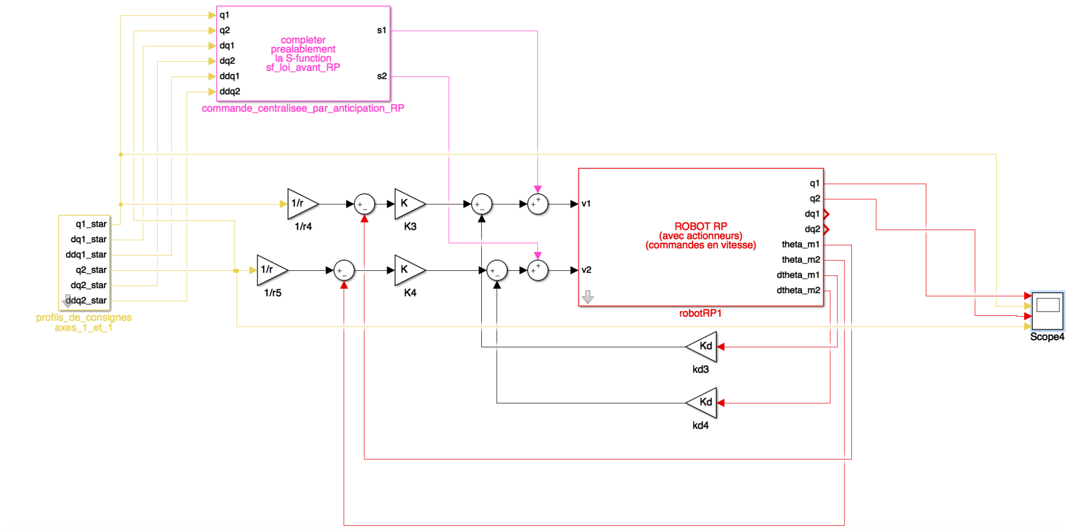


Code de l'implémentation sur Matlab

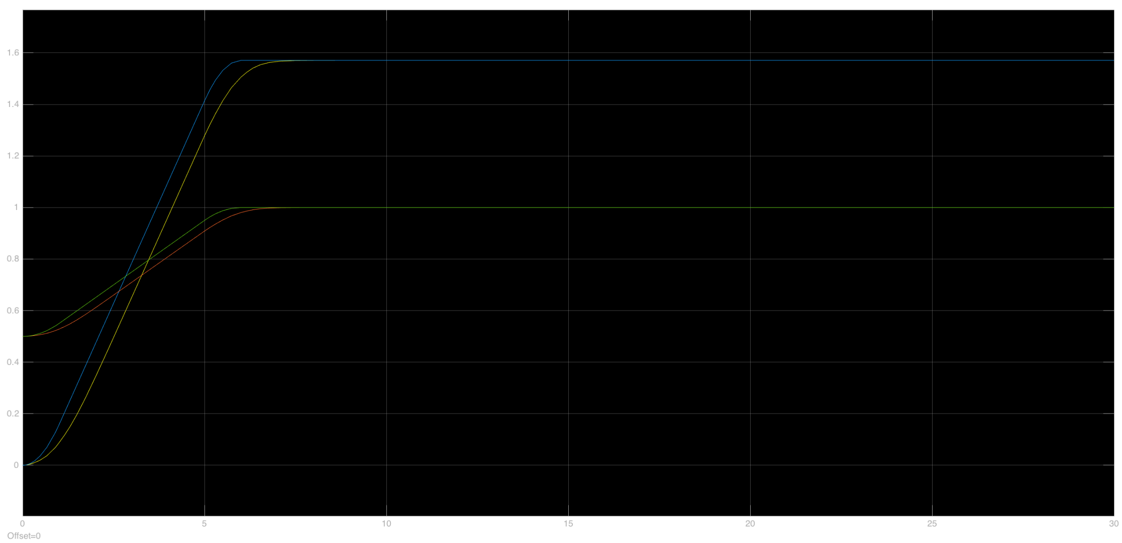
```
function [sys,x0,str,ts] = sf_loi_avant_RP(t,x,u,flag)
% NE PAS MODIFIER CI-DESSOUS ; CF. SEULEMENT mdlOutputs PLUS BAS
%
switch flag,
  case 0,
    [sys,x0,str,ts] = mdlInitializeSizes;
  case 3,
```

```
    sys = mdlOutputs(t,x,u);
  case {1,2,4,9}
    sys = [];
  otherwise
    error(['Unhandled flag = ',num2str(flag)]);
end
%==============================================================================
function [sys,x0,str,ts] = mdlInitializeSizes
% NE PAS MODIFIER CI-DESSOUS ; CF. SEULEMENT mdlOutputs PLUS BAS
%
sizes = simsizes;
sizes.NumContStates  = 0;
sizes.NumDiscStates  = 0;
sizes.NumOutputs     = 2;               % [d1;d2]
sizes.NumInputs      = 6;               % [q1;q2;dq1;dq2;ddq1;ddq2]
sizes.DirFeedthrough = 1;
sizes.NumSampleTimes = 1;
sys = simsizes(sizes);
x0  = [];
str = [];
ts  = [-1 0];                           % période héritée du bloc père
%==============================================================================
function sys = mdlOutputs(t,x,u)
%
q1 = u(1); q2 = u(2); dq1 = u(3); dq2 = u(4); ddq1 = u(5); ddq2 =
 u(6);
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% À COMPLÉTER À PARTIR D'ICI %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% sys DOIT ÊTRE INTIALISÉ AVEC LE CONTENU DU VECTEUR DE SORTIE DU BLOC
% e.g.
Km_R = 0.3;
Beff = 1/80;
Jm = 1/100;
R = [1/200 1/30 1];
r = R(1);
m = 15;
Jeff = Jm+r^2*m;
gy = -9.81;
d1 = 2*m*q2*dq1*dq2-m*q2*gy*cos(q1);
d2 = -m*q2*dq1*dq1-m*gy*sin(q1);
s1 = ddq1/r * Jeff/Km_R + dq1/r * Beff/Km_R + r*d1/Km_R;
s2 = ddq2/r * Jeff/Km_R + dq2/r * Beff/Km_R + r*d2/Km_R;
sys = [s1;s2];

%==============================================================================
```

Réalisation du montage sous Simulink



On obtient les réponses suivantes pour l'axe 1 et 2 pour r=1/200



A l'aide de plusieurs simulations, on remarque que plus on augmente la valeur de r, plus la valeur de l'axe 2 devient aberrante. Pour un r=1/200, la réponse est très bonne, la valeur voulue pour l'axe 1 et 2 est atteinte rapidemment.

Commande non linéaire centralisée par découplage

On souhaite maintenant mettre en place une commande en boucle fermée linéarisante découplante.

Code de l'implémentation sur Matlab

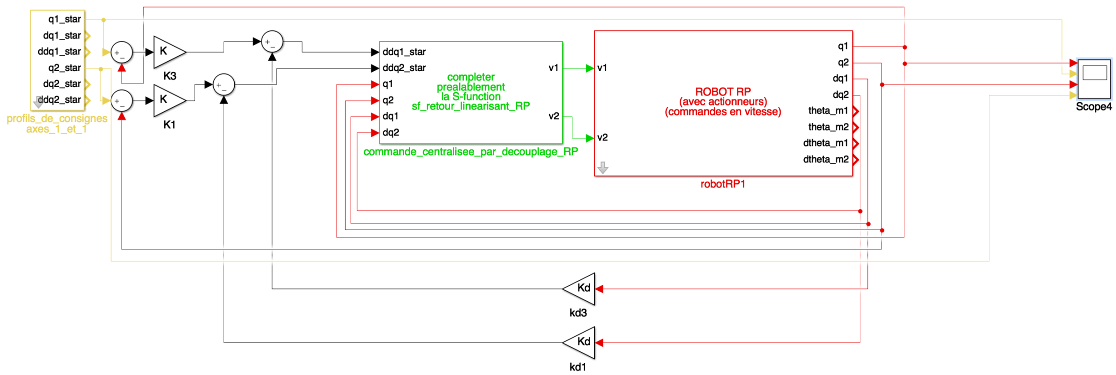
```
function [sys,x0,str,ts] = sf_retour_linearisant_RP(t,x,u,flag)
% NE PAS MODIFIER CI-DESSOUS ; CF. SEULEMENT mdlOutputs PLUS BAS
%
```

```
switch flag,
  case 0,
    [sys,x0,str,ts] = mdlInitializeSizes;
  case 3,
    sys = mdlOutputs(t,x,u);
  case {1,2,4,9}
    sys = [];
  otherwise
    error(['Unhandled flag = ',num2str(flag)]);
end
%==============================================================================
function [sys,x0,str,ts] = mdlInitializeSizes
% NE PAS MODIFIER CI-DESSOUS ; CF. SEULEMENT mdlOutputs PLUS BAS
%
sizes = simsizes;
sizes.NumContStates  = 0;
sizes.NumDiscStates  = 0;
sizes.NumOutputs     = 2;              % [d1;d2]
sizes.NumInputs      = 6;              % [q1;q2;dq1;dq2;ddq1;ddq2]
sizes.DirFeedthrough = 1;
sizes.NumSampleTimes = 1;
sys = simsizes(sizes);
x0  = [];
str = [];
ts  = [-1 0];                          % période héritée du bloc père
%==============================================================================
function sys = mdlOutputs(t,x,u)
%
ddq1_star = u(1); ddq2_star = u(2);
q1 = u(3); q2 = u(4); dq1 = u(5); dq2 = u(6);
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% À COMPLÉTER À PARTIR D'ICI %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% sys DOIT ÊTRE INTIALISÉ AVEC LE CONTENU DU VECTEUR DE SORTIE DU BLOC
% e.g.
Km_R = 0.3;
Beff = 1/80;
Jm=1/100;
R = [1/200 1/30 1];
r = R(1);
m = 15;
Jeff=Jm+r^2*m;
gy=-9.81;


d1 = 2*m*q2*dq1*dq2-m*dq2*gy*cos(q1);
d2 = -m*q2*dq1*dq1-m*gy*sin(q1);
s1 = ddq1_star/r * Jeff/Km_R + dq1/r * Beff/Km_R + r*d1/Km_R;
s2 = ddq2_star/r * Jeff/Km_R + dq2/r * Beff/Km_R + r*d2/Km_R;

sys = [s1;s2];
%==============================================================================
```

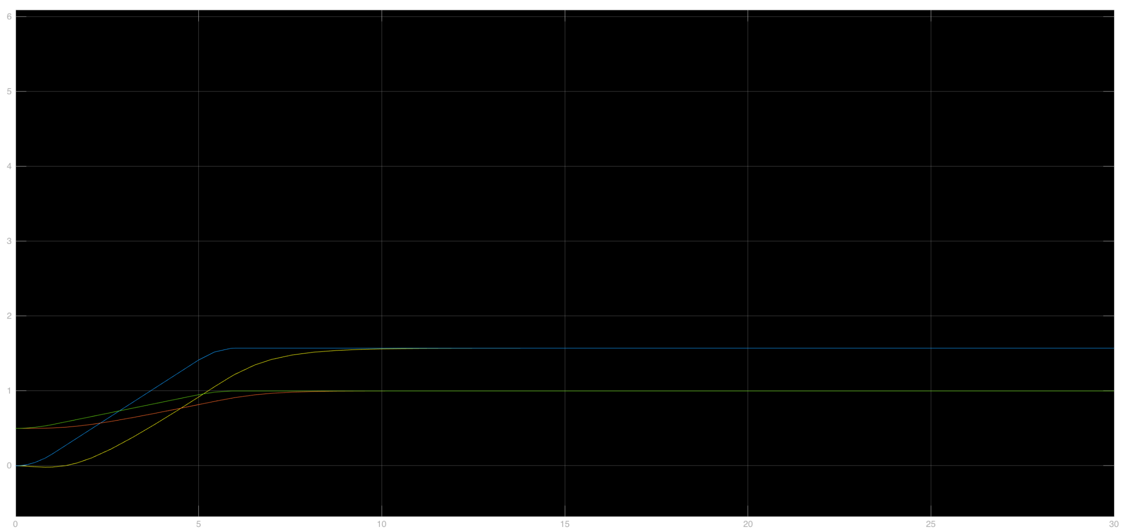
Nouvelle valeur de K et Kd pour la commande PD

```
Kd = 3;
K = (Kd^2)/4;
```

Montage du système via Simulink



Résultat via un montage Simulink pour r=1/200



On peut remarquer que le système a juste un léger retard sur le temps de montée mais exécute bien la commande voulue.

Published with MATLAB® R2015b