ROB315 - TP
TP1 - Cinématiques directe et inverse
TP2 - Dynamique et commande

BRAMBILLA Davide Luigi - GOMES DA SILVA Rafael

20 Décembre 2020

TP 1

1 Modèle géométrique direct

Le *modèle géométrique direct* (MGD) est un modèle mécanique utilisé afin d'étudier le comportement des bras manipulateurs. Il donne la possibilité de calculer les *coordonnées opérationnelles*, qui donnent la situation de l'organe terminal, en fonction des *coordonnées articulaires*, qui donnent la position de chaque moteur du robot et permettent donc de déterminer la position et l'orientation de l'organe terminal.

1.1 Q1: Convention Denavit-Hartenberg Modifiée (DHM)

Nous avons complété la figure 2 en lui donnant les repères des corps successifs du robot. Nous avons utilisé la convention *Denavit-Hartenberg Modifiée* (DHM) dite de *Khalil-Kleinfinger*.
Ci dessous nous avons reporté la figure 1.1 qui contient les repères que nous avons associé au robot:

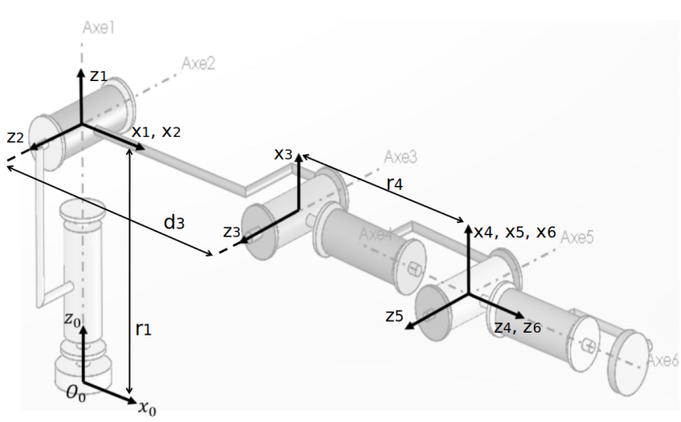

Figure 1.1: Repères de notre bras manipulateur

1.2 Q2: Tableau DHM

Le paramètres géométriques qui nous intéressent afin de pouvoir étudier le modèle géométrique direct sont:

- α_i: qui est l'angle autour de l'axe x_{i-1} entre les axes z_{i-1} et z_i
- d_i: qui est la distance le long de l'axe x_{i-1} entre les axes z_{i-1} et z_i
- θ_i: qui est l'angle autour de l'axe z_i entre les axes x_{i-1} et x_i
- r_i: qui est la distance le long de l'axe z_i entre les axes x_{i-1} et x_i

Nous avons 6 articulations rotoides qui donc vont tourner autour de leur axe z.
Nous avons reporté ci-dessous le tableau complété avec les paramètres géométriques du robot pour chaque repere.

i	α_i	d_i	θ_i	r_i
1	0	0	q_1	r_1
2	$\frac{\pi}{2}$	0	q_2	0
3	0	d_3	$\frac{\pi}{2}+q_3$	0
4	$\frac{\pi}{2}$	0	q_4	r_4
5	$-\frac{\pi}{2}$	0	q_5	0
6	$\frac{\pi}{2}$	0	q_6	0

1.3 Q3: Fonctions pour le MGD du robot

1.3.1 Q3.1: Calcule de la matrice de transformation homogène

La matrice de transformation homogène nous permet de passer d'un repère $\mathscr{R}_s$ au repère $\mathscr{R}_d$ suivant.

$$\overline{g}_{sd} = \begin{bmatrix} \boldsymbol{R_{sd}} & \boldsymbol{p_{sd}} \\ \boldsymbol{0_{1x3}} & 1 \end{bmatrix}$$

Nous avons implémenté sur *Matlab* une fonction générique *CalculTransformationElem(α_i, d_i, θ_i, r_i)* que nous avons reporté ci-dessous:

Code 1: *CalculTransformationElem(α_i, d_i, θ_i, r_i)*

```
function g = CalculTransformationElem(alpha, d, theta, r)
    g = [cos(theta) -sin(theta) 0 d;...
        cos(alpha)*sin(theta) cos(alpha)*cos(theta) -sin(alpha) -r*sin(alpha);...
        sin(alpha)*sin(theta) sin(alpha)*cos(theta) cos(alpha) r*cos(alpha);...
        0 0 0 1;];
end
```

1.3.2 Q3.2: Calcule de la matrice de transformation homogène

Dans cette section il nous es demandé d'implémenter un fonction *CalculMGD(α, d, θ, r)* qui nous permet de calculer le modèle géométrique directe d'un robot en prenant en argument les paramètres géométriques du robot (α, d, θ, r).

Code 2: *CalculMGD(alpha, d, theta, r)*

```
function [g_06, g_elem] = CalculMGD(alpha, d, theta, r)
    g_06 = eye(4);
    N = length(alpha);
    g_elem = cell(1,N);
    for i = 1 : N
        g_elem{i} = CalculTransformationElem(alpha(i), d(i), theta(i), r(i));
        g_06 = g_06 * g_elem{i};
    end
end
```

1.3.3 Q3.3: Calcule le MGD d'un robot

Pour exécuter cette partie nous avons écrit une section à l'intérieure du *TP1.m* qui va calculer la matrice de transformation homogène $\bar{g}_{0E}$:

Code 3: *Section du TP1.m afin de calculer le MGD du robot*

```
%% Q3: MGD du robot
g_6E = CalculTransformationElem(0,0,0,rE);
[g_06, g_element] = CalculMGD(alpha, d, theta, r);
g_0E = g_06 * g_6E
```

Nous avons reporté ci-dessous le résultat:

Pour $q = q_i$:

$$\overline{g}_{0E} = \begin{bmatrix} 0 & 0 & -1 & -0.1 \\ 1 & 0 & 0 & -0.7 \\ 0 & -1 & 0 & 0.3 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Pour $q = q_f$:

$$\overline{g}_{0E} = \begin{bmatrix} -0.7 & 0.7 & 0.0 & 0.64 \\ 0.0 & 0.0 & -1.0 & -0.1 \\ -0.7 & -0.7 & 0.0 & 1.13 \\ 0.0 & 0.0 & 0.0 & 1.0 \end{bmatrix}$$

1.4 Q4: Calcule du MGD d'un robot

Afin de calculer les positions de l'organe terminale nous avons continué le script *main* en calculant les positions P_x, P_y et P_z et les paramètres liés à l'orientation. Afin de faire cela nous avons implémenté les lignes de code suivantes dans le main:

Code 4: *Section du TP1.m afin de calculer la position de l'organe terminale*

```
%% Q4: Axe et angle de rotation quivalent  une rotation donne

% Vecteur de position
P_0E = g_0E(1:3, 4)
```

```
% Matrice de rotation
R_0ER = g_0E(1:3, 1:3);
% Trouve l'angle q et l'axe n a partir d'une rotation donne
[ang_q,n] = AngleAxe_from_Rot (R_0ER)
```

Code 5: *Function AngleAxe_from_Rot (R)*

```
function [ang_q,n] = AngleAxe_from_Rot (R)
    % Angle de rotation quivalent  une rotation donne
    ang_q = atan2(0.5*sqrt((R(3,2) - R(2,3))^2 +...
                (R(1,3) - R(3,1))^2 + (R(2,1) - R(1,2))^2), ...
                0.5*(R(1,1)+ R(2,2) + R(3,3)-1));
    % Axe de rotation quivalent  une rotation donne
    n = [(R(3,2) - R(2,3))/(2*sin(ang_q)), ...
        (R(1,3) - R(3,1))/(2*sin(ang_q)), ...
        (R(2,1) - R(1,2))/(2*sin(ang_q))];
end
```

Ci-dessous je vais reporter les résultats obtenus pour les deux configurations:

1.4.1 Configuration articulaire initiale

Pour la configuration initiale nous avons défini le vecteur q comme:

$$q_i = \begin{bmatrix} -\frac{\pi}{2}, & 0 & -\frac{\pi}{2} & -\frac{\pi}{2} & -\frac{\pi}{2} & -\frac{\pi}{2} \end{bmatrix}'$$

et nous avons obtenu les résultats suivants:

$$P = \begin{bmatrix} P_x & P_y & P_z \end{bmatrix}' = \begin{bmatrix} -0.1 & -0.7 & 0.3 \end{bmatrix}'$$

$$n = \begin{bmatrix} -0.5773 & -0.5773 & 0.5773 \end{bmatrix}'$$

$$q = 2.0943$$

1.4.2 Configuration articulaire finale

Pour la configuration initiale nous avons défini le vecteur q comme:

$$q_f = \begin{bmatrix} 0 & \frac{\pi}{4} & 0 & \frac{\pi}{2} & \frac{\pi}{2} & 0 \end{bmatrix}'$$

et nous avons obtenu les résultats suivants:

$$P = \begin{bmatrix} P_x & P_y & P_z \end{bmatrix}' = \begin{bmatrix} 0.6364 & -0.1 & 1.1364 \end{bmatrix}'$$

$$n = \begin{bmatrix} 0.2811 & 0.6786 & -0.6786 \end{bmatrix}'$$

$$q = 2.5936$$

1.5 Q5: Fonction pour la visualisation des Repères

Pour cette étape, la fonction "*VisualisationRepere*" montre à la fois les repères du robot et les éléments de son corps. Pour cela, d'abord on fait le calcul de la MGD de notre robot à partir des paramètres souhaités, puis nous supposons que le point fixe du robot est à (0,0,0) et que les angles de rotation de la matrice de rotation valent 0^o.

Pour déterminer la séquence des points a partir du point initial, nous avons utilisé la relations ci-dessous. La équation (1) réalise la transformation d'un point à l'aide d'une matrice de homogène, et l'équation (2) effectue la mise à jour de la matrice homogène.

$$^0p(t) = R_{0i}(t)^i p \tag{1}$$

$$\overline{g}_{0N}(q) = \overline{g}_{01}(q_1)...\overline{g}_{(i-1)i}(q_i)...\overline{g}_{(N-1)N}(q_N) \tag{2}$$

Les résultats obtenus sont reportés dans les figures ci-dessous. La Figure 1.2 montre le résultat pour le cas avec la configuration articulaire $q = q_i$, et la Figure 1.3 montre le résultat pour le cas avec la configuration articulaire $q = q_f$.

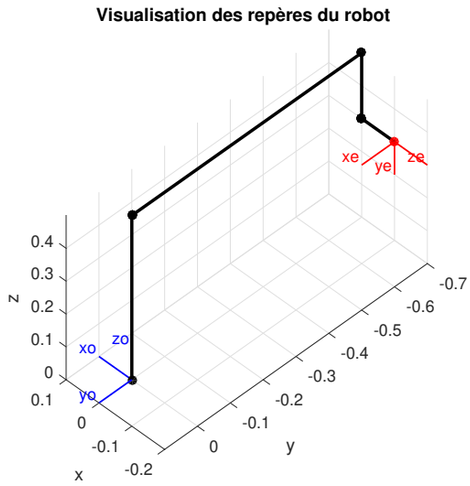

Figure 1.2: Affichage des repères de notre bras manipulateur pour la configuration articulaire $q = q_i$

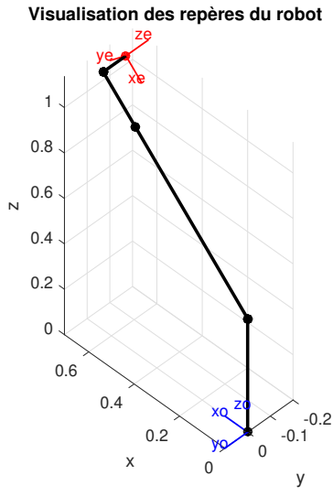

Figure 1.3: Affichage des repères de notre bras manipulateur pour la configuration articulaire $q = q_f$

Les codes ci-dessous montrent la fonction "*VisualisationRepere*" utilisé pour représenter le robot après une certaine configuration articulaire, ainsi que la fonction "*plot_articulation*" qui trace le corps robot et la fonction "*plot_repere*" qui trace les repères $\mathscr{R}_O$ et $\mathscr{R}_E$ du robot.

Code 6: *Section du TP1.m afin de afficher les repères du robot*

```
%% Q5: Visualisation des repere du robot
figure('Name','Question 05: Visualisation des repere du robot','NumberTitle','off')
VisualisationRepere(q, 1, 'black');
title('Visualisation des repres du robot');
```

Code 7: *Function VisualisationRepere(q, label$_{On}$, rep_color)*

```
function VisualisationRepere(q, labelOn, rep_color)
    % Parametres
    r4 = 0.2;
    r1 = 0.5;
    r = [r1 0 0 r4 0 0];
    d3 = 0.7;
    d = [0 0 d3 0 0 0];
    alpha = [0 pi/2 0 pi/2 -pi/2 pi/2];
    rE = 0.1;
    theta = [q(1) q(2) pi/2+q(3) q(4) q(5) q(6)];

    % MGD du robot
    g_6E = CalculTransformationElem(0,0,0,rE);
    [g_06,g_element] = CalculMGD(alpha, d, theta, r);

    %Point de dpart: (0,0,0)
    origin = [0;0;0] ;
    % q = 0 pour les matrices de Rotation Rxq, Ryq et Rzq initialement
    RotN = eye(3) ;
```

```
    % Plot R0
    plot_repere(origin, RotN, 'blue','o', labelOn, rep_color) ;

    % Plot Articulations et RE
    plot_articulation(origin, RotN, g_element, g_6E, labelOn, rep_color);


    xlabel('x');
    ylabel('y');
    zlabel('z');
    grid on;
    axis equal;
    view(-135,45)
end
```

Code 8: *Fonction "plot_articulation" dans le fichier VisualisationRepere.m*

```
function plot_articulation(origin, RotN, g_element, g_6E, labelOn, rep_color)
    position = origin;
    for i = 1:length (g_element)
        position = origin + RotN*g_element{i}(1:3,4);
        RotN = RotN*g_element{i}(1:3,1:3);
        plot3([origin(1) position(1)],[origin(2) position(2)],[origin(3)
            position(3)],rep_color,'LineWidth',2);
        hold on
        scatter3(origin(1),origin(2),origin(3),rep_color,'filled');
        hold on
        scatter3(position(1),position(2),position(3),rep_color,'filled');
        hold on
        origin = position;
    end

    % Position et orientation du repre terminal
    position = origin + RotN*g_6E(1:3,4);
    RotN = RotN*g_6E(1:3,1:3);

    %Plot the terminal
    plot3([origin(1) position(1)],[origin(2) position(2)],[origin(3)
        position(3)],rep_color,'LineWidth',2);
    hold on
    scatter3(position(1),position(2),position(3),'red','filled');
    hold on

    %Plot terminal
    plot_repere(position, RotN, 'red', 'e', labelOn, rep_color)
end
```

Code 9: *Fonction "plot_repere" dans le fichier VisualisationRepere.m*

```
function plot_repere(origin, RotN, c, id, labelOn, rep_color)
    point = origin;
    rotation = RotN;
    x = [1;0;0];
    y = [0;1;0];
    z = [0;0;1];
    O1 = 0.15 * rotation * x;
    O2 = 0.15 * rotation * y;
    O3 = 0.15 * rotation * z;
    plot3([point(1) point(1)+O1(1)],[point(2) point(2)+O1(2)],[point(3)
        point(3)+O1(3)],c,'LineWidth',1);
    hold on
    if labelOn
        text(0.05 +point(1)+O1(1),0.05 +point(2)+O1(2),0.05 +point(3)+O1(3), strcat('x',id),
            'Color', c);
    end


    plot3([point(1) point(1)+O2(1)],[point(2) point(2)+O2(2)],[point(3)
        point(3)+O2(3)],c,'LineWidth',1);
    hold on
    if labelOn
        text(0.05 +point(1)+O2(1),0.05 +point(2)+O2(2),0.05 +point(3)+O2(3), strcat('y',id),
            'Color', c);
    end

    plot3([point(1) point(1)+O3(1)],[point(2) point(2)+O3(2)],[point(3)
        point(3)+O3(3)],c,'LineWidth',1);
```

```
    hold on
    if labelOn
        text(0.05 +point(1)+O3(1),0.05 +point(2)+O3(2),0.05 +point(3)+O3(3), strcat('z',id),
            'Color', c);
    end
end
```

2 Modèle cinématique direct

Le *modèle cinématique direct* (MCD) calcule la vitesse dans le domaine *cartésien* en fonction de la *vitesse articulaire* du robot. Pour faire cela il utilise la matrice Jacobienne que nous allons calculer dans la section suivante.

2.1 Q6: Calcul de la Jacobienne

Afin de calculer la matrice *jacobienne* nous avons implémenté la fonction *CalculJacobienne(α, d, θ, r)*. Cette matrice nous permet d'exprimer les vitesses des *coordonnées opérationnelles* du repere terminal en fonction des *vitesses articulaires*.

Code 10: *Section du TP1.m afin de calculer la Jacobienne et les torseurs cinématiques du robot*

```
%% Q6: Calcul de la Jacobienne et torseurs cinmatiques

% Vitesse articulaire
q_point = [0.5 1 -0.5 0.5 1 -0.5]';

% Calcula de la Jacobienne
J = CalculJacobienne(alpha, d, theta, r)
V0E0E = J * q_point
```

Nous avons reporté ci-dessous les résultats:

1. Pour $q = q_i$

$$J = \begin{bmatrix} 0.70 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 \\ -0.10 & -0.20 & -0.20 & 0.10 & 0.00 & 0.00 \\ 0.00 & 0.70 & 0.00 & 0.00 & -0.10 & 0.00 \\ 0.00 & -1.00 & -1.00 & 0.00 & 0.00 & -1.00 \\ 0.00 & 0.00 & 0.00 & 0.00 & -1.00 & 0.00 \\ 1.00 & 0.00 & 0.00 & -1.00 & 0.00 & 0.00 \end{bmatrix}$$

2. Pour $q = q_f$

$$J = \begin{bmatrix} 0.10 & -0.63 & -0.14 & 0.07 & -0.07 & 0.00 \\ 0.63 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 \\ 0.00 & 0.63 & 0.14 & -0.07 & -0.07 & 0.00 \\ 0.00 & 0.00 & 0.00 & 0.70 & 0.70 & 0.00 \\ 0.00 & -1.00 & -1.00 & 0.00 & 0.00 & -1.00 \\ 1.00 & 0.00 & 0.00 & 0.70 & -0.70 & 0.00 \end{bmatrix}$$

Pour effectuer le calcul jacobien, nous utilisons les considérations présentes à la page 86 du matériel de cours, qui propose un moyen pratique d'obtenir la matrice jacobienne par composition de vitesse a l'aide de l'équation (3).

$$^0J_i(q) = \begin{bmatrix} R_{0i}(Z_i \times p_{iN}) \\ R_{0i}(Z_i) \end{bmatrix} \dot{q}_i \tag{3}$$

- R_{0i}: Matrice de rotation de $\mathscr{R}_i$ vers $\mathscr{R}_0$
- Z_i vecteur $\begin{bmatrix} 0 & 0 & 1 \end{bmatrix}^T$ de l'articulation i.
- p_{iN} vecteur de translation (de $\mathscr{R}_i$ vers $\mathscr{R}_N$)

Le code utilisé est indiqué ci-dessous:

Code 11: *CalculJacobienne(α, d, θ, r)*

```
function J = CalculJacobienne(alpha, d, theta, r)
    rE = 0.1;
    [g_06,g_elem] = CalculMGD(alpha, d, theta, r);
    g_6E = CalculTransformationElem(0,0,0,rE);
    g_0E = g_06 * g_6E;

    N = length(theta);    % nombre d'axes du robot
    J = zeros(6,N);       % Initialisation de la matrice jacobienne
    z_i = [0 0 1]';       %Vecteur unitaire portant l'articulation i
```

```
    for i=1:N
        p_iE = zeros(3,1);
        g_0i = eye(4);
        for j=1:i
            g_0i = g_0i*g_elem{j};
        end

        %Calcul de R_0i
        R_0i = g_0i(1:3,1:3);

        %calcul de p_iE (vecteur de translation du repere R_i  R_E)
        p_0E = g_0E(1:3,4);

        p_0i = g_0i(1:3,4);
        p_iE = p_0E - p_0i;

        %Calcul de Ji dans le repere R_0
        oJi = [cross(R_0i*z_i,p_iE) ; R_0i*z_i];

        %Concatenation de la matrice J
        J(1:6,i) = oJi;
    end
end
```

2.1.1 Le calcul des torseurs cinématiques

Afin de calculer les *torseurs cinématiques* nous avons défini la vitesse articulaire comme donnée dans l'énonce du TP:

$$\dot{q} = \begin{bmatrix} 0.5 & 1 & -0.5 & 0.5 & 1 & -0.5 \end{bmatrix}'$$

et nous avons utilisé la formule reporté dans l'énonce selon laquelle la multiplication entre la matrice jacobienne et les vitesses articulaires donnent les torseurs cinématiques.

Torseur cinématique pour $q = q_i$

Pour la configuration articulaire $q = q_i$ et en utilisant ${}^0\mathscr{V}_{i,N}(O_E) = {}^0 J_i(q)\dot{q}$, nous avons le résultat suivant pour le torseur cinématique en O_E:

$${}^0\mathscr{V}_{i,N}(O_E) = \begin{bmatrix} 0.35 & -0.1 & 0.6 & 0 & -1 & 0 \end{bmatrix}'$$

Torseur cinématique pour $q = q_f$

Pour la configuration articulaire $q = q_f$ et en utilisant ${}^0\mathscr{V}_{i,N}(O_E) = {}^0 J_i(q)\dot{q}$, nous avons le résultat suivant pour le torseur cinématique en O_E:

$${}^0\mathscr{V}_{i,N}(O_E) = \begin{bmatrix} -0.5510 & 0.3182 & 0.4596 & 1.0607 & 0 & 0.1464 \end{bmatrix}'$$

2.2 Q7: Qualification de la transmission des vitesses

Dans cette étape du TP, nous analyserons les effets de la transmission de vitesse du robot en considérant uniquement les vitesses de translation, représentées par ${}^0 J_v(q)$.

Pour vérifier la direction dans laquelle la transmission de vitesse est la plus importante, nous pouvons analyser les valeurs singuliers σ_i et les vecteurs propres $\boldsymbol{e_i}$ de JJ^T. Le vecteur propre associé à la plus grande valeur singulier représente la direction privilégiée pour une configuration donnée.

De plus, nous pouvons également mesurer la capacité du manipulateur à engendrer des vitesses dans certaines directions. Cette mesure est effectuée à l'aide de l'équation (4) que nous donne la valeur de la manipulabilité en vitesse du robot, où $\mathscr{W}$ est le volume de l'ellipsoïde de vitesse.

$$\mathscr{W} = \prod_{i=1}^{r} \sigma_i \geq 0 \tag{4}$$

Ci-dessous, nous présenterons, pour chacune des configurations articulaires q_i et q_f proposées, les vecteurs de direction privilégiée, les valeurs de manipulabilités en vitesse et une représentation graphique des ellipsoïdes en vitesse pour le robot étudié.

Code 12: *Section du TP1.m afin de calculer la transmission des vitesses*

```
%% Q7: Transmission des vitesses

    % Limitation de J  les vitesses de translation
    J7 = J(1:3,:)

    % Dcomposition en valeurs singuliers
    [U,S,V] = svd(J7*J7.')
```

```
%Direction  privilgi
[m,idx] = max(diag(S));
Direction_privilegie = V(:,idx);

% Calcul de la manipulabilit
eigenvalues = sqrt(diag(S));
manipulabilite = prod(eigenvalues);

% Qualit  de la transmission de la vitesse
VisualisationEllipsoides(J(1:3,:), q, g_0E);
title('Ellipsode de vitesse');
```

2.2.1 Configuration q_i

- **Direction privilégiée**

  En analysant les valeurs singulières et les vecteurs propres de JJ^T ci-dessous, il est possible d'observer que la direction privilégiée dans ce cas est celle dans laquelle pointe le vecteur $V = \begin{bmatrix}0.3660 & -0.3263 & 0.8715\end{bmatrix}$, et la manipulabilité du robot pour cette configuration est $\mathscr{W} = 0.1116$. La Figure 2.1 montre l'ellipsoïde de vitesse correspondent pour notre robot.

- **Valeurs singulières**

$$\sigma = \begin{bmatrix}0.5524 & 0 & 0\\ 0 & 0.4918 & 0\\ 0 & 0 & 0.0458\end{bmatrix}$$

- **Vecteur propres**

$$S = \begin{bmatrix}0.3660 & 0.9186 & 0.1489\\ -0.3263 & -0.0232 & 0.9450\\ 0.8715 & -0.3944 & 0.2913\end{bmatrix}$$

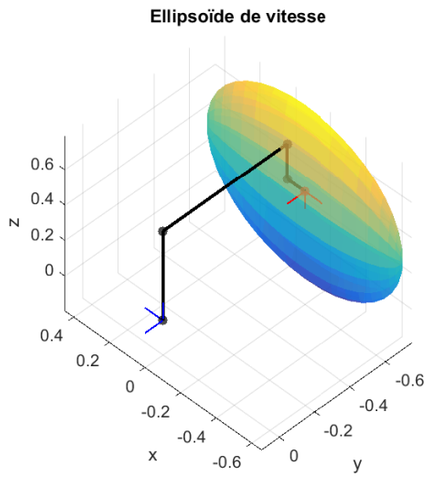

Figure 2.1: Visualisation de l'ellipsoïde de vitesse pour la configuration articulaire $q = q_i$

2.2.2 Configuration q_f

- **Direction privilégiée**

  En analysant les valeurs singulières et les vecteurs propres de JJ^T ci-dessous, il est possible d'observer que la direction privilégiée dans ce cas est celle dans laquelle pointe le vecteur $V = \begin{bmatrix}-0.7114 & -0.0975 & 0.6959\end{bmatrix}$, et la manipulabilité du robot pour cette configuration est $\mathscr{W} = 0.0590$. La Figure 2.2 montre l'ellipsoïde de vitesse correspondent pour notre robot.

- **Valeurs singulières**

$$\sigma = \begin{bmatrix}0.8695 & 0 & 0\\ 0 & 0.4057 & 0\\ 0 & 0 & 0.0099\end{bmatrix}$$

- **Vecteur propres**

$$V = \begin{bmatrix}-0.7114 & 0.0103 & -0.7027\\ -0.0975 & 0.9888 & 0.1132\\ 0.6959 & 0.1490 & -0.7025\end{bmatrix}$$

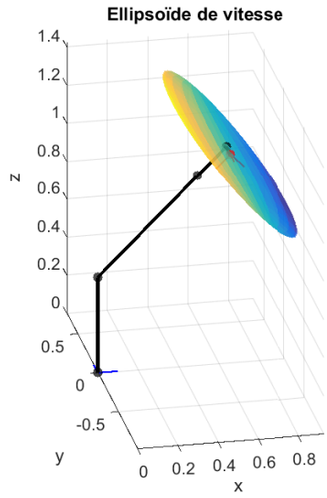

Figure 2.2: Visualisation de l'ellipsoid de vitesse pour la configuration articulaire $q = q_f$

Code 13: *Function VisualisationEllipsoides(J, q, g_{0E})*

```
function VisualisationEllipsoides(J, q, g_0E)
    % Dcomposition en valeurs singuliers
    [U,S,V] = svd(J*J.');

    % Ellipsoide des vitesses
    AxisValues = sqrt(diag(S));

    % Visualisation du Robot et les rperes
    VisualisationRepere(q, 0, 'black');

    %Rotation (De la Q4) Axe et angle de rotation quivalent  une rotation donne
    pf = g_0E(1:3,4);
    [q_r,n] = AngleAxe_from_Rot (V);

    % Ellipsodes de vitesse
    % [x,y,z] =  (xc,yc,zc,xr,yr,zr,n)
    [x,y,z] = ellipsoid(pf(1),pf(2),pf(3),AxisValues(1),AxisValues(2),AxisValues(3));
    Ellipsoide = surf(x,y,z);
    alpha 0.7;
    Ellipsoide.EdgeColor = 'none';
    rotate(Ellipsoide,n,radtodeg(q_r),pf);
end
```

3 Modèle géométrique inverse

Le *modèle géométrique inverse* (MGI) réalise la fonction inverse par rapport au modèle direct c'est à dire la détermination de la configuration des moteurs en fonction de la position de l'organe terminal.

3.1 Q8: Fonction pour le MGI

Nous avons donc implémenté la fonction *MGI* en lui donnant comme paramètres la position opérationnelle X_d desirée, la condition initiale q_0, la taille du pas α_{step}, le nombre maximal d'itérations k_{max} et l'erreur cartésien maximale tolérée ϵ_x qui permet d'arrêter l'algorithme dans le moment où nous sommes suffisamment proches de la solution.
Nous avons donc implémenté la fonction *MGI(X_d, q_0, k_{max}, ϵ_x, α_{step})* qui permet de calculer $q*$.

Code 14: *MGI($X_d, q, k_{max}, \epsilon_x, \alpha_{step}$)*

```
function q_etoile = MGI(Xd, q, kmax, epsx, aStep)

    % Calcul des parametres
    rE = 0.1;
    r4 = 0.2;
    r1 = 0.5;
    r = [r1 0 0 r4 0 0];
    d3 = 0.7;
    d = [0 0 d3 0 0 0];
    alpha = [0 pi/2 0 pi/2 -pi/2 pi/2];
    k = 0;

    while true
```

```
        k = k+1;

        theta = [q(1) q(2) pi/2+q(3) q(4) q(5) q(6)];

        J = CalculJacobienne(alpha, d, theta, r);
        Ji = J(1:3, :);

        [g_06, ~] = CalculMGD(alpha, d, theta, r);
        g_0E = g_06*CalculTransformationElem(0,0,0,rE);

        X = g_0E(1:3, 4);
        dX = Xd - X;

        % Methode de descente de gradient
%         q = q + aStep*J'*dX;

        % Methode de Newton-Raphson
        q = q + pinv(Ji) * dX;

        error = norm(dX, 2);
        % Condition de fin de la boucle
        if (error < epsx) || (k > kmax)
            break
        end
    end
    q_etoile = q;
end
```

et nous avons utilisé cette fonction dans les cas suivantes.

3.1.1 Premier Cas

Dans ce premier cas, nous avons initialisé les arguments comme donnée dans la consigne:

$$X_d = X_{d_i} = \begin{bmatrix} -0.1 & -0.7 & 0.3 \end{bmatrix}'$$

$$q_0 = \begin{bmatrix} -1.57 & 0 & -1.47 & -1.47 & -1.47 & -1.47 \end{bmatrix}'$$

$$k_{max} = 100$$

$$\alpha_{step} = 0.005$$

$$\epsilon_x = 1mm$$

En exécutant notre code:

Code 15: *Section du TP1.m afin de vérifier un cas pour le MGI du robot*

```
%% Q8.1 : Calculer modele geometrique inverse

% Parametres du model
X_di = [-0.1 -0.7 0.3]';
q0 = [-1.57 0 -1.47 -1.47 -1.47 -1.47]';
kmax = 100;
aStep = 0.005;
epsx = 0.001;

% Calcul de q*
q_etoile1 = MGI(X_di, q0, kmax, epsx, aStep);
```

Nous avons obtenu le résultat suivante:

$$q_1* = \begin{bmatrix} -1.5725 & 0.0132 & -1.5232 & -1.4452 & -1.4819 & -1.47 \end{bmatrix}'$$

3.1.2 Deuxième Cas

Dans le deuxième cas, nous avons initialisé les arguments comme donnée dans la consigne:

$$X_d = X_{d_f} = \begin{bmatrix} 0.64 & -0.1 & 1.14 \end{bmatrix}'$$

$$q_0 = \begin{bmatrix} 0 & 0.8 & 0 & 1 & 2 & 0 \end{bmatrix}'$$

$$k_{max} = 100$$

$$\alpha_{step} = 0.005$$

$$\epsilon_x = 1mm$$

En exécutant notre code:

Code 16: *Section du TP1.m afin de vérifier un cas pour le MGI du robot*

```
%% Q8.2 : Calculer modele geometrique inverse

% Parametres du model
X_df = [0.64 -0.1 1.14]';
q0 = [0 0.8 0 1 2 0]';
kmax = 100;
aStep = 0.005;
epsx = 0.001;
q_etoile2 = MGI(X_df, q0, kmax, epsx, aStep);
```

Nous avons obtenu le résultat suivante:

$$q_2* = \begin{bmatrix} -0.0246 & 0.7643 & -0.1782 & 1.0017 & 1.5693 & 0 \end{bmatrix}'$$

3.1.3 La vérification

Afin de pouvoir verifier que notre code soit correct, nous allons effectuer le calcul du modele geometrique directe sur les résultats que nous avons obtenu dans le point précèdent: cela nous permettra de comparer ce que la fonction *MGD* nous donne avec la valeur de départ initiale et, si la *difference* entre les deux valeurs est petite, on aura vérifie le bon fonctionnement du code.

Code 17: *Vérification du resultat pour la question 8.1, dans le fichier TP1.m*

```
% Verification du resultat 8.1
theta_etoile1 = [q_etoile1(1) q_etoile1(2) pi/2+q_etoile1(3) q_etoile1(4) q_etoile1(5)
    q_etoile1(6)];
[g_06_etoile1, g_elem_etoile1] = CalculMGD(alpha, d, theta_etoile1, r);
g_0E_etoile1 = g_06_etoile1*CalculTransformationElem(0,0,0,rE);
diff1 = abs(X_di - g_0E_etoile1(1:3,4));
```

Code 18: *Vérification du resultat pour la question 8.2, dans le fichier TP1.m*

```
% verification du resultat 8.2
theta_etoile2 = [q_etoile2(1) q_etoile2(2) pi/2+q_etoile2(3) q_etoile2(4) q_etoile2(5)
    q_etoile2(6)];
[g_06_etoile2, g_elem_etoile2] = CalculMGD(alpha, d, theta_etoile2, r);
g_0E_etoile2 = g_06_etoile2*CalculTransformationElem(0,0,0,rE);
diff2 = abs(X_df - g_0E_etoile2(1:3,4));
```

Dans le cas1 la différence que l'on obtient est:

$$|diff_1| = 10^{-7} * \begin{bmatrix} 0.0024 & 0.5096 & 0.0165 \end{bmatrix}'$$

Dans le cas2 la différence que l'on obtient est:

$$|diff_2| = 10^{-7} * \begin{bmatrix} 0.2770 & 0.4613 & 0.5327 \end{bmatrix}'$$

Nous avons pu donc vérifier que nos fonctions sont correctes étant donné que nous arrivons à avoir un différence petite entre la valeur de départe et la valeur obtenue après avoir effectué une transformation *MGI* et *MGD*.

4 Modèle cinématique inverse

4.1 Q9: Trajectoire rectiligne

La suivi de trajectoire est possible dans ce cas grâce à la cinématique inverse, car il est possible de déterminer quels sont les bons angles articulaires que le robot doit avoir pour atteindre une certaine position.

Profitant de cette méthode, on peut alors générer une loi horaire qui régit le mouvement du bras. Pour cela, nous utilisons le vecteur directeur qui indique la direction de Xd_i à Xd_f de manière discrétisée, en considérant la valeur du temps d'échantillonnage donné (T_e). De plus, pour garantir que le robot se déplace à une vitesse constante, nous mettons à jour la position souhaitée toutes les $1ms$, en tenant compte du temps total nécessaire au robot pour parcourir une trajectoire de longueur "*distance*" avec la vitesse V. La Figure 4.1 montre l'idée implémentée, et l'équation 5 montre la loi qui définit le paramétrage de la trajectoire souhaitée, où $\overrightarrow{\boldsymbol{n}}$ est un vecteur directeur normalisé, avec la direction $\overrightarrow{Xd_i, Xd_f}$.

$$\begin{aligned} Xd_x &= x_0 + \overrightarrow{\hat{n}_x} t \\ Xd_y &= y_0 + \overrightarrow{\hat{n}_y} t \\ Xd_z &= z_0 + \overrightarrow{\hat{n}_z} t \end{aligned} \tag{5}$$

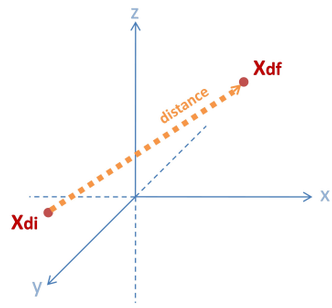

Figure 4.1: Schéma de la loi horaire proposée pour le suivi de trajectoire

Pour visualiser la trajectoire suivi par le robot afin de valider la planification de trajectoire proposée, la fonction *VisualisationChemin.m* a été créée, qui prend comme argument le vecteur de configurations articulaires que le robot a dû effectuer pour suivre la trajectoire rectiligne. Dans cette fonction auxiliaire créée, un certain nombre de configurations articulaires que le robot fait pendant son chemin sont tracées.

La Figure 4.2 montre un exemple de la représentation de la *trajectoire* suivie, en affichant 6 instants de temps différents. Sur cette figure, la configuration initiale du robot est indiquée en vert, les configurations suivants sont affichées en noir et le chemin suivi est indiqué en rose.

Code 19: *Section du TP1.m afin de détérminer la suivi de trajectoire*

```
%% Q9: Trajectoire  suivre
%Parametres donnes dans l' nonc  de la question
X_di = [-0.1 -0.7 0.3]';
X_df = [0.64 -0.1 1.14]';
V = 1;
Te = 1e-3;

% Execute la fonction de suivi de trajectoire
[Xtot, qTot] = MCI(X_di,X_df,V,Te,qi);

% Plot les rsultats obtenus
figure('Name','Question 09: Trajectoire  suivre','NumberTitle','off')
VisualisationChemin(qTot, Xtot, 6);
title('Chemin suivi (mouvement rectiligne)');
```

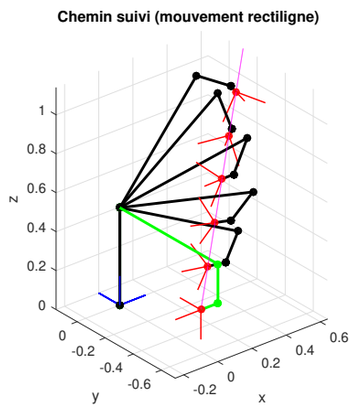

Figure 4.2: Chemin suivi par le robot pour le cas de suivi de trajectoire

Code 20: $MCI(X_{di}, X_{df}, V, T_e, q_i)$

```
function [XdTot, qTot] = MCI(Xdi, Xdf, V, Te, qi)
% Parametres
kmax = 100;
epsx = 0.005;
aStep = 0.001;

% Transformations pour l'espace discretise
distance = norm(Xdf - Xdi,2);
step = (V*Te)*(Xdf - Xdi)/distance; %Normalis (vecteur unitaire)
temps = distance/V;

% Initialisations des variables
Xd = Xdi;
XdTot = Xdi;
```

```
q = qi;
qTot = qi;
t = 0;
    % Vrification de q a chaque 1ms
    while (t < temps)
        % Mis a jour de Xd e q
        Xd = Xd + step;
        q = MGI(Xd, q, kmax, epsx, aStep);

        % Sauvegarde des positions et des configurations
        XdTot = [XdTot Xd];
        qTot = [qTot q];

        % Incrementation du temps
        t = t+1e-3;
    end
end
```

Code 21: *VisualisationChemin(q, X_{dk}, samples)*

```
function VisualisationChemin(q, Xdk, samples)
    % Dfinition des parametres
    N = size(q,2);
    plotStep = round(N/samples);

    % Affiches une <samples> quantit de configurations du robot
    for i = 1:N
        if (mod(i,plotStep)) == 1
            if i == 1
                % Affiche la premiere configurations avec la couleur verte
                VisualisationRepere(q(:,i)', 0, 'green');
                else
                VisualisationRepere(q(:,i)', 0, 'black');
            end
        end
    end
    % Plots la droite suivi
    plot3(Xdk(1,:),Xdk(2,:),Xdk(3,:),'m--', 'color', 'magenta');
end
```

Comme l'on peut le voir, le robot suit le chemin de manière satisfaisante, à partir de sa configuration initiale jusqu'à la configuration finale, tout en considérant la position souhaitée de son organe terminal.

4.2 Q10: Analyse de l'évolution des valeurs des variables articulaires

A partir de la Figure 4.3, il est possible de vérifier l'évolution des variables articulaires du robot étudié. Comme on peut l'observer, les articulations suivent une trajectoire qui respecte les conditions de continuité nécessaires pour une bonne suivi de trajectoire, c'est-à-dire que les changements d'angle appartiennent à une classe de solutions applicables au problème de suivi d'une trajectoire continue en classe *C2*.

Code 22: *Section du TP1.m afin de vérifier l'évolution des variables articulaires*

```
%% Q10
% afficher les differents courbes
qmin = [-pi -pi/2 -pi -pi -pi/2 -pi]';
qmax = [0 pi/2 0 pi/2 pi/2 pi/2]';

% Plot graphique avec l'volution de chaque variable articulaire
figure('Name','Question 10: volution des variables articulaires','NumberTitle','off')
plotEvolution(qTot, qmin, qmax)
```

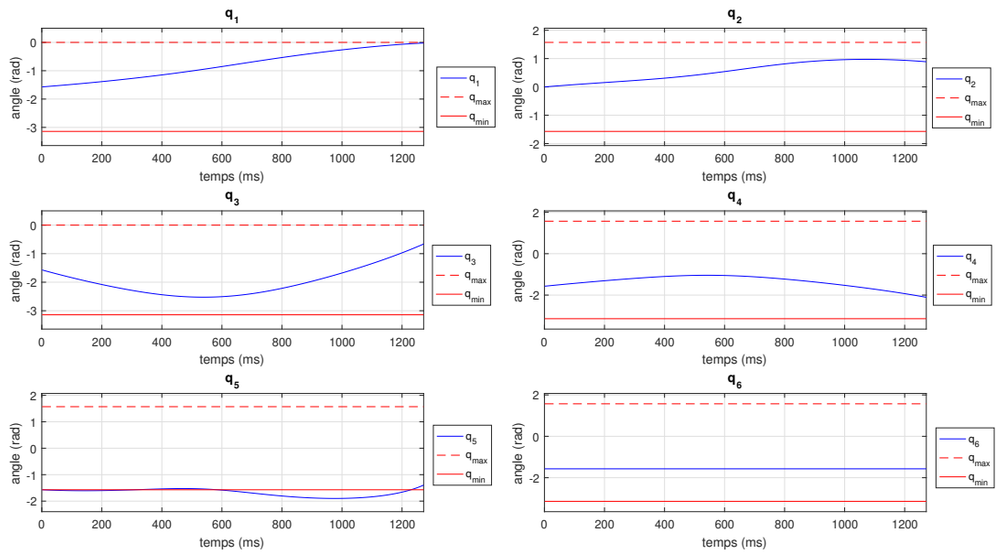

Figure 4.3: Évolution des variables articulaires pour le problème de suivi de trajectoire

Cependant, le mouvement proposé ne prend pas en compte les limites du système, telles que ses butées. Cela signifie que l'ensemble de solutions n'effectue qu'une seule *tâche principale* (qui est de se déplacer du point Xd_i vers le point Xd_f).

L'énonce nous propose d'ajouter les limites inférieure et supérieure pour la valeur de chaque variable articulaire, et il est possible de remarquer que la variable articulaire q_5 a une valeur qui est inférieure à la borne inférieure proposée. Physiquement, cela peut être considéré comme un problème pour le fonctionnement du robot, car la configuration proposée à un certain moment n'est pas réalisable avec le vrai robot, ce qui limiterait le mouvement du bras.

Dans l'exercice suivant, nous analyserons la planification de trajectoire en tenant compte des tâches secondaires (dans ce cas, ce sera la limitation des valeurs des variables articulaires.

La fonction *plotEvolution.m* ci-dessous a été utilisé pour tracer les résultats vus dans la Figure 4.3.

Code 23: *plotEvolution(q, q_{min}, q_{max})*

```
function plotEvolution(q, qmin, qmax)
    for i=1:6
        % Plot volution
        subplot(3,2,i)
        plot(q(i,:), 'blue')
        hold on

        % Plot les bornes
        plot(ones(1,size(q,2)).*qmax(i), '--', 'color', 'red')
        plot(ones(1,size(q,2)).*qmin(i), 'red')

        max_int = max(qmax(i), max(q(i,:)));
        min_int = min(qmin(i), min(q(i,:)));
        ylim([min_int-0.5 max_int+0.5])

        legend(sprintf('q_{%d}',(i)),'q_{max}', 'q_{min}');
        grid on
        xlim([0 size(q,2)]);
        title(sprintf('q_{%d}',(i)));
        xlabel('temps (ms)')
        ylabel('angle (rad)')
    end
end
```

4.3 Q11: Suivi de trajectoire avec tâche secondaire

Dans cette dernière étape du *TP1*, nous avons effectué un test pour vérifier l'utilisation du modèle cinématique inverse pour effectuer le suivi de trajectoire comme *tâche principale*, mais cette fois en tenant compte d'une *tâche secondaire*, qui est de respecter les valeurs imites de *maximum* et *minimum* de chaque variable articulaire.

Pour cela, les fonctions écrites dans le code 14 et 20 ont été modifiées comme indiqué dans les codes 25 et 26 ci-dessous. L'image 4.4 montre le résultat obtenu pour dessiner la trajectoire à suivre, tandis que l'image 4.5 montre l'évolution des variables articulaires dans le temps.

Code 24: *Section du TP1.m afin de vérifier l'évolution des variables articulaires (tâche secondaire)*

```
%% Q11: loignement des butes articulaires
```

```
% Parametres
X_di = [-0.1 -0.7 0.3]';
X_df = [0.64 -0.1 1.14]';
V = 1;
Te = 0.001;
qmin = [-pi -pi/2 -pi -pi -pi/2 -pi]';
qmax = [0 pi/2 0 pi/2 pi/2 pi/2]';

% Chemin a suivre avec constraint
[Xtot, qTot] = MCIbutees(X_di, X_df, V, Te, qi, qmin, qmax);

% Plot graphique avec l'volution de chaque variable articulaire
figure('Name','Question 11: volution des variables articulaires','NumberTitle','off')
plotEvolution(qTot, qmin, qmax)

figure('Name','Question 11: Visualisation du Chmin','NumberTitle','off')
VisualisationChemin(qTot, Xtot, 6);
title('Chemin suivi (mouvement rectiligne avec constraint)');
```

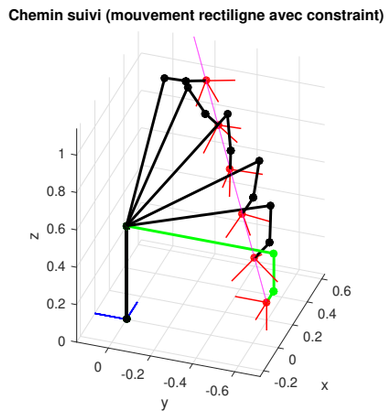

Figure 4.4: Chemin suivi par le robot pour le cas avec tâche secondaire

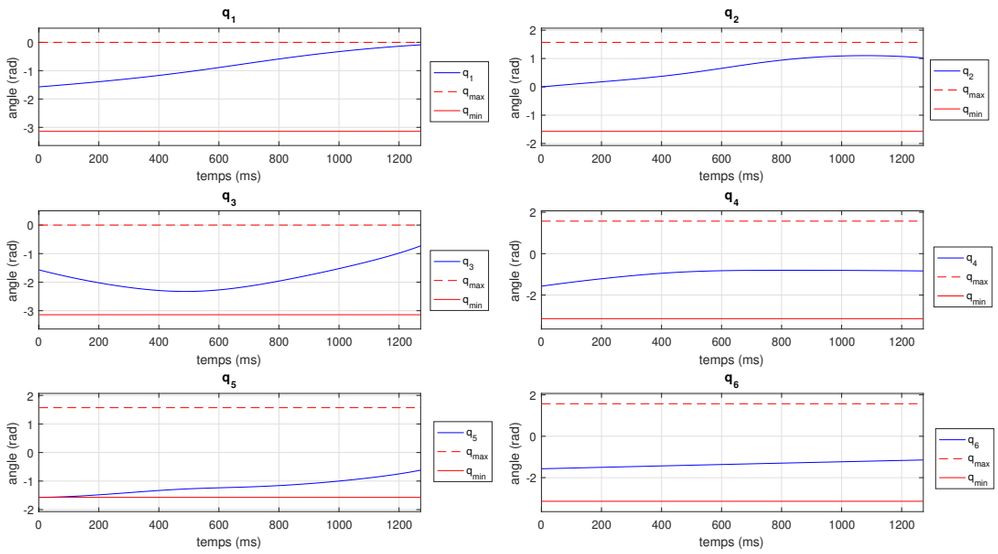

Figure 4.5: Évolution des variables articulaires pour le problème avec tâche secondaire

Comme on peut le voir sur l'image 4.5, le robot suit la trajectoire souhaitée, en partant du point Xd_i et en allant vers le point Xd_f en suivant une trajectoire rectiligne. En revanche, contrairement au résultat de la figure 4.3 de la question précédente, on peut voir sur la Figure 4.5 que toutes les variables articulaires se trouvent maintenant dans les intervalles délimités par les vecteurs $\boldsymbol{q\_max}$ et $\boldsymbol{q\_min}$ donnés dans l'énonce du cours. Avec cela, nous pouvons vérifier le bon fonctionnement de la fonction créée.

Comme on pourrait voir dans le code suivant, ce qu'on a fait a été de récupérer les fonctions *MCI* et *MGI* implémentées dans les questions précédentes et, dans *MCIbutees* nous avons substitué l'appel à la fonction *MGI* avec l'appel à une nouvelle fonction *MGIb*.
Dans cette fonction nous avons substitué la *méthode de Newton-Raphson* avec la technique du *gradient*

projeté dans le noyau de ${}^0J_v(q)$ et nous avons considéré la minimisation de cette fonction potentielle:

$$H_{but} = \sum \left(\frac{q_i - \bar{q}_i}{q_{max} - q_{min}} \right)^2$$

En particulier nous allons utiliser le gradient de la fonction potentielle en implémentant la fonction d'optimisation suivante:

$$\dot{q}* = J^{\#}\dot{X}_d + (I_n - J^{\#}J)(-\alpha_H \nabla H_{but})$$

Code 25: *MCIbutees*($X_{di}, X_{df}, V, T_e, q_i, q_{min}, q_{max}$)

```
function [XdTotbutee, qTotbutee] = MCIbutees(Xdi, Xdf, V, Te, qi, qmin, qmax)

    % Parametres
    kmax = 100;
    epsx = 0.005;
    aStep = 0.001;
    aH = 0.01;

    % Transformations pour l'espace discretise
    distance = norm(Xdf - Xdi,2);
    step = (V*Te)*(Xdf - Xdi)/distance; %Normalis (vecteur unitaire)
    temps = distance/V;

    % Initialisations des variables
    Xd = Xdi;
    XdTotbutee = Xdi;
    q = qi;
    qTotbutee = qi;
    t = 0;
        % Vrification de q a chaque 1ms
        while (t < temps)
            % mise a jour de la position
            Xd = Xd + step;

            %calcul de la configuration lie a la position courante
            q = MGIb(Xd, q, kmax, epsx, aH, qmin, qmax);

            % sauvegarde des positions et des configurations
            XdTotbutee = [XdTotbutee Xd];
            qTotbutee = [qTotbutee q];

            % Incrementation du temps
            t = t+1e-3;
        end
end
```

Code 26: *MGIb*($Xd,\ q,\ k_{max}, epsx, a_H, q_{min}, q_{max}$)

```
function q_etoile = MGIb(Xd, q, kmax, epsx, aH, qmin, qmax)
    % Parametres
    rE = 0.1;
    r4 = 0.2;
    r1 = 0.5;
    r = [r1 0 0 r4 0 0];
    d3 = 0.7;
    d = [0 0 d3 0 0 0];
    alpha = [0 pi/2 0 pi/2 -pi/2 pi/2];
    theta = [q(1) q(2) pi/2+q(3) q(4) q(5) q(6)];
    k = 0;

    while true
        k = k+1;

        J = CalculJacobienne(alpha, d, theta, r);
        Ji = J(1:3, :);
        [g_06, ~] = CalculMGD(alpha, d, theta, r);
        g_0E = g_06*CalculTransformationElem(0,0,0,rE);
        X = g_0E(1:3, 4);

        dX = Xd - X;

        % Calcul de la fonction objectif
        qbar = (qmax-qmin)/2;

        H = [];
        for i = 1:size(q,1)
            H = [H; 2*(q(i)-qbar(i))/(qmax(i)-qmin(i))^2];
```

```
        end

        % Solution de l'optimisation quadratique
        q = q + (pinv(Ji) * dX) + (eye(size(Ji,2))-(pinv(Ji)*Ji))*(-aH * H')';
        error = norm(dX, 2);

        % Condition de fin de la boucle
        if (error < epsx) || (k > kmax)
            break
        end
    end
    q_etoile = q;
end
```

TP 2

5 Modèle dynamique

Le modèle dynamique consiste à établir les relations entre les efforts des actionneurs et les mouvements qui en dérivent. Ce que ce modèle nous ramène à faire c'est d'expliciter les équations différentielles du deuxième ordre afin de construire les équations du mouvement.

Dans le cas de notre modele, nous allons écrire le modele dynamique comme:

$$A(q) * \ddot{q} + C(q, \dot{q}) * \dot{q} + G(q) + \Gamma_f(\dot{q}) = \Gamma$$

où:

- $A(q)$ est la matrice d'*inertie* que nous allons calculer dans la question 13.
- $C(q, \dot{q})$ est le vecteur des couples *articulaires* dus aux forces de *Coriolis* et centrifuge que nous allons calculer dans la question 17.
- $G(q)$ qui est le vecteur des couples *articulaires* de gravité.
- $\Gamma_f(\dot{q})$ qui est le vecteur des couples de *frottement* que nous allons calculer dans la question 17.

5.1 Q12: Les matrices jacobiennes ${}^0J_{v_{Gi}}$ et ${}^0J_{\omega_i}$ et les vitesses

Dans cette question il nous est demandé de calculer la vitesse ${}^0V_{v_{Gi}}$ du centre de masse de notre système et la vitesse de rotation ${}^0\omega_i$ de chaque corps du système.

Pour faire cela il nous est suggéré d'implémenter une fonction $CalculMatriceJacobienneGi(alpha, d, theta, r, x_G, y_G, z_G)$ afin de calculer les matrices jacobiennes ${}^0J_{v_{Gi}}$ et ${}^0J_{\omega_i}$. Les vecteurs x_G, y_G et z_G contiennent les coordonnées des différents centres de masse des corps.

Les valeurs de vitesse et de vitesse rotation seront en suite calculés avec les formules suivantes:

$${}^0V_{v_{Gi}} = {}^0 J_{v_{Gi}}(q) * \dot{q}$$

$${}^0\omega_i = {}^0 J_{\omega_i}(q) * \dot{q}$$

Nous allons faire des itérations (une pour chaque corps du système) que nous permettent de calculer les relatives jacobiennes. Nous allons utiliser la *formule de Varignon* afin de modifier le repere de référence de $\mathscr{R}_O$ à $\mathscr{R}_{G_i}$.

Nous nous sommes basé sur la fonction montré en classe que nous avons reporté ci-dessous:

Code 27: *CalculMatriceJacobienneGi(α, d, θ, r, x_G, y_G, z_G)*

```
function [OJv_Gi, OJ_wi] = CalculMatriceJacobienneGi(alpha, d, theta, r, x_G, y_G, z_G)

    %Initialisation
    J = CalculJacobienne(alpha, d, theta, r); %Matrice jacobienne du robot

    OJ_Oi = zeros(size(J,1), size(J,2), size(J,2));
    OJ_Gi = zeros(size(J,1), size(J,2), size(J,2));
    OJv_Gi = zeros(3, size(J,2), size(J,2));
    OJ_wi = zeros(3, size(J,2), size(J,2));
    g_elem = zeros(4,4);
    ROi = zeros(3,3,size(J,2));
    g_0i = eye(4);

    % Vecteur de position de l'organe effecteur dans R0:

    [g_06, g_elem] = CalculMGD(alpha, d, theta, r);
    rE = 0.1;
    g_6E = CalculTransformationElem(0,0,0,rE);
    g_0E = g_06 *g_6E;
    O0OE = g_0E(1:3, 4);
    OEO0 = -O0OE;

    O0Oi = zeros(3,1,6);

    %Boucle sur les corps
    for i = 1:size(J,2)
        %Construction de toutes les matrices jacobiennes des corps Ci
        %exprimee au centre Oi du repere Ri attache a Ci
        OJ_Oi(:,1:i,i) = J(:,1:i);
        %Vecteur OiGi exprim dans Ri
        iOiGi = [x_G(i) y_G(i) z_G(i)]';
```

```
        %Matrice de rotation ROi du repre R0  Ri
        g_elem = CalculTransformationElem(alpha(i), d(i), theta(i), r(i));
        %Matrice de transformation lmentaire de passage de C(i-1)
        g_0i = g_0i*g_elem;
        ROi(:,:,i) = g_0i(1:3,1:3);
        %Vecteur OEGi dans R0  calculer: OEGi = OEOi + OiGi
        %Vecteur OiGi exprim dans R0
        OOiGi = ROi(:,:,i)*iOiGi;
        %Vecteur OEOi exprim dans R0: OEOi = OEO0 + OO0i
        OO0i(:,:,i) = g_0i(1:3,4);
        OEOi = OEO0 + OO0i(:,:,i);
        OOEGi = OEOi + OOiGi;
        OPreproduitVect_OEGi = [0 -OOEGi(3) OOEGi(2);...
                                OOEGi(3) 0 -OOEGi(1);...
                                -OOEGi(2) OOEGi(1) 0];
        %Formule de Varignon
        OJ_Gi(:,:,i) = [eye(3) -OPreproduitVect_OEGi;...
                        zeros(3,3) eye(3)]*OJ_Oi(:,:,i);

        %Resultats
        OJv_Gi(:,:,i) = OJ_Gi(1:3,:,i);
        OJ_wi(:,:,i) = OJ_Gi(4:6,:,i);
    end
end
```

5.2 Le calcul de vitesses pour $\dot{q} = [-0.5, 1, -0.5, 0.5, 1, -0.5]$

En appliquant les formules que nous avons reporté ci-dessus nous avons pu calculer les vitesses:

Code 28: *Section du fichier TP2.m pour le calcul des vitesses*

```
%% Q12 - MatriceJacobienneGi
[OJv_Gi, OJ_wi] = CalculMatriceJacobienneGi(alpha, d, theta, r, x_G, y_G, z_G);
OVGi = [];
OwGi = [];

for i = 1:6
OVGi = [OVGi OJv_Gi(:,:,i) * dq];
OwGi = [OwGi OJ_wi(:,:,i) * dq];
end

% Les vitesses
OVGi
OwGi
```

Nous avons obtenu les matrices suivantes qui contiennent les vitesses des centres de masse des différents articulations (pour chaque colonne nous avons un vecteur vitesse correspondant):

$$ {}^0V_{v_{Gi}} = \begin{bmatrix} 0 & 0.175 & 0.35 & 0.35 & 0.35 & 0.35 \\ 0 & 0 & -0.05 & -0.1 & -0.1 & -0.1 \\ 0 & 0.35 & 0.7 & 0.7 & 0.7 & 0.7 \end{bmatrix} $$

$$ {}^0\omega_i = \begin{bmatrix} 0 & -1 & -0.5 & -0.5 & -0.5 & 0 \\ 0 & -0 & -0 & -0 & -1 & -1 \\ 0.5 & 0.5 & 0.5 & 0 & 0 & 0 \end{bmatrix} $$

5.3 Q13: La matrice d'inertie A

Dans cette question nous allons proposer la fonction qui calculera la matrice d'inertie A du système. Cette matrice contient les moments d'inertie du robot par rapport au trois plans x, y, z et les produits d'inertie par rapport au trois plans yz, xz et xy.

Afin de faire cela nous avons considère les valeurs des *tenseurs d'inertie* des corps données dans les respectifs repères et nous avons dû les ramener dans le repere d'origine. Afin de faire cela nous avons utilisé le *Théorème de Huygens généralisé* et, ensuite, nous sommes retournés dans le repere d'origine en utilisant la matrice de rotation. Enfin, nous avons utilisé l'équation suivante pour le calcul de la matrice d'inertie:

$$ A = \sum \left(m_i^0 J_{v_{Gi}}^T(q) {}^0J_{v_{Gi}}(q) + {}^0\ J_{\omega_i}^T(q) {}^0I_i^0 J_{\omega_i}(q) \right) $$

Ci-dessous nous avons reporté la fonction que nous avons implémenté:

Code 29: *CalculMatriceInertie(q)*

```
function A = CalculMatriceInertie(q)
    %Calcul des parametres
    m=[15 10 1 7 1 0.5];
```

```
    x_G1 = 0; y_G1 = 0; z_G1 = -0.25;
    x_G2 = 0.35; y_G2 = 0; z_G2 = 0;
    x_G3 = 0; y_G3 = -0.1; z_G3 = 0;
    x_G4 = 0; y_G4 = 0; z_G4 = 0;
    x_G5 = 0; y_G5 = 0; z_G5 = 0;
    x_G6 = 0; y_G6 = 0; z_G6 = 0;
    x_G = [x_G1 x_G2 x_G3 x_G4 x_G5 x_G6]';
    y_G = [y_G1 y_G2 y_G3 y_G4 y_G5 y_G6]';
    z_G = [z_G1 z_G2 z_G3 z_G4 z_G5 z_G6]';

    iI(:,:,1)=[0.80 0 0.05 ; 0 0.80 0 ; 0.05 0 0.10];
    iI(:,:,2)=[0.10 0 0.10; 0 1.50 0 ; 0.10 0 1.50];
    iI(:,:,3)=[0.05 0 0 ; 0 0.01 0 ; 0 0 0.05];
    iI(:,:,4)=[0.50 0 0 ; 0 0.50 0 ; 0 0 0.05];
    iI(:,:,5)=[0.01 0 0 ; 0 0.01 0 ; 0 0 0.01];
    iI(:,:,6)=[0.01 0 0 ; 0 0.01 0 ; 0 0 0.01];

    d = [0 0 0.7 0 0 0];
    r = [0.5 0 0 0.2 0 0];
    alpha = [0 pi/2 0 pi/2 -pi/2 pi/2];
    theta = [q(1) q(2) q(3)+pi/2 q(4) q(5) q(6)]';

    Rred = [100 100 100 70 70 70];
    Jm = 1e-5*[1 1 1 1 1 1];

    %initialisation des tableaux
    g_0i = eye(4);
    Ig = zeros(3*size(q,1),3);
    A = zeros(size(q,1),size(q,1));

    %Calcul des matrices jacobiennes de chacun corps
    [OJv_Gi, OJ_wi] = CalculMatriceJacobienneGi(alpha, d, theta, r, x_G, y_G, z_G);

    for i = 1 : length(q)
        %  Thorme   de Huygens generalis
        Huygens = [((y_G(i))^2+(z_G(i))^2) -(x_G(i)*y_G(i)) -(x_G(i)*z_G(i));...
               -(x_G(i)*y_G(i)) ((x_G(i))^2+(z_G(i))^2) -(y_G(i)*z_G(i));...
               -(x_G(i)*z_G(i)) -(y_G(i)*z_G(i)) ((x_G(i))^2+(y_G(i))^2);];

        %Transport du tenseur d'inertie dans le repre R0
        g_elem = CalculTransformationElem(alpha(i),d(i),theta(i),r(i));
        g_0i = g_0i*g_elem;
        R0i = g_0i(1:3,1:3);

        % Application theoreme de Huygens generalis
        iI_Gi = iI(:,:,i) - m(i) * Huygens;

        % Retour dans le repere qui nous interesse
        0I_Gi = R0i * iI_Gi * R0i';

        % Calcul matrice d'inertie du manipulateur
        A = A + ((m(i)*OJv_Gi(:,:,i)')*OJv_Gi(:,:,i) + (OJ_wi(:,:,i)')*0I_Gi*OJ_wi(:,:,i));
    end

    %Ajout des contributions des inerties des actionneurs
    A = A + diag(Rred.^2.*Jm);

end
```

Enfin, nous avons appelé cette fonction pour tester son fonctionnement et la valeur de la matrice A (pour q $= [-\pi/2, 0, -\pi/2, -\pi/2, -\pi/2, -\pi/2]'$) que nous avons obtenu est:

$$A = \begin{bmatrix} 6.435 & 0 & 0 & -0.07 & 0 & 0 \\ 0 & 7.165 & 0.91 & 0 & 0 & 0.01 \\ 0 & 0.91 & 1.01 & 0 & 0 & 0.01 \\ -0.07 & 0 & 0 & 0.119 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0.069 & 0 \\ 0 & 0.01 & 0.01 & 0 & 0 & 0.059 \end{bmatrix}$$

5.4 Q14: Les bornes de la matrice d'inertie A

Une des propriétés de la matrice d'inertie est qu'elle est bornée inférieurement et supérieurement. Nous allons étudier les deux scalaires positifs qui représentent la borne supérieure μ_2 et la borne inférieure μ_1 pour la matrice d'inertie.

Afin de faire cela nous allons faire un certain nombre d'itérations (1000 dans notre cas) et, à chaque itération, nous allons calculer la matrice d'inertie pour une nouvelle valeur de q (comprise entre q_{max} et

q_{min}).

Ci-dessous nous avons reporté l'algorithme et la relative fonction implémentés:

Code 30: *Section du TP2.m afin de calculer la borne inférieure et supérieure de la matrice d'inertie*

```
%% Q14: Borne de la Matrice A
% butees articulaires qmin et qmax dfinies a la question Q10
qmin = [-pi -pi/2 -pi -pi -pi/2 -pi]';
qmax = [0 pi/2 0 pi/2 pi/2 pi/2]';

% Calcul des bornes pour la matrice A
[mu1,mu2] = FindBorneA(A, qmin, qmax, 1000)
```

Code 31: *FindBorneA(A,qmin,qmax,1000)*

```
function [mu1, mu2] = FindBorneA(A, qmin, qmax, NumTry)
    % Initialisations des bornes de la matrice A
    mu1 = inf;
    mu2 = 0;

    % Nous allons faire varier les q entre qmin et qmax
    dq = (qmax-qmin)/NumTry;
    qCur = qmin;

    for i = 1 : NumTry
        qCur = qCur + dq;
        A = CalculMatriceInertie(qCur);
        maxCur = max(eig(A));
        minCur = min(eig(A));
        if maxCur > mu2
            mu2 = maxCur;
        end
        if minCur < mu1
            mu1 = minCur;
        end
    end
end
```

Les valeurs obtenues pour notre problème sont:

- $\mu_1 = 0.0574$
- $\mu_2 = 10.1985$

5.5 Q15: Le vecteur des couples articulaires de gravité G

Dans cette section nous allons implémenter la fonction que nous permet de obtenir le vecteur des couples articulaires de gravité $G(q)$.

Afin de faire cela il nous est demandé d'utiliser la formulation analytique du gradient de l'énergie potentielle $E_p(q) = g^t(\sum m_i^0 p_{Gi}(q))$ qui, en définissant le vecteur g comme le vecteur de gravité, nous fait obtenir:

$$G(q) = -\left({}^0J_{vG1}^t m_1 g + ... + {}^0J_{vG6}^t m_6 g\right)$$

La fonction implémentée a été reporté ci-dessous:

Code 32: *CalculCoupleGravite(q)*

```
function G = CalculCoupleGravite(q)

    %Parametres
    m=[15 10 1 7 1 0.5];
    x_G1 = 0; y_G1 = 0; z_G1 = -0.25;
    x_G2 = 0.35; y_G2 = 0; z_G2 = 0;
    x_G3 = 0; y_G3 = -0.1; z_G3 = 0;
    x_G4 = 0; y_G4 = 0; z_G4 = 0;
    x_G5 = 0; y_G5 = 0; z_G5 = 0;
    x_G6 = 0; y_G6 = 0; z_G6 = 0;
    x_G = [x_G1 x_G2 x_G3 x_G4 x_G5 x_G6];
    y_G = [y_G1 y_G2 y_G3 y_G4 y_G5 y_G6];
    z_G = [z_G1 z_G2 z_G3 z_G4 z_G5 z_G6];

    d = [0 0 0.7 0 0 0];
    r = [0.5 0 0 0.2 0 0];
    alpha = [0 pi/2 0 pi/2 -pi/2 pi/2];
    theta = [q(1) q(2) q(3)+pi/2 q(4) q(5) q(6)];

    % Calcul des matrices jacobiennes de chacun des corps
```

```
    [OJv_Gi, ~] = CalculMatriceJacobienneGi(alpha, d, theta, r, x_G, y_G, z_G);

    %Vecteur gravit  exprim  dans R0
    g = [0 0 -9.81]';

    %Calcul du couple de gravit produit par les corps
    G = zeros(length(theta), 1);

    for i = 1:length(m)
        G = G - OJv_Gi(:,:,i)'*m(i)*g;
    end
end
```

La valeur du vecteur G obtenu est:

$$G = \begin{bmatrix} 0 & 99.5715 & 0 & 0 & 0 & 0 \end{bmatrix}'$$

5.6 Q16: Le majorant du vecteur des couples g_b

Aussi pour le vecteur des couples articulaires induits par la gravité $G(q)$ il existe une propriété qui dit que sa valeur est borné.
Dans cette section il nous est demandé d'implémenter un algorithme afin de trouver le majorant g_b en norme 1 du vecteur de couple articulaires de gravité $||G(q)||_1$.

Le code que nous avons implémenté est similaire à lequel pour la question Q14 et nous l'avons reporté ci-dessous:

Code 33: *Section du main afin de calculer le majorant du vecteur de couple articulaires de gravité* $||G(q)||_1$

```
%% Q16: Calcule d'un majorant pour G

% butes  articulaires qmin et qmax dfinies  la question Q10
qmin = [-pi -pi/2 -pi -pi -pi/2 -pi]';
qmax = [0 pi/2 0 pi/2 pi/2 pi/2]';

% Calcul du majorant pour de G
maxG = FindMajorantG (G, qmin,qmax, 1000)
```

Code 34: *FindMajorantG(G,qmin,qmax,NumTry)*

```
function [maxG] = FindMajorantG (G, qmin,qmax, NumTry)
    maxG = 0;

    % Nous allons faire varier les q entre qmin et qmax
    dq = (qmax-qmin)/NumTry;
    qCur = qmin;
    for i = 1 : NumTry
        qCur = qCur + dq;
        G = CalculCoupleGravite(qCur);
        curG = norm(G,1);
        if curG > maxG
            maxG = curG;
        end
    end
end
```

La valeur obtenu dans le cas de notre problème est:

$$g_b = 117.3237$$

5.7 Q17: Le vecteur des couples articulaires produit par les forces de frottement Γ_f et l'implementation du modèle du robot en *Simulink*

Dans cette section nous allons proposer un bloc de simulation qui permet de programmer le modèle dynamique *direct* du système à partir des fonctions calculées précédemment, de la fonction pour le calcul de la couple due au frottement et de la fonction pour le calcul de la couple due à la force de *Coriolis* et à la force *Centrifuge*.

Nous allons reporter ci-dessous seulement la fonction pour le calcul de la *couple due au frottement* car la deuxième fonction était déjà donnée implémentée. Pour cela nous allons nous baser sur le modèle de *frottement* proposé dans le *TP*:

$$\tau_{fi}(\dot{q}_i) = diag(\dot{q}_i) * F_{vi}$$

Code 35: *CalculCoupleFrottement($\dot{q}$)*

```
function Gamma = CalculCoupleFrottement(dq_dt)
```

```
    %parametres
    Fv = 10*ones(size(dq_dt),1);

    %Fonction donne    l'interieure du TP
    Gamma = diag(dq_dt)*Fv;
end
```

Ensuite, nous avons implémenté les blocs afin d'implémenter le modèle de notre robot sur *Simulink* et nous avons obtenu:

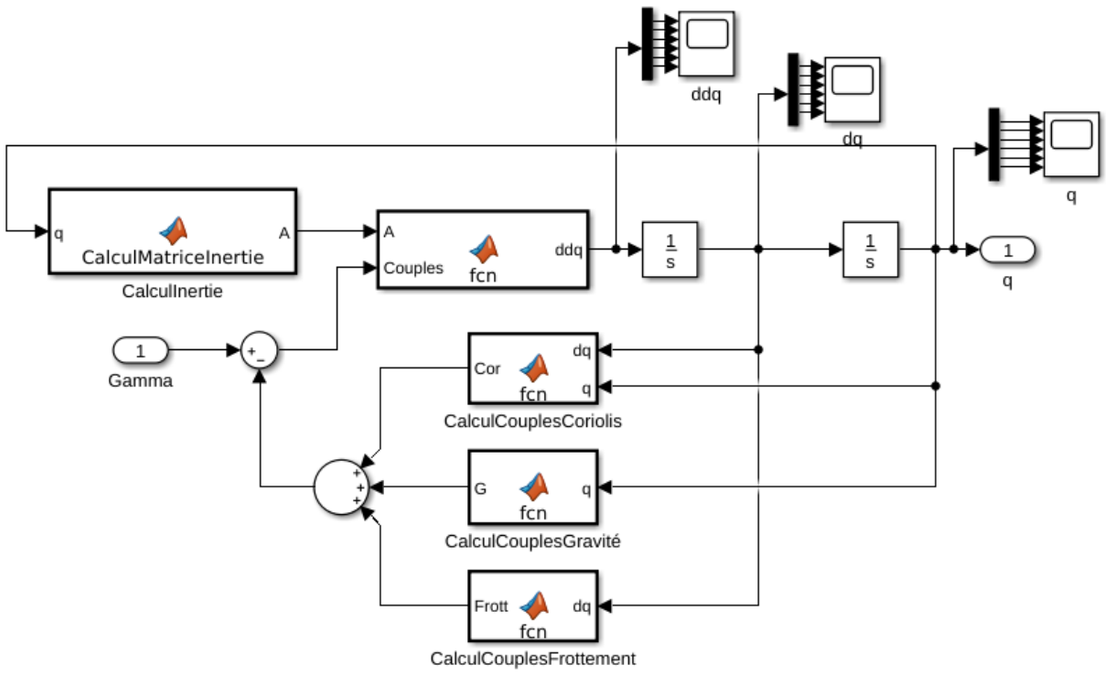

Figure 5.1: Modèle du robot dans *Simulink*

qui va traduire en un schéma à blocs l'équation de mouvement que nous avons reporté au début de TP.

En particulier ce que on a fait c'était de extraire $\ddot{q}$ et l'exprimer en fonction du reste des paramètres en obtenant:

$$\ddot{q} = A^{-1}(\Gamma - (\Gamma_f(\dot{q}) + G(q) + C(q,\dot{q})\dot{q}))$$

Enfin, il a été possible d'obtenir les valeurs de q, $\dot{q}$ en utilisant les blocs *intégrateurs* de *Simulink* et les faire re-boucler afin d'obtenir les *Couples* désirées.

C'est important de noter que la fonction *CalculCoupleCoriolisCentrifuge(q,q̇)* donne déjà la quantité $C(q,\dot{q})\dot{q}$.

6 Génération de trajectoire articulaire

6.1 Q18: Le temps minimal pour les différents articulations

Nous devons générer une trajectoire polynomiale de degré 5 à suivre dans l'espace articulaire.

Nous avons les suivants conditions aux limites:

- $q_{di} = \begin{bmatrix}-1 & 0 & -1 & -1 & -1 & -1\end{bmatrix}$'rad
- $q_{df} = \begin{bmatrix}0 & 1 & 0 & 0 & 0 & 0\end{bmatrix}$'rad
- les vitesses et les accélérations initiales et finales doivent être nulles $\dot{q}(0) = 0$, $\dot{q}(t_f) = 0$, $\ddot{q}(0) = 0$, $\ddot{q}(t_f) = 0$
- utiliser une période d'échantillonage $T_e = 1$ ms

et nous devons calculer le temps minimal pour chaque articulation en tenant en compte seulement du vecteur k_a(qui dépend du rapport des couples moteurs maximaux, des rapport de réduction et des inerties maximales vu par les articulations).

Le temps pour chaque articulation est donné par la formule suivante:

$$t_{fi} = max\left(\frac{15|D_i|}{8k_{vi}}, \sqrt{\frac{10|D_i|}{\sqrt{3}k_{ai}}}\right)$$

où $D = q_{max} - q_{min}$ et étant donnée que nous allons considérer seulement les termes k_{ai} nous aurons que:

$$t_{fi} = \sqrt{\frac{10|D_i|}{\sqrt{3}k_{ai}}}$$

comme nous avons implementé ci-dessous:

Code 36: *Section du fichier TP2.m pour le calcul des temps*

```
%% Q18: valuation des paramtres de la generation de la trajectoire
% Configurations articulaires dsires
qdi = [-1 0 -1 -1 -1 -1]';
qdf = [0 1 0 0 0 0]';

% Parametres
Te = 0.001;
Rred = [100 100 100 70 70 70];

% Couples moteurs maximale
tau = 5;

% Vecteur des accelerations articulaires maximales
ka = ((Rred * tau) / mu2)';
D = abs(qdf - qdi);
% Calcul des temps
tf = (sqrt((10/sqrt(3))*D./ka))
tf = max(tf)
```

Le vecteur des valeur du temps maximales pour chaque articulation obtenue est le suivant:

$$tf = \begin{bmatrix} 0.3432 & 0.3432 & 0.3432 & 0.4102 & 0.4102 & 0.4102 \end{bmatrix}' sec$$

6.2 Q19: L'évolution temporelle des trajectoires articulaires q_{c_i}

Dans cette question nous allons étudier le comportement des valeurs prises par q parmi le point de départ q_i et le point d'arrivée q_f.

Nous avons utilisé la formule suivante afin de obtenir l'évolution temporelle de q:

$$q(t) = q_0 + r(t) * D$$

. où la loi horaire $r(t)$ est donnée par:

$$r(t) = 10\left(\frac{t}{t_f}\right)^3 - 15\left(\frac{t}{t_f}\right)^4 + 6\left(\frac{t}{t_f}\right)^5$$

Nous avons implémenté la fonction *GeneTraj(q_{di}, q_{df}, t)* et nous l'avons reportée ci-dessous:

Code 37: *GeneTraj*(q_{di}, q_{df}, t)

```
function qc = GeneTraj(q_di, q_df, t)

    tf = 500; %in ms

    D = (q_df - q_di);

    r = 10 * (t/tf)^3 - 15 * (t/tf)^4 + 6 * (t/tf)^5;

    qc = q_di + r*D;
end
```

Nous avons créé un modèle *Simulink* que nous avons reporté ci-dessous:

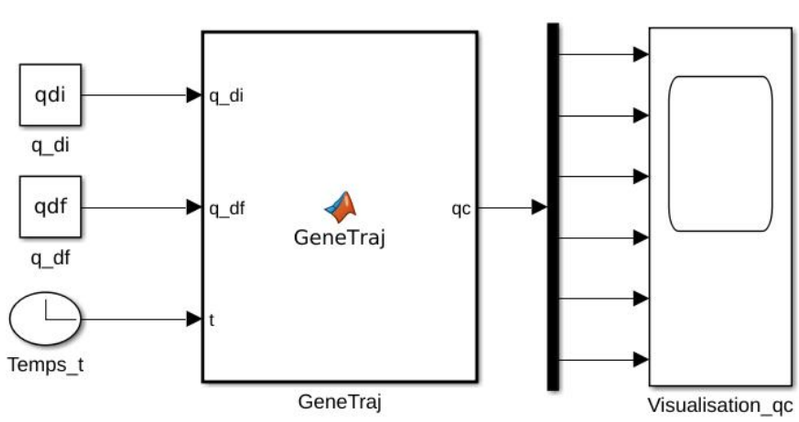

Figure 6.1: Schéma *Simulink* pour la génération de trajectoire

Ensuite nous avons effectué la simulation et étudié l'évolution temporelle des trajectoires articulaires pour les différents corps pendant la période de temps entre $0ms$ et $500ms$.

Il est possible de voir que les articulations commencent avec une vitesse nulle jusqu'à arriver à avoir la vitesse maximale au milieu du graphique et recommencer à ralentir pour arriver dans la position finale avec une vitesse encore nulle.

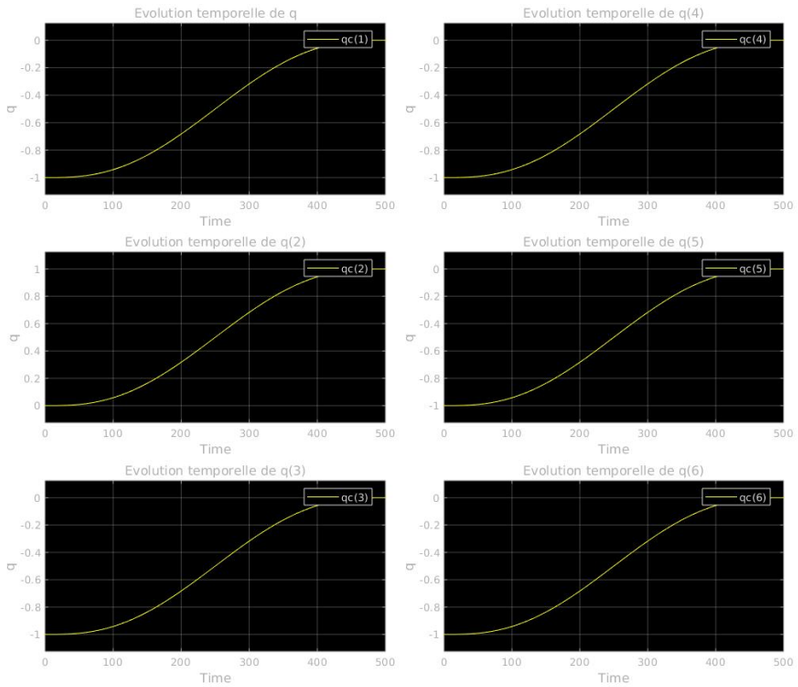

Figure 6.2: Évolution temporelle des trajectoires articulaires

7 Commande dans l'espace articulaire

7.1 Q20: L'ajout d'un bloc de commande

Enfin, en partant du schéma développé dans la question 19, nous avons inséré le modèle du robot développé dans la question 18 et la commande que nous allons développer dans la suite. Nous avons reporté ci-dessous le schéma:

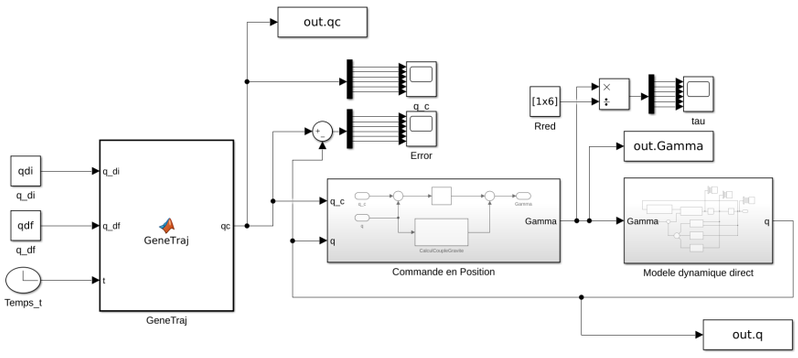

Figure 7.1: Schéma *Simulink* pour le contrôle

et à l'intérieur du bloc de contrôle nous avons inséré les blocs suivants:

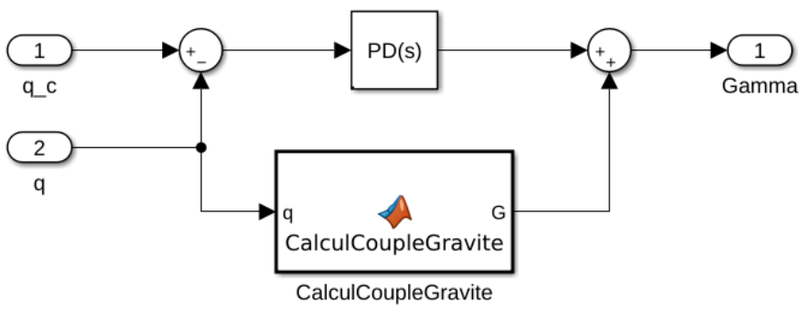

Figure 7.2: Schéma de contrôle

afin d'implémenter la commande articulaire *P.D.* reportée dans la consigne:

$$\Gamma = K_p * (q_d - q) + K_d(\dot{q}_d - \dot{q}) + \hat{G}(q)$$

En particulier, nous allons insérer une fonction *Matlab* afin de calculer la couple de gravité et le contrôleur *PD* pour lequel on devra ensuite choisir les deux coefficients de gain: **coefficient proportionnel** et **coefficient dérivative**.

Nous avons choisi les valeurs suivantes:

- $K_p = 1$
- $K_d = 0.4$

que nous permettent d'avoir les comportements suivants, en fonction du temps (en *milliseconds*), pour la valeur de q, $\dot{q}$ et $\ddot{q}$:

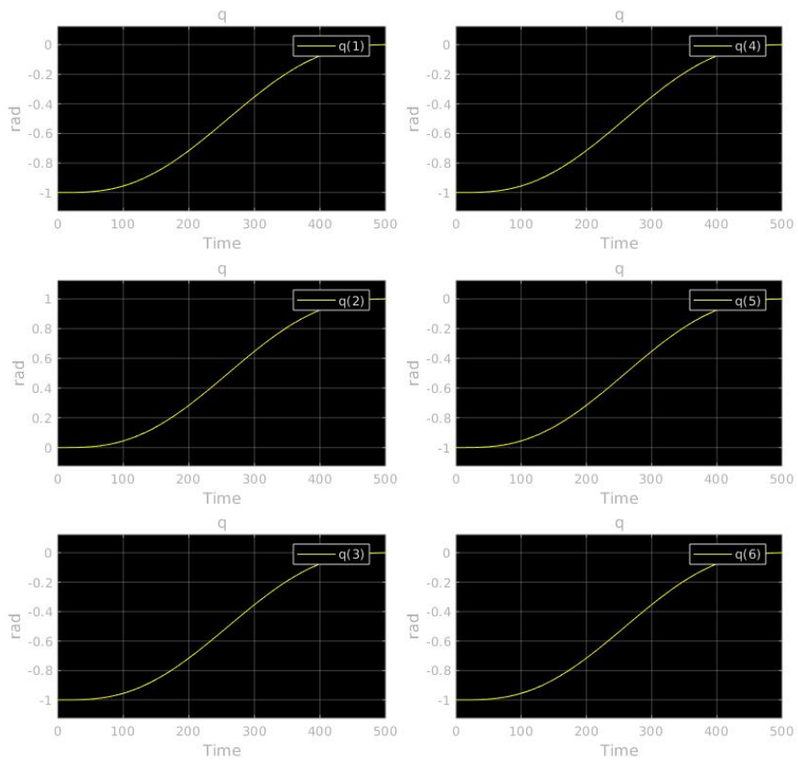

Figure 7.3: Comportement de q

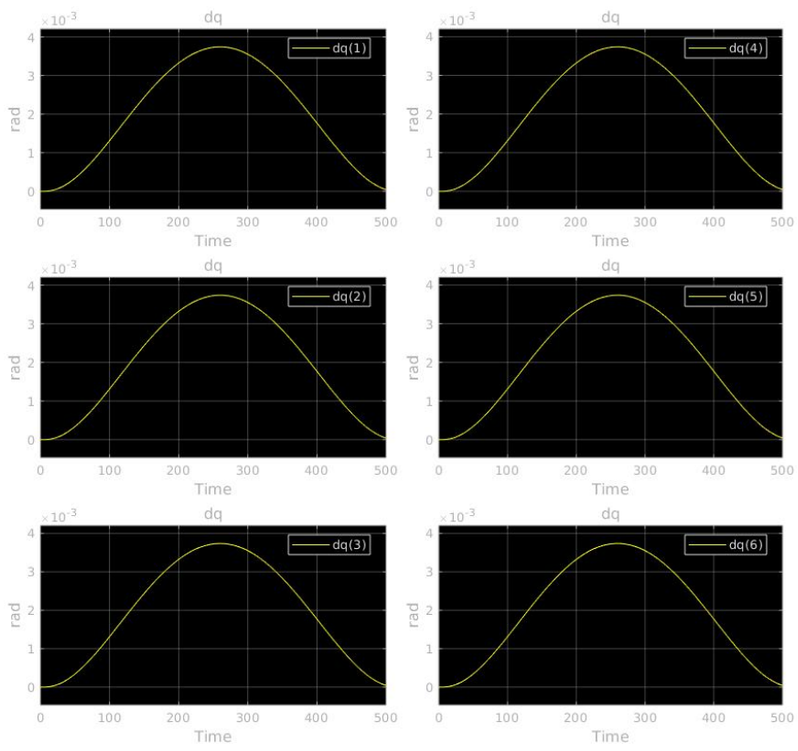

Figure 7.4: Comportement de $\dot{q}$

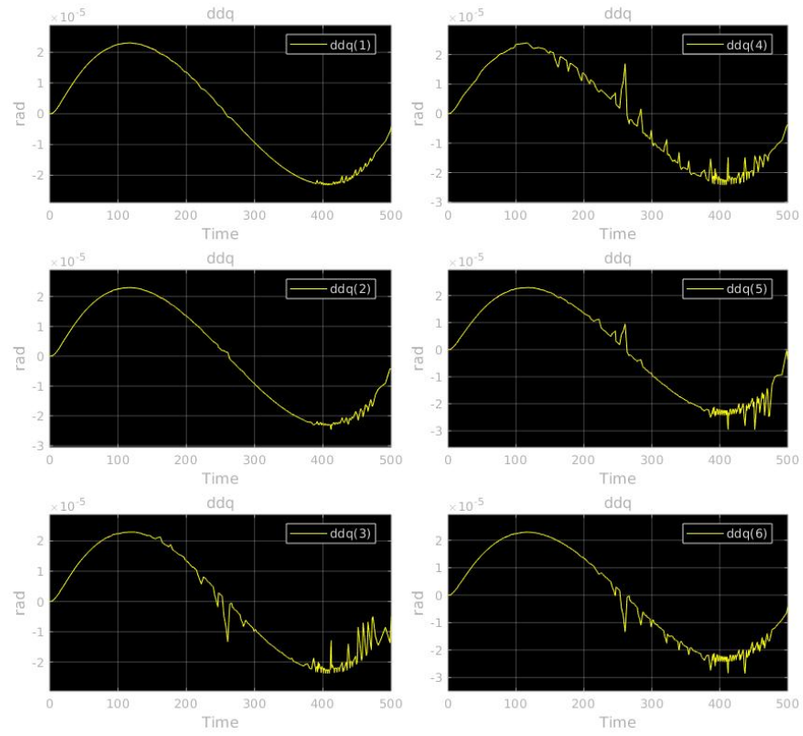

Figure 7.5: Comportement de $\ddot{q}$

Nous avons aussi reporté l'évolution de la valeur de l'erreur entre q_c et q qui, comme demandé dans la consigne, reste toujours plus petite de $0.05rad$.

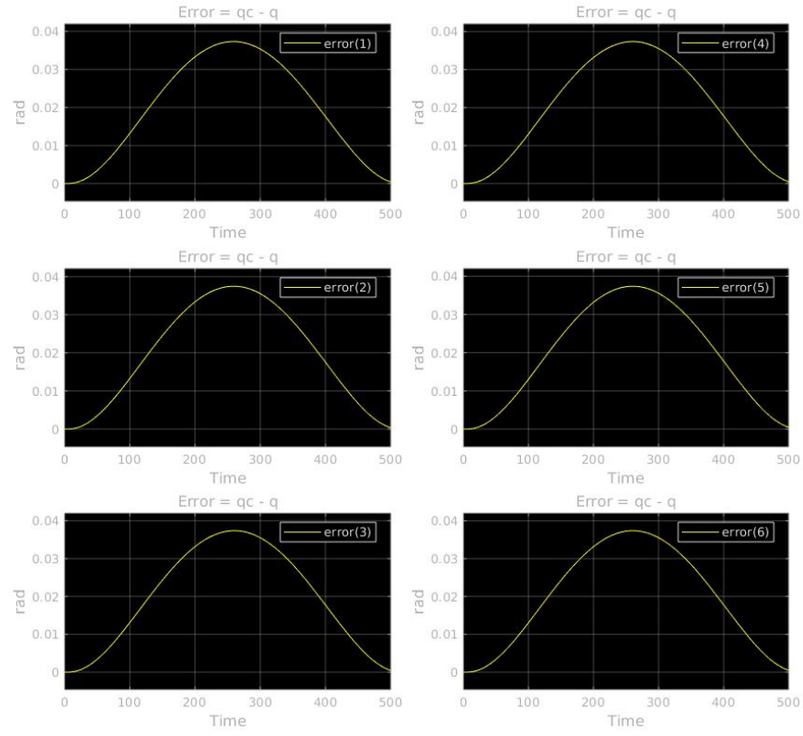

Figure 7.6: Évolution de l'erreur entre q_c et q

Comme dernière chose, nous avons reporté l'évolution temporelle des couples articulaires de commande $\tau_i(t)$ correspondante au réglage des coefficients de gain afin de vérifier que ses valeurs soient plus petites de la couple maximale des moteurs qui vaut $5N * m$. Pour obtenir les valeurs nous avons dû diviser le vecteur des couples articulaires Γ par le *Rapport de réduction*. Les évolutions que nous avons obtenu sont:

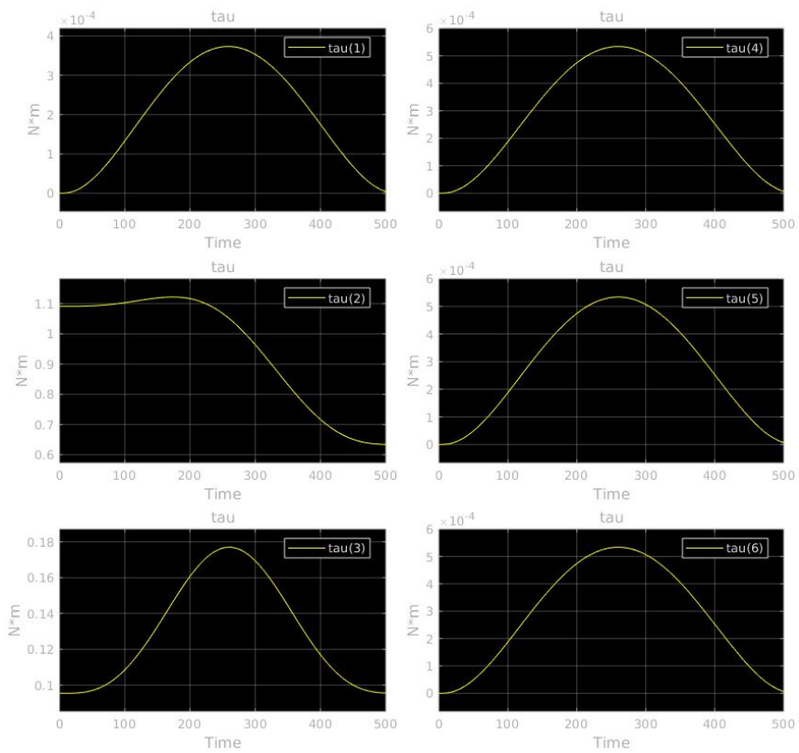

Figure 7.7: Évolution des couples articulaires de commande $\tau_i(t)$

Finalement, il est possible de voir qu'elle est mineur de la couple maximale des moteurs.

Les mains des TPs

TP1

Code 38: *TP1.m*

```
%% Parametres generales
close all
clear all
clc
% Configurations articulaires possibles
qi = [-pi/2 0 -pi/2 -pi/2 -pi/2 -pi/2]';
qf = [0 pi/4 0 pi/2 pi/2 0]';

% Configuration articulaire choisi
q = qi;

% Convention DHM
d3 = 0.7;
r1 = 0.5;
r4 = 0.2;
alpha = [0 pi/2 0 pi/2 -pi/2 pi/2];
d = [0 0 d3 0 0 0];
theta = [q(1) q(2) pi/2+q(3) q(4) q(5) q(6)];
r = [r1 0 0 r4 0 0];
rE = 0.1;

%% Q1 et Q2
% Sur le rapport

%% Q3: MGD du robot
g_6E = CalculTransformationElem(0,0,0,rE);
[g_06, g_element] = CalculMGD(alpha, d, theta, r);
g_0E = g_06 * g_6E

%% Q4: Axe et angle de rotation equivalent a une rotation donnee

% Vecteur de position
P_0E = g_0E(1:3, 4)
% Matrice de rotation
R_0ER = g_0E(1:3, 1:3);
% Trouve l'angle q et l'axe n a partir d'une rotation donnee
[ang_q,n] = AngleAxe_from_Rot (R_0ER)
```

```
%% Q5: Visualisation des repere du robot
figure('Name','Question 05: Visualisation des repere du robot','NumberTitle','off')
VisualisationRepere(q, 1, 'black');
title('Visualisation des reperes du robot');

%% Q6: Calcul de la Jacobienne et torseurs cinematiques

% Vitesse articulaire
q_point = [0.5 1 -0.5 0.5 1 -0.5]';

% Calcula de la Jacobienne
J = CalculJacobienne(alpha, d, theta, r)
V0E0E = J * q_point


%% Q7: Transmission des vitesses

    %  Limitation de J a les vitesses de translation
    J7 = J(1:3,:)

    % Decomposition en valeurs singuliers
    [U,S,V] = svd(J7*J7.')

    %Direction privilegie
    [m,idx] = max(diag(S));
    Direction_privilegie = V(:,idx);

    % Calcul de la manipulabilite
    eigenvalues = sqrt(diag(S));
    manipulabilite = prod(eigenvalues);

    %Qualite de la transmission de la vitesse
    VisualisationEllipsoides(J(1:3,:), q, g_0E);
    title('Ellipsoide de vitesse');


%% Q8.1 : Calculer modele geometrique inverse

% Parametres du model
X_di = [-0.1 -0.7 0.3]';
q0 = [-1.57 0 -1.47 -1.47 -1.47 -1.47]';
kmax = 100;
aStep = 0.005;
epsx = 0.001;

% Calcul de q*
q_etoile1 = MGI(X_di, q0, kmax, epsx, aStep)

% Verification du resultat 8.1
theta_etoile1 = [q_etoile1(1) q_etoile1(2) pi/2+q_etoile1(3) q_etoile1(4) q_etoile1(5)
    q_etoile1(6)];
[g_06_etoile1, g_elem_etoile1] = CalculMGD(alpha, d, theta_etoile1, r);
g_0E_etoile1 = g_06_etoile1*CalculTransformationElem(0,0,0,rE);
diff1 = abs(X_di - g_0E_etoile1(1:3,4))

%% Q8.2 : Calculer modele geometrique inverse

% Parametres du modele
X_df = [0.64 -0.1 1.14]';
q0 = [0 0.8 0 1 2 0]';
kmax = 100;
aStep = 0.005;
epsx = 0.001;
q_etoile2 = MGI(X_df, q0, kmax, epsx, aStep)

% verification du resultat 8.2
theta_etoile2 = [q_etoile2(1) q_etoile2(2) pi/2+q_etoile2(3) q_etoile2(4) q_etoile2(5)
    q_etoile2(6)];
[g_06_etoile2, g_elem_etoile2] = CalculMGD(alpha, d, theta_etoile2, r);
g_0E_etoile2 = g_06_etoile2*CalculTransformationElem(0,0,0,rE);
diff2 = abs(X_df - g_0E_etoile2(1:3,4))


%% Q9: Trajectoire a suivre
%Parametres donnees dans l'enonce de la question
X_di = [-0.1 -0.7 0.3]';
X_df = [0.64 -0.1 1.14]';
```

```
V = 1;
Te = 1e-3;

% Execute la fonction de suivi de trajectoire
[Xtot, qTot] = MCI(X_di,X_df,V,Te,qi);

% Plot les resultats obtenus
figure('Name','Question 09: Trajectoire a suivre','NumberTitle','off')
VisualisationChemin(qTot, Xtot, 6);
title('Chemin suivi (mouvement rectiligne)');
%% Q10
% afficher les differents courbes
qmin = [-pi -pi/2 -pi -pi -pi/2 -pi]';
qmax = [0 pi/2 0 pi/2 pi/2 pi/2]';

% Plot graphique avec l'evolution de chaque variable articulaire
figure('Name','Question 10: evolution des variables articulaires','NumberTitle','off')
plotEvolution(qTot, qmin, qmax)
%% Q11: eloignement des butees articulaires

% Parametres
X_di = [-0.1 -0.7 0.3]';
X_df = [0.64 -0.1 1.14]';
V = 1;
Te = 0.001;
qmin = [-pi -pi/2 -pi -pi -pi/2 -pi]';
qmax = [0 pi/2 0 pi/2 pi/2 pi/2]';

% Chemin a suivre avec constraint
[Xtot, qTot] = MCIbutees(X_di, X_df, V, Te, qi, qmin, qmax);

% Plot graphique avec l'volution de chaque variable articulaire
figure('Name','Question 11: volution des variables articulaires','NumberTitle','off')
plotEvolution(qTot, qmin, qmax)

figure('Name','Question 11: Visualisation du Chmin','NumberTitle','off')
VisualisationChemin(qTot, Xtot, 6);
title('Chemin suivi (mouvement rectiligne avec constraint)');
```

TP2

Code 39: *TP2.m*

```
%% Parametres generales
close all
clear all
clc

% Initalisation Parametres
q = [-pi/2 0 -pi/2 -pi/2 -pi/2 -pi/2]';
dq = [0.5 1 -0.5 0.5 1 -0.5]';
% Convention DHM
d3 = 0.7;
r1 = 0.5;
r4 = 0.2;
alpha = [0 pi/2 0 pi/2 -pi/2 pi/2];
d = [0 0 d3 0 0 0];
theta = [q(1) q(2) pi/2+q(3) q(4) q(5) q(6)];
r = [r1 0 0 r4 0 0];
rE = 0.1;

% Initalisation Parametres 2
m = [15 10 1 7 1 0.5];
x_G1 = 0; y_G1 = 0; z_G1 = -0.25;
x_G2 = 0.35; y_G2 = 0; z_G2 = 0;
x_G3 = 0; y_G3 = -0.1; z_G3 = 0;
x_G4 = 0; y_G4 = 0; z_G4 = 0;
x_G5 = 0; y_G5 = 0; z_G5 = 0;
x_G6 = 0; y_G6 = 0; z_G6 = 0;
x_G = [x_G1 x_G2 x_G3 x_G4 x_G5 x_G6]';
y_G = [y_G1 y_G2 y_G3 y_G4 y_G5 y_G6]';
z_G = [z_G1 z_G2 z_G3 z_G4 z_G5 z_G6]';

%% Q12 - MatriceJacobienneGi
[OJv_Gi, OJ_wi] = CalculMatriceJacobienneGi(alpha, d, theta, r, x_G, y_G, z_G);
OVGi = [];
OwGi = [];
```

```
for i = 1:6
OVGi = [OVGi OJv_Gi(:,:,i) * dq];
OwGi = [OwGi OJ_wi(:,:,i) * dq];
end

% Les vitesses
OVGi
OwGi
%% Q13 - Matrice d'Inertie
A = CalculMatriceInertie(q)

%% Q14: Borne de la Matrice A

%  butes  articulaires qmin et qmax definies a la question Q10
qmin = [-pi -pi/2 -pi -pi -pi/2 -pi]';
qmax = [0 pi/2 0 pi/2 pi/2 pi/2]';

% Calcul des bornes pour la matrice A
[mu1,mu2] = FindBorneA(A, qmin, qmax, 1000)

%% Q15: Calcule du couple de gravite
G = CalculCoupleGravite(q)

%% Q16: Calcule d'un majorant pour G

% butees articulaires qmin et qmax definies a la question Q10
qmin = [-pi -pi/2 -pi -pi -pi/2 -pi]';
qmax = [0 pi/2 0 pi/2 pi/2 pi/2]';

% Calcul du majorant pour de G
maxG = FindMajorantG (G, qmin,qmax, 1000)

%% Q17: Modele dynamique direct sur simulink
% Integree a la question 20

%% Q18: evaluation des parametres de la generation de la trajectoire

% Configurations articulaires desirees
qdi = [-1 0 -1 -1 -1 -1]';
qdf = [0 1 0 0 0 0]';

% Parametres
Te = 0.001;
Rred = [100 100 100 70 70 70];

% Couples moteurs maximale
tau = 5;

% Vecteur des accelerations articulaires maximales
ka = ((Rred * tau) / mu2)';
D = abs(qdf - qdi);

% Calcul des temps
tf = (sqrt((10/sqrt(3))*D./ka))
tf = max(tf)

%% Q19
SimOut=sim('ModeleSimulinkQ19_2016a.slx');

%% Q20
SimOut=sim('ModeleSimulinkQ20_2016a.slx');
```