ROB 315 - TP2: Dynamics and control

Yan Chen & Dajing Gu

January 2020

1 Introduction

In this TP, we will study the **dynamic** and **control** modeling of a robot manipulator, whose geometric and kinematic models were studied in TP1. This practical work (TP) is a continuation of TP1, some solutions of questions from TP1 are used in this TP.

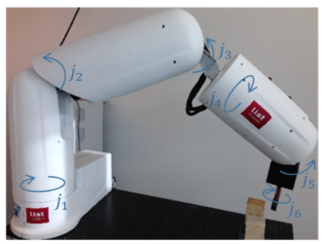

Figure 1.1: Prototype of the robotic manipulation of CEA-LIST

2 Dynamic model

The matrix of the inverse dynamic model for rigid robot manipulator is of the form below:

$$A(q)\ddot{q} + C(q,\dot{q})\dot{q} + G(q) + \Gamma_f(\dot{q}) = \Gamma \tag{2.1}$$

- $A(q) \in \mathbb{R}^{6\times 6}$ inertia matrix, symmetric and positive definite;
- $C(q,\dot{q}) \in \mathbb{R}^6$ vector of joint torques due to the Coriolis and centrifugal forces;
- $G(q) \in \mathbb{R}^6$ vector of joint torques due to gravity;
- $\Gamma_f(\dot{q})$ vector of joint friction torques;

With $q = [q1, ..., q_6]^t$ the vectors of joint positions, $\dot{q} = [\dot{q1}, ..., \dot{q}_6]^t$ the vectors of velocities abd $\ddot{q} = [\ddot{q1}, ..., \ddot{q}_6]^t$ the vectors of accelerations. $\mathcal{R}_i$ are the frames attached to the links of the robot.

2.1 Q12

In order to determine the velocity ${}^0V_{G_i}$ of the center of mass G_i and the rotation speed ${}^0\omega_i$ of all the rigid bodies C_i in the frame $\mathcal{R}_0$, we need to calculate the Jacobian matrices ${}^0J_{v_{G_i}}$ and ${}^0J_{\omega_{G_i}}$ according to the formula :

$${}^0V_{G_i} = {}^0J_{v_{G_i}}(q)\dot{q} \quad \text{and} \quad {}^0\omega_i = {}^0J_{\omega_i}(q)\dot{q}$$

The position of the center of mass G_i expressed in the frame R_i of body C_i is given by

$$\overrightarrow{O_iG_i} = \begin{bmatrix} x_{G_i} & y_{G_i} & z_{G_i} \end{bmatrix}^t \quad \text{for} \quad i = 1, \dots, 6.$$

In the help of the Varignon formula :

$$V_{G_i} = V_{O_E} + \omega_i \times \overrightarrow{O_E G_i}, \text{ ie } : {}^0J_{G_i} = \begin{bmatrix} I_{3\times3} & -{}^0\widehat{\overrightarrow{O_E G_i}} \\ 0_{3\times3} & I_{3\times3} \end{bmatrix} {}^0J_{O_E}$$

in the help of the Rodrigues formula :

$$\widehat{w} = \begin{bmatrix} 0 & -w_z & w_y \\ w_z & 0 & -w_x \\ -w_y & w_x & 0 \end{bmatrix}$$

and in the help of `ComputeJac`$(alpha, d, theta, r)$ developed in **Question 6**, we have programmed the function `ComputeJacGi`$(alpha, d, theta, r, x_G, y_G, z_G)$.

Listing 1: Code of the function `ComputeJacGi`$(alpha, d, theta, r, x_G, y_G, z_G)$

```
function [OJv_Gi, OJ_wi, ROi, O0Oi] = ComputeJacGi(alpha, d, theta, r, x_G, y_G, z_G)

    %Initialisation
    J = ComputeJac(alpha, d, theta, r); %Matrix jacobian of the 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));
    ROi = zeros(3,3,size(J,2));
    g_0i = eye(4);

    % Position vector of the effector organ in R0:

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

    O0Oi = zeros(3,1,6);

    %Loop on the bodies
    for i = 1:size(J,2)
        %Construction of all the Jacobian matrices of the fields Ci
        %expressed at the center Oi of the reference Ri attached to Ci
        OJ_Oi(:,1:i,i) = J(:,1:i);
        %OiGi vector expressed in Ri
        iOiGi = [x_G(i) y_G(i) z_G(i)]';
        %Rotation matrix ROi of the reference R0 to Ri
        g_elem = TransformMatElem(alpha(i), d(i), theta(i), r(i));
        %Rotation matrix ROi of the reference R0 to Ri
        g_0i = g_0i*g_elem;
        ROi(:,:,i) = g_0i(1:3,1:3);
        %OEGi vector in RO to be calculated: OEGi = OEOi + OiGi
        %OiGi vector expressed in R0
        OOiGi = ROi(:,:,i)*iOiGi;
        %OEOi vector expressed in R0: OEOi = OEO0 + O0Oi
        O0Oi(:,:,i) = g_0i(1:3,4);
        OEOi = OEO0 + O0Oi(:,:,i);
        OOEGi = OEOi + OOiGi;
        OPreproduitVect_OEGi = [0 -OOEGi(3) OOEGi(2);...
                                OOEGi(3) 0 -OOEGi(1);...
                                -OOEGi(2) OOEGi(1) 0];
        %Formula of Varignon, the Rodrigus formula
        OJ_Gi(:,:,i) =  [eye(3) -OPreproduitVect_OEGi;...
                        zeros(3,3) eye(3)]*OJ_Oi(:,:,i);

```

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

2.2 Q13

In this question, we want to calculate the inertia matrix $A(q) \in \mathbb{R}^{6\times 6}$. A function $\mathrm{A} =$ `ComputeMatInert(q)` is programmed according to the formula:

$$A = \sum_{i=1}^{N} \left(m_i^0 J_{vG_i}^t(q)^0 J_{vG_i}(q) +^0 J_{\omega_i}^t(q)^0 I_i^0 J_{\omega_i}(q) \right)$$

and the generalized Huygens theorem :

$$I_{G_i} = I_{O_i} - m_i \begin{bmatrix} b^2+c^2 & -ab & -ac \\ (\text{ sym. }) & a^2+c^2 & -bc \\ (\text{ sym. }) & (\text{ sym. }) & a^2+b^2 \end{bmatrix}$$

The influence of the actuators is also taken into consideration.

Listing 2: Code of the function `A = ComputeMatInert(q)`

```
function A = ComputeMatIner(q)
    % parameters
    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 = zeros(3,3,length(q));
    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 of the tables
    g_0i = eye(4);
    Ig = zeros(3*length(q),3);
    A = zeros(length(theta),length(theta));
    ROi = zeros(3,3,length(m));

    % Calculation of the Jacobian matrices of each body
    [OJv_Gi, OJ_wi, ROi, OOOi] = ComputeJacGi(alpha, d, theta, r, x_G, y_G, z_G);
    % Calculation of the kinetic energy of each body
    A = zeros(length(theta), length(theta));
```

```
    for i = 1:length(m)
        %Expression of the tensor of inertia matrix iI_Gi of Ci in Ri of
        %origin Gi, in the statement of the question it's expressed in Ri
        %of origin Oi
        OPreproduitVect_OiGi = [0 -z_G(i) y_G(i); z_G(i) 0 -x_G(i); -y_G(i) x_G(i) 0];
        iI_Gi = iI(:,:,i) + m(i) * OPreproduitVect_OiGi* OPreproduitVect_OiGi; % ... Generalized Huygens theorem: Huygens = - OPreproduitVect_OiGi* ... OPreproduitVect_OiGi

        % Expression of inertia tensor expressed in R0
        OI_Gi = ROi(:,:,i) * iI_Gi * ROi(:,:,i)';
        A = A + ((m(i)*OJv_Gi(:,:,i)')*OJv_Gi(:,:,i) + (OJ_wi(:,:,i)')*OI_Gi*OJ_wi(:,:,i));
    end

    %Addition of actuator inertia contributions
    A = A + diag(Rred.^2.*Jm);
end
```

In order to test the code, we calculate the inertia matrix of $\boldsymbol{q_i}$, whose result is given as follows:

$$A = \begin{bmatrix} 6.4350 & 0.0000 & 0.0000 & -0.0700 & -0.0000 & 0.0000 \\ 0.0000 & 7.1650 & 0.9100 & 0.0000 & 0.0000 & 0.0100 \\ 0.0000 & 0.9100 & 1.0100 & 0.0000 & 0.0000 & 0.0100 \\ -0.0700 & 0.0000 & 0.0000 & 0.1190 & 0.0000 & 0.0000 \\ -0.0000 & 0.0000 & 0.0000 & 0.0000 & 0.0690 & 0.0000 \\ 0.0000 & 0.0100 & 0.0100 & 0.0000 & 0.0000 & 0.0590 \end{bmatrix}$$

2.3 Q14

In this question, we want to calculate the lower and upper bounds $0 < \mu_1 < \mu_2$ of the inertia matrix in using the computation of the eigenvalues of $A(q)$, which means :

$$\mu_1 \mathbb{I} \preceq A(q) \preceq \mu_2 \mathbb{I}$$

A discretization of joint angles will be made between limits q_{min} and q_{max} defined in question **Q10** , and for each value q in this discretization, we will note the maximum eigenvalue and the minimum eigenvalue of the inertia matrix associated to calculate finally the two bounds.

Listing 3: Code of the function `ComputeBounds(A, qmin, qmax, N)`

```
function [mu1, mu2] = ComputeBounds(A, qmin, qmax, N)
    %N: number of iterations
    dq = (qmax-qmin)/N;
    q = qmin;
    mu1 = inf;
    mu2 = 0;
    for i = 1:N
        q = q + dq;
        A = ComputeMatIner(q);
        if max(eig(A)) > mu2
            mu2 = max(eig(A));
        end
        if min(eig(A)) < mu1
            mu1 = min(eig(A)) ;
        end
    end
end
```

Here the code for Q14:

Listing 4: Code of Q14

```

```

```
% Parameters
qmin = [-pi -pi/2 -pi -pi -pi/2 -pi];
qmax = [0 pi/2 0 pi/2 pi/2 pi/2];
N = 500 %N: number of iterations
[mu1,mu2] = ComputeBounds(A, qmin, qmax, N)
```

And we can obtain that $\boldsymbol{\mu_1 = 0.0574}$, and $\boldsymbol{\mu_2 = 10.1985}$.

2.4 Q15

In this section, we compute the vector of joint torques due to gravity $G(q) \in \mathbb{R}^6$ in the matrix form of the inverse dynamic model 2.1. The analytical expression of the gradient of the potential energy $E_p(q) = g^t \left(\sum_{i=1}^{6} m_i {}^0p_{G_i}(q) \right)$ is used here, that is :

$$G(q) = -\left({}^0J_{vG_1}^t m_1 g + ... + {}^0J_{vG_6}^t m_6 g \right) \tag{2.2}$$

where $g = [0 \quad 0 \quad -9.81]^t$.

Listing 5: Code of the function `G = ComputeGravTorque(q)`

```
function [G] = ComputeGravTorque(q)

    %Parameters
    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)];

    % Calculation of the Jacobian matrices of each body
    [OJv_Gi, ¬] = ComputeJacGi(alpha, d, theta, r, x_G, y_G, z_G);

    %Gravity vector expressed in R0
    g = [0 0 -9.81]';

    %Calculation of the torque of gravity produced by bodies
    G = zeros(length(theta), 1);

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

2.5 Q16

A upper bound g_b of $||G(q)||_1$ is considered in this section, such that :

$$\forall q \in [q_{min}, q_{max}], \quad ||G(q)||_1 \leq g_b \tag{2.3}$$

where $|| \bullet ||_1$ denotes the norm 1 of a vector.

The programming method is just like **Q14**, a discretization of joint angle is used. The code is as follows :

Listing 6: Code of the function `UpperBound = ComputUpperBound(q, qmin, qmax, N)`

```
function [UpperBound] = ComputeUpperBound(G, qmin, qmax, N)

    UpperBound = 0;
    dq = (qmax - qmin)/N;
    q = qmin;
    for i = 1:N
        q = q + dq;
        G = ComputeGravTorque(q);
        G_Norm = norm(G,1);
        if G_Norm > UpperBound
            UpperBound = G_Norm;
        end
    end
end
```

2.6 Q17

In this question, a simulation block of the robot is designed to simulate the direct dynamic model. The previous created functions are exploited and two new functions: Γ_f = `ComputeFrictionTorque`$(\dot{q})$ and `c = ComputeCCTorque`$(q, \dot{q})$ are also used.

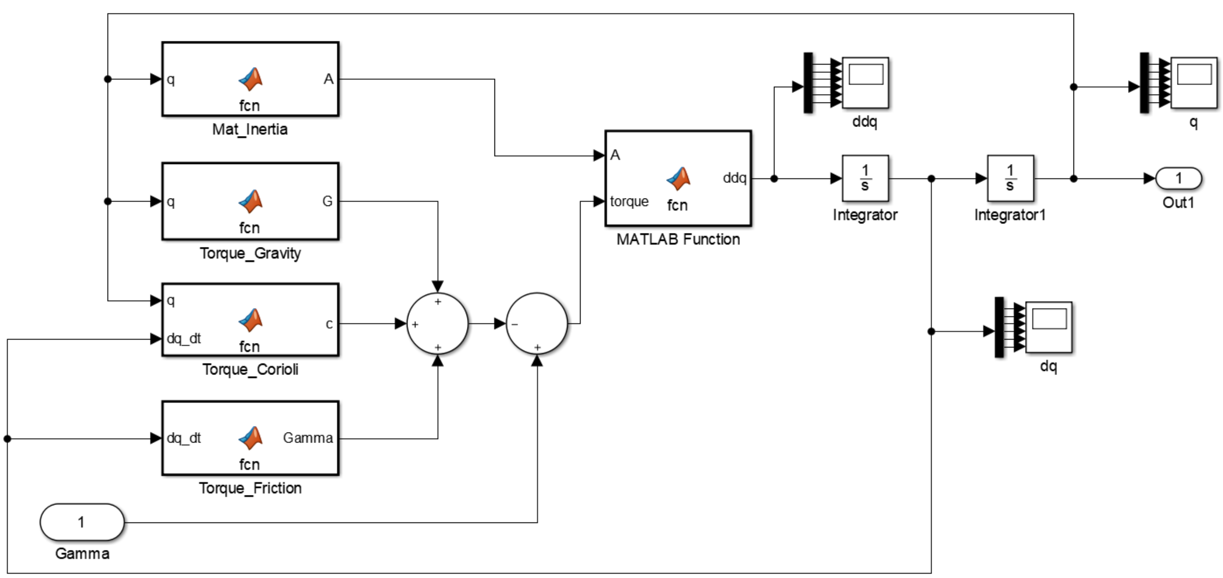

Figure 2.1: Simulink model of Q17

The function Γ_f = `ComputeFrictionTorque`(dq_dt) is programmed according to the formula, which returns the vector of joint torques produced by joint friction. :

$$\tau_{f_i}(\dot{q}_i) = \text{diag}(\dot{q}_i) F_{v_i} \tag{2.4}$$

The code associated is presented below:

Listing 7: Code of the function Γ_f = `ComputeFrictionTorque`(dq_dt)

```
function Gamma = ComputeFrictionTorque(dq_dt)
    % parameters
    Fv = 10*ones(length(dq_dt), 1);
    Gamma = diag(dq_dt)* Fv;
end
```

The function `c = ComputeCCTorque`$(q, \dot{q})$ which returns the vector of joint torques to the Coriolis and centrifugal effects have already been pre-programmed.

According to the formula ci-dessous, the simulation block presented in the figure 2.1 is created.

$$\ddot{q} = A(q)^{-1} \left(\Gamma - C(q, \dot{q})\dot{q} - G(q) - \Gamma_f \right) = A(q)^{-1}(\Gamma - H(q, \dot{q})) \tag{2.5}$$

3 Trajectory generation in the joint space

3.1 Q18

In this section, we want to generate a polynomial trajectory of degree 5 to be followed in the joint space allowing to reach in minimal time tf the desired final configuration q_{d_f} from the initial configuration q_{d_i}. These initial and final configurations are defined as follows :

$$\begin{aligned} q_{d_i} &= [-1.00, 0.00, -1.00, -1.00, -1.00, -1.00]^t \, rad \\ q_{d_f} &= [0.00, 1.00, 0.00, 0.00, 0.00, 0.00]^t \, rad \end{aligned} \tag{3.1}$$

This movement is performed at zero initial and final velocities and accelerations, and is sampled at a period $Te = 1ms$. So here are the boundary conditions :

$$\begin{aligned} q(0) &= q_{d_i} \quad q(t_f) = q_{d_f} \\ \dot{q}(0) &= 0, \quad \dot{q}(t_f) = 0 \\ \ddot{q}(0) &= 0, \quad \ddot{q}(t_f) = 0 \end{aligned} \tag{3.2}$$

Minimal final time for the j-th joint can be calculated by the equation as follows :

$$t_{f_i} = max \left[\frac{15|D_i|}{8k_{v_i}}, \sqrt{\frac{10|D_i|}{\sqrt{s}k_{a_i}}} \right] \tag{3.3}$$

But the vector k_a of maximum joint accelerations is just taken into account here. So the equation above becomes :

$$t_{f_j} = max \left[\sqrt{\frac{10|D_j|}{\sqrt{3}k_{a_j}}} \right] \tag{3.4}$$

where $k_{a_i} = \frac{r_{red_i} * \tau_{max_i}}{\mu_2}$

Finally, we have minimal final global time t_f while coordinating all the joints :

$$t_f = max(t_{f_1}, ..., t_{f_n}) \tag{3.5}$$

Listing 8: Calculate the minimal final global time t_f

```
%% Q18
qdi = [-1 0 -1 -1 -1 -1]';
qdf = [0 1 0 0 0 0]';
Te = 0.001;
Rred = [100 100 100 70 70 70];
```

```
tau = 5;
mu2 = 10.1985;
ka = ((Rred*tau)/mu2)';
D = abs(qdf -qdi);
tf = sqrt(10*D ./ (sqrt(3)*ka));
tf = max(tf);
```

The value of the minimal final global time : $t_f = 0.4102\, seconds$.

3.2 Q19

In this question, we study the temporal evolution of the desired joint trajectories q_{c_i} (for $i = 1, ..., 6$) from q_{d_i} to q_{d_f} when t varies from 0 to a minimum final global time $t_f = 0.5$ s.

According to the general formulation:

$$q(t) = q_{d_i} + r(t)D \quad \text{where} \quad D = q_{d_f} - q_{d_i} \tag{3.6}$$

and the expression of the time law:

$$r(t) = 10\left(\frac{t}{tf}\right)^3 - 15\left(\frac{t}{tf}\right)^4 + 6\left(\frac{t}{tf}\right)^5 \tag{3.7}$$

the function `qc = GenTraj`(q_{d_i}, q_{d_f}, t) is then programmed.

Listing 9: `qc = GenTraj`(q_{d_i}, q_{d_f}, t)

```
function qc = GenTraj(q_di, q_df, t)
    D = q_df - q_di;
    tf = 500; %ms
    r = 10*(t/tf)^3 -15*(t/tf)^4 + 6*(t/tf)^5;
    qc = q_di + r*D
end
```

We have created the trajectory generation with Simulink as illustrated in the Figure 3.1.

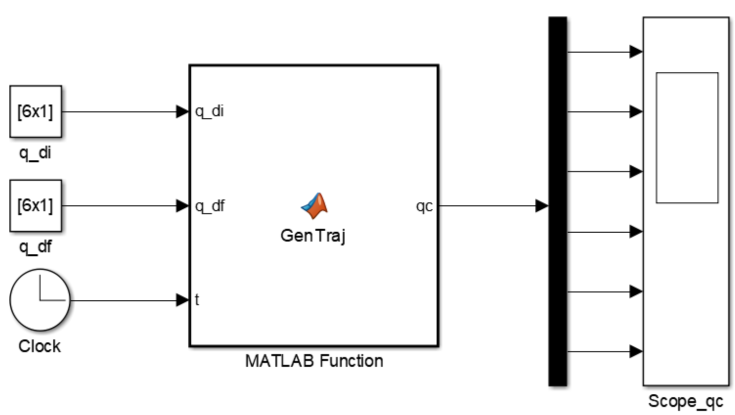

Figure 3.1: Trajectory generation with Simulink

And we have simulated the temporal evolution of the joint trajectories q_{c_i} during $0.5s$. The results are shown in the Figure 3.2. These graphs shows that every joint has same change tendency of velocity. The velocity rises gradually to the maximum, then remains constant, and finally decreases gradually to zero. The change of velocity is nonlinear.

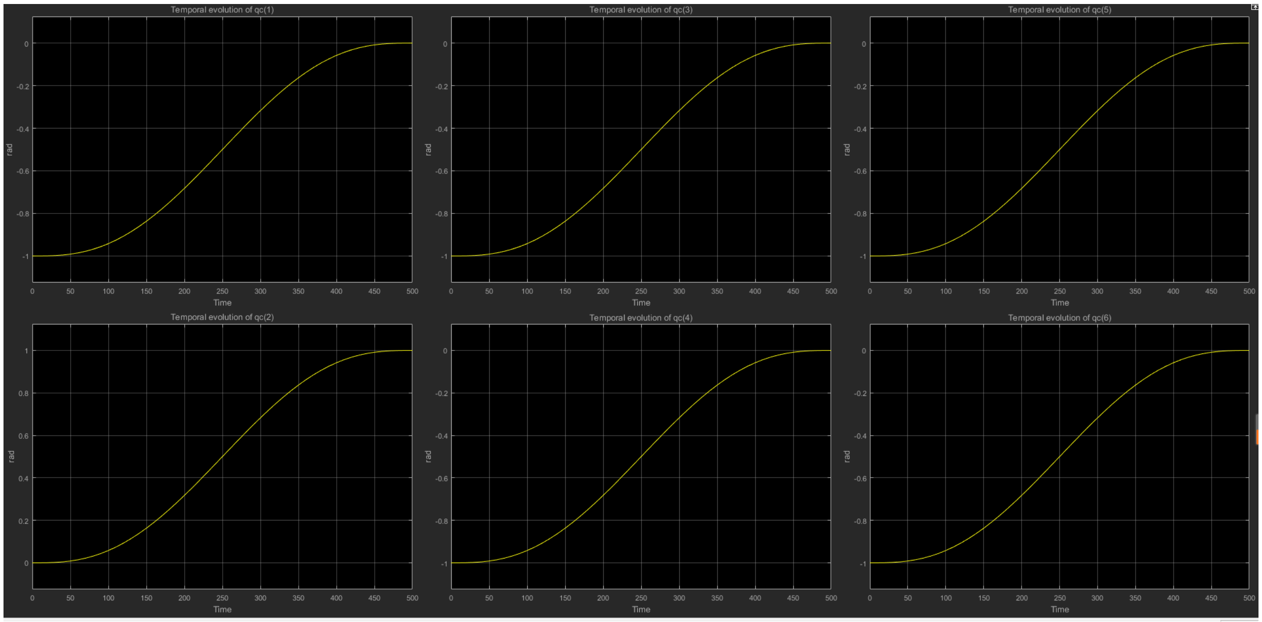

Figure 3.2: Temporal evolution of the desired joint trajectories q_{c_i}

4 Joint control law

4.1 Q20

The robot is position-controlled with a P.D. controller with gravity compensation:

$$\Gamma = K_p\left(q_d - q\right) + K_d\left(\dot{q}_d - \dot{q}\right) + \hat{G}(q) \tag{4.1}$$

By choosing the joint gains $\boldsymbol{K_{p_i} = 1}$ and $\boldsymbol{K_{d_i} = 0.5}$, the position-control block is implemented in *Simulink*, whose schematic diagram is presented below:

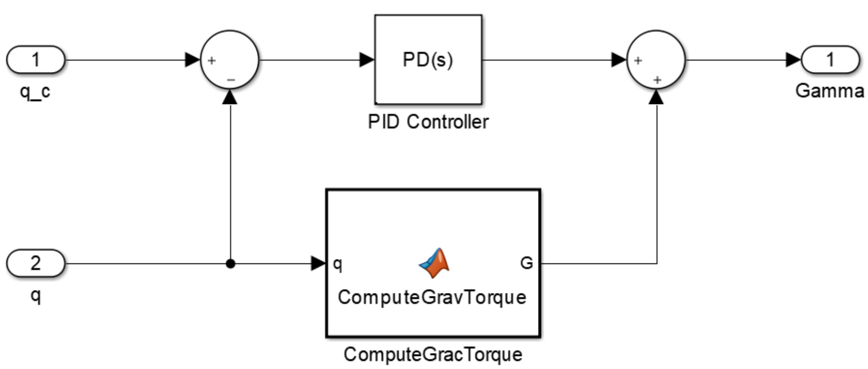

Figure 4.1: Position-control blcok in Simulink

Then using the previously defined blocks, the closed-loop control scheme is built, which is presented in the figure 4.2.

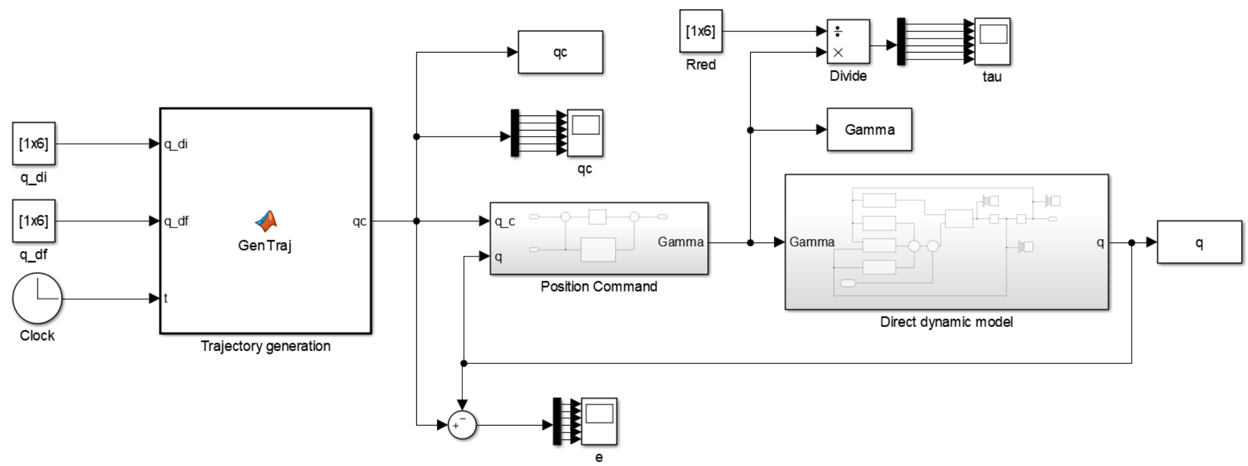

Figure 4.2: Closed-loop control scheme in Simulink

Here we plot the temporal evolution of joint trajectories $q_i(t)$ presented in the figure 4.3, as well as the temporal evolution of the tracking errors $e(t)$ presented in the figure 4.4:

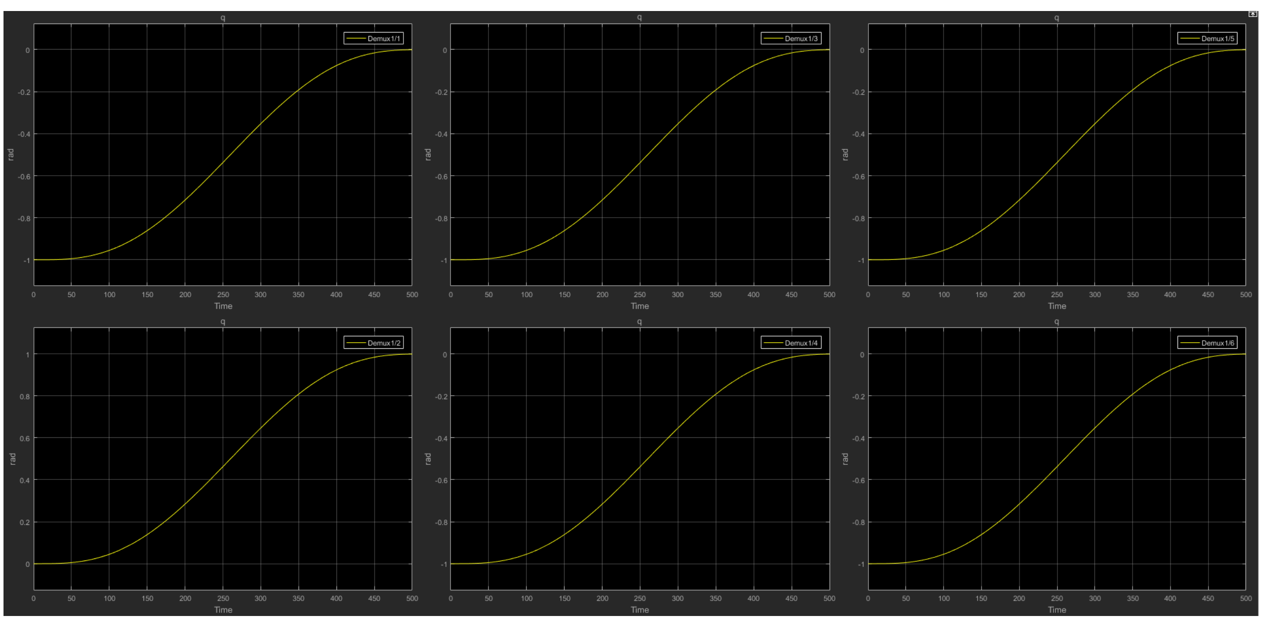

Figure 4.3: Temporal evolution of joint trajectories q_i

We plot also the temporal evolution of the control joint torques $\tau_i(t)$ corresponding to our gain tuning in order to make sure its values are smaller than the maximum torque of $5N * m$.

By dividing $\Gamma_i(t)$ with the reduction ratio, we can obtain the control joint torques $\tau_i(t)$, which is presented in the figure 4.5. It can be seen that $\tau_i(t)$ is always smaller than $5N * m$, which has verified our choice of K_{p_i} and K_{d_i}.

The figures of other parameters, such as q_c, $\dot{q}$, $\ddot{q}$ are also available in our model of *Simulink* although they are not presented in this report.

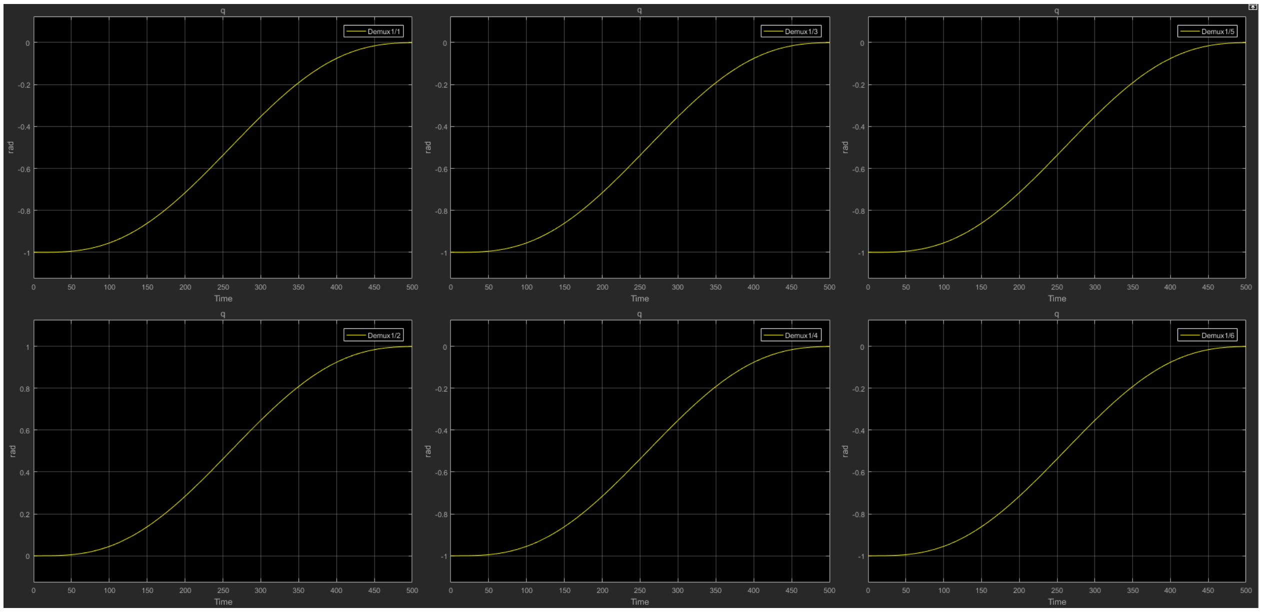

Figure 4.4: Temporal evolution of tracking errors $e(t) = q_c(t) - q(t)$

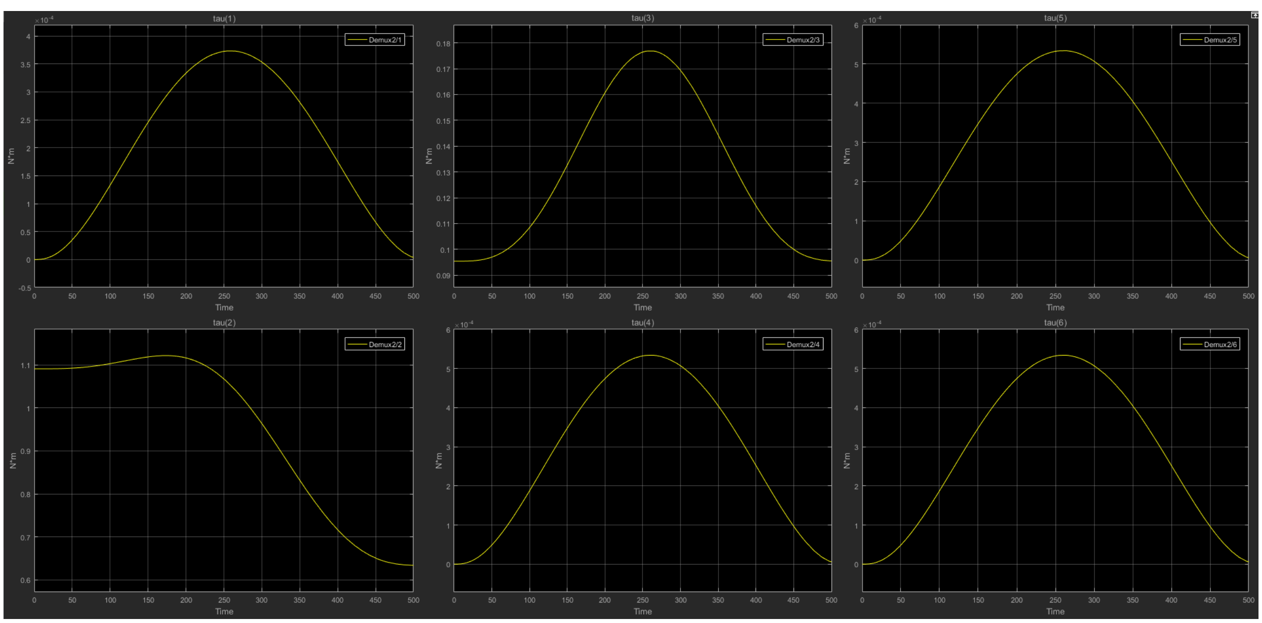

Figure 4.5: Temporal evolution of the control joint torques $\tau_i(t)$