Queste Adrien, TRAÏA Yacine

Exam Robotics 2

Réalisé à l'ENS Rennes

Date de remise : 30/04/2024

Table des matières

1 Introduction

The robot we are considering has 4 axes (pivot connections). So, if we call the space joint N, in our configuration we have N = 4. The instruction concerns a robot performing movements in the plane whose dimension we note as "dim R = M", with M = 2. We then have N > M. We find ourselves in the case of a redundant robot ! Under these conditions, we carried out a study where we delve into two essential aspects of robotic manipulation : redundancy resolution and dynamic modeling with control strategies. Starting with redundancy resolution, we define the robot's configuration and derive its forward and differential kinematic relations, enabling us to compute trajectories for end-effector tasks. We then explore velocity control methods and verify solutions, including error correction strategies. Transitioning to dynamic modeling and control, we employ the Lagrange method to derive dynamic equations and implement regulating controllers such as PD, PID, and PD with gravity compensation. Finally, in an advanced exercise, we integrate kinematic and dynamic control approaches to execute tasks seamlessly, focusing on trajectory tracking and control optimization.

2 Redundancy resolution

In this section, we're going to implement 4 methods we've seen in class for following a trajectory we want to impose on our robot. The trajectory we have chosen is a circular trajectory with a radius of 500.

Question 1. Define the robot configuration q $\in R^N$
To define the robot correctly. Homogeneous transformation matrixs were used. Robot lengths were defined as follows :

— l1=800m ; l2=600m ;

— l3=400m ; l4=200m.

We could consider this length for the robot because, in this section, we are not including the physics of the robot, so a length of 800 m is possible. The robot has 4 pivots along the z axis. This means that its end effector is only located in the (Oxy) plane. This means that its end-effector is only located in the (Oxy) plane. Furthermore, given that this is a 4-arm robot and that it only works in one plane, it has 4 degrees of freedom and is therefore, by definition, redundant, as we mentioned earlier.This redundancy offers multiple advantages for improving robot performance. In particular, it enables obstacles to be avoided and kinematic singularities to be bypassed, while optimising execution time and minimising energy consumption. Following the rules we have set out, we can find the expression for T_{01}, T_{12}, T_{23}, and T_{34}.T_{01},T_{12},T_{23}, and T_{34}

$$T_{01} = \begin{pmatrix} cos(q_1) & -sin(q_1) & 0 & 0 \\ sin(q_1) & cos(q_1) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} ; \; T_{12} = \begin{pmatrix} cos(q_2) & -sin(q_2) & 0 & 0 \\ sin(q_2) & cos(q_2) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$T_{23} = \begin{pmatrix} cos(q_3) & -sin(q_3) & 0 & 0 \\ sin(q_3) & cos(q_3) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} ; \; T_{34} = \begin{pmatrix} cos(q_4) & -sin(q_4) & 0 & 0 \\ sin(q_4) & cos(q_4) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Question 2. Considering a given task r $\in R^M$, in the specific the position of the robot's end-effector, derive the forward kinematic relation, f(q) : $R^N \rightarrow R^M$, such that r(t) = f(q(t))
As we mentioned earlier the task was to definite a circle. So it was the task of the robot. In order to derive this relation, we have to understand what are r ant q. R is a list of points (x,y) and q are the angles of the robots. In order to join this two relation ; we have to define the direct and the indirect geometric model. In our case, we have the access to r(t) so we have to define q(t). This is the reason why we define in our code the direct geometric model :

```
function X=MGD_PEINTRE_SIMPLE(Peintre,q)

l1=Peintre.l1;
l2=Peintre.l2;
l3=Peintre.l3;
l4=Peintre.l4;
q1=q(1);
q2=q(2);
q3=q(3);
q4=q(4);

 X=[l2*cos(q1 + q2) + l1*cos(q1) + l3*cos(q1 + q2 + q3) + l4*cos(q1 + q2 + q3+q4)
    l2*sin(q1 + q2) + l1*sin(q1) + l3*sin(q1 + q2 + q3) + l4*sin(q1 + q2 + q3+q4)];



end
```

Question 3. Derive the differential kinematic relation, and in particular the Jacobian matrix, J(q) $\in R^(MN)$ such that $\dot{r}(t) = \frac{d}{dt} f(q(t))$=J(q)$\dot{q}$.
In order to calculate the Jacobian, we have ton consider the task. We want to describe a trajectory on a plan so it will define the jacobian as the derivation matrix from X.

```
MATLAB Variable: J                                          Page 1
26 avr. 2024                                          10:17:59 AM

val(q1, q2, q3, q4) =

[- 400*sin(q1 + q2 + q3) - 200*sin(q1 + q2 + q3 + q4) - 600*sin(q1 + q2) - 800*sin
(q1), - 400*sin(q1 + q2 + q3) - 200*sin(q1 + q2 + q3 + q4) - 600*sin(q1 + q2), -
400*sin(q1 + q2 + q3) - 200*sin(q1 + q2 + q3 + q4), -200*sin(q1 + q2 + q3 + q4)]
[  400*cos(q1 + q2 + q3) + 200*cos(q1 + q2 + q3 + q4) + 600*cos(q1 + q2) + 800*cos
(q1),   400*cos(q1 + q2 + q3) + 200*cos(q1 + q2 + q3 + q4) + 600*cos(q1 + q2),
400*cos(q1 + q2 + q3) + 200*cos(q1 + q2 + q3 + q4),  200*cos(q1 + q2 + q3 + q4)]
```

FIGURE 1 – Jacobian matrix

In this matrix, we can see the biggest problem that is the Jacobian is a matrix 4x2. The problem is that this matrix is not invertible, but we can calculate the pseudo inverse. This matrix includes the problem of redundancy too and so problem of singularities.

Question 4. Compute a simple trajectory for the end-effector task, e.g., a line or a circle in the robot workspace. You should have the analytical expression of r(t) for $t \in [0, ..., T]$, such that you can compute the corresponding derivatives

We decided to draw a circle by the robot :

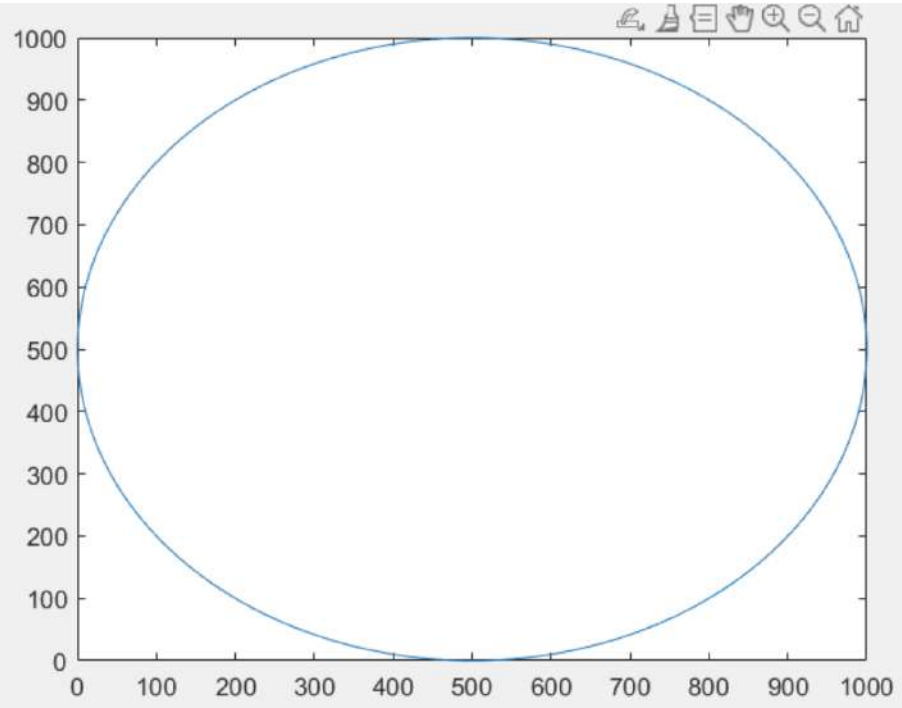

FIGURE 2 – Circle

The center of the circle is at (500,500), the radius of 500m and the first joint of the Robot is at (0,0). This configuration was chosen in order to offer to the robot the possibility of success, but in the same it was the possibility to enter difficulties for the robot in including many points near of the first joint.

Question 5. Assume that the robot is controlled in velocity, i.e., the robot input is $\dot{q}$. For each time stamp $t_k \in [0, ..., T]$, with a certain sampling Δt, compute the joint velocity $\dot{q}(tk)$ that accomplish the task, i.e., such that $\dot{r}(tk) = J(q(tk))\dot{q}(tk)$. To solve the problem uses the methods presented in class.

In our case, in order to simplify the implementation of the equations in Matlab, we define a sample time of 1s. This is the reason why, it is written in our code $q = q + dq$. So in order to define the derivative of the task X at a time t, we input $dX = X(t+1) - X(t)$. To do that, we use close points on the circle in order to have a correct approximation of the derivative. It works pretty good. In all cases, we export video of tracing, they are available in the files.zip. It was important for us to understand which methods are the better. So we decide to define the initial position of the robot different as the first point forced by the trajectory. This choice was important in order to see what is the comportment of the method when we are not at the good position. In the first approach, we decide to not optimize the different method.

Pseudo inverse

For this first method we used a method based on the Jacobian. This consists of choosing a solution from among the inifinities that typically exist, which minimises an appropriate norm. We can see the main point in the code that is $dq = J^{\#}dX$. For each point, we do an iteration in order to calculate dX at each moment. If we are far, we have big dX and so a big velocity at the joint. This method shows some initial difficulties in correctly detecting the circle, especially with a less dense discretisation. Limitations of Jacobian-based methods include the lack of

guaranteed global avoidance of singularities due to the resolution solution arbitrarily chosen during execution, the reduction of joint velocities which remains a purely local property likely to trigger unwanted phenomena, and the induction of non-repeatable movements in joint space. Furthermore, cyclic movements in task space do not necessarily translate into cyclic movements in joint space, adding a further layer of complexity.. For us this method was the worst method. Moreover, we can see in this method that the gestion of the arms is terrible. This is reason why we implemented the others methods. Video nammed Jacobian non optimization

Null-space projection

The second class of methode that we used to solve redundancy is the null-space method. As the name said this method explore the self motion capability of the structure. The general solution of the equation $J\dot{q} = \dot{r}$ can be written in the following form : $\dot{q} = J^{\#}\dot{r} + (I - J^{\#}J)\dot{q_0}$. Here $\dot{q}$ is a particular solution of $J\dot{q} = \dot{r}$ and $(I - J^{\#}J)\dot{q_0}$ is the set of all possible solutions associated to $J\dot{q} = 0$ so $\dot{q}$ is exactly the null space of J. So if this null space motion can be parameterised in this form that's means that the matrix $(I - J^{\#}J)$ is the orthogonal projection of $\dot{q_0}$ In this method, we use this code :

```
    function [q,Xr]=CINEMATIQUE_INVERSE_TRAJECTOIRE_Null_SPACE(Peintre,X,q0,qinit)
% q --> trajectoire articulaire, tableau contenant autant de lignes que de ddl du
% robot et autant de colonnes que de points dans la trajectoire
% Xr --> trajectoire op rationnelle reconstruite    partir de q pour
% validation
% Peintre --> param tres g om triques du robot
% X --> trajectoire op rationnelle, tableau contenant 2 lignes (X et Y) et
% autant de colonnes que de points dans la trajectoire
% alpha --> facteur d'acroissement pour la fonction it rative entre 0 et 1
% tol --> tol rance sur la position du robot    chaque pas
% imax --> nombre d'it rations max pour la boucle de calcul par jacobienne
% qinit --> coordonn es articulaires initiant le calcul

q=zeros(numel(qinit),numel(X(1,:)));
Xr=zeros(numel(X(:,1)),numel(X(1,:)));
q_int=qinit;
for i=1:numel(X(1,:)) %pour tout point de la trajectoire
X_int=MGD_PEINTRE_SIMPLE(Peintre,q_int); % on calcule la position courante du robot (
    modifier avec votre propre MGD)
dX=X(:,i)-X_int;%on calcule la distance entre la position courant et la position d sir e
J=J_PEINTRE_SIMPLE(Peintre,q_int); % jacobienne
Jinv=pinv(J);
dq=eval(Jinv*dX+(eye(4)-Jinv*J)*q0(q_int(1),q_int(2),q_int(3),q_int(4)));
q_int=q_int+dq;%accroissement sur q
q(:,i)=q_int;% on remplit le tableau q
Xr(:,i)=MGD_PEINTRE_SIMPLE(Peintre,q_int);% on calcule la trajectoire du robot  partir de q
    pour v rification.

end


end
```

In this method, we implemented the gradient in order to limit the angle of the first and the last joint between (-10 and 10 degrees). This method was really great. However, there is a big velocities and movements at the beginning when the arms are not on the circle. So we can correct it with implementation. In this try, we can see that the robot can really perform the task and maintain the first and last arms in the good angles. The inconvenient of this method is that you have to calculate the gradient. However, this method seems really great in order to avoid obstacles or to limit so parameters of the robots. Video nammed : null sapce projection

Task augmentation

```
function [q,Xr]=Task_augmented(Peintre,X,q1_target,q4_target,Jq,Jq2,qinit)

q=zeros(numel(qinit),numel(X(1,:)));
Xr=zeros(numel(X(:,1)),numel(X(1,:)));
q_int=qinit;

for i=1:numel(X(1,:)) %pour tout point de la trajectoire
X_int=MGD_PEINTRE_SIMPLE(Peintre,q_int); % on calcule la position courante du robot (
    modifier avec votre propre MGD)
dX=X(:,i)-X_int;dq4=q4_target-q_int(4);dq1=q1_target-q_int(1);dtarget=[dX;dq4;dq1];%on calcule
     la distance entre la position courant et la position d sir e
J=J_PEINTRE_SIMPLE(Peintre,q_int);
J=[J;Jq;Jq2];
Jinv=pinv(J);
dq=Jinv*dtarget;
q_int=q_int+dq;%accroissement sur q
q(:,i)=q_int;% on remplit le tableau q
Xr(:,i)=MGD_PEINTRE_SIMPLE(Peintre,q_int);% on calcule la trajectoire du robot  partir de q
    pour v rification.

end


end
```

In this code, we can see that the method is not different as the pseudo inverse. In fact, we have only modified the Jacobian. At the beginning, we've said that the problem of the robot is 4 degrees of freedom but only 2 degrees are necessary for the task. So we have redundancy. So we can implement others task. Or choice was to fix the fisrt arm at pi/3 and the last one at pi/2. This method could realize many tasks as possible in the same time. So for this method, we have to define the jacobian J2=[0 0 0 1] (control of the last angle) and J3=[1 00 0 0] (control of the first angle). The main problem of this method is if you input bad angles for articulations because if the report can't do the task one of the task due to too many constraints. It will create no results for all the tasks. So this is the main problem of this method, it should have a prioritization in the different tasks (spoiler alert : it's task priority). Video nammed task augmented

Task Priority

In this method, we use this code :

```
function [q,Xr]=Task_Priority(Peintre,X,q4_target,q1_target,Jq,Jq2,qinit)

q=zeros(numel(qinit),numel(X(1,:)));
Xr=zeros(numel(X(:,1)),numel(X(1,:)));
q_int=qinit;

for i=1:numel(X(1,:)) %pour tout point de la trajectoire
X_int=MGD_PEINTRE_SIMPLE(Peintre,q_int); % on calcule la position courante du robot (
    modifier avec votre propre MGD)
dX=X(:,i)-X_int;dq4=q4_target-q_int(4);dq1=q1_target-q_int(1);%on calcule la distance entre la
     position courant et la position d sir e
J=J_PEINTRE_SIMPLE(Peintre,q_int);Jinv=pinv(J);J2=Jq;J3=Jq2;J2inv=pinv(J2);J3inv=pinv(J3);
P1=(eye(4)-Jinv*J);P2=(eye(4)-pinv(J2*P1)*(J2*P1));
dq=Jinv*dX+P1*(pinv(J2*P1))*(dq4-J2*Jinv*dX)+P1*P2*((pinv(J3*P2))*(dq1-J3*J2inv*dq4));
q_int=q_int+dq;%accroissement sur q
q(:,i)=q_int;% on remplit le tableau q
Xr(:,i)=MGD_PEINTRE_SIMPLE(Peintre,q_int);% on calcule la trajectoire du robot  partir de q
    pour v rification.

end


end
```

For this method, we input priority on the circle. We decided to force an angle of pi/2 for the last arm. So in theory, the robot can do this task and the first one. We can see on the video that

it is true. Moreover, we input a third task as a 0degres for the fisrt arm. In theroy, the robot can do this many times but not at the bottom of the circle and the right of the circle. We can see in the video that is the real action of the robot. The last angle is at pi/2 every time and the first angle is at every time near to zero even if this arm has to move because mathematically it cost a lot for the robot to change the angle of 0°. However, if the robot has to move he will move. That is what we can see on the video. Video nammed Task priority

Question 6. Verify that the computed solution $q(\dot{t})$ is a real solution. For this integrate $\dot{q}(t)$ (set the initial condition q(0) such that the task is accomplished) to obtain q(t). Compute the task value from forward kinematics and compare it with the desired trajectory. With the first simplification, which is dt=1, in order to obtain q with dq, q(t)=q(t-1)+dq. So with this method, we can observe the different task control.

Question 7. Previous methods are only for planning, i.e., when the robot starts in the correct configuration,and there are no errors, nor disturbances along the motion. Now consider an initial error such that $f(q(0)) \neq r(0)$. Implement a control action in the inverse differential kinematic such that the error goes to zero. In our case, we've decided to include in all the case, disturbance created by initial positions. But we 've created optimization in order to correct it only with the pseudo inverse. This method work too for the other tasks.

3 Dynamic modeling and control

Question 8. Considering joints of rectangular shape, for each link, define it's mass, the center of mass (CoM), and inertia with respect to the CoM. Using the Lagrange method derive the dynamics of the robot. After defining some parameters in the preamble :

```
%param tres inertiels
sym_m1=specs.m1; sym_m2=specs.m2; sym_m3=specs.m3;sym_m4=specs.m4; sym_I1=specs.I1; sym_I2=
    specs.I2;sym_I3=specs.I3; sym_I4=specs.I4
% Param tres du mouvement
syms q1 q2 q3 q4 dq1 dq2 dq3 dq4 ddq1 ddq2 ddq3 ddq4 real
g=9.81; %gravit
    q=[q1 q2 q3 q4]';
    dq=[dq1 dq2 dq3 dq4]';
    ddq=[ddq1 ddq2 ddq3 ddq4]';
    n=numel(q);
```

Dynamic resolution using the Lagrange method begins with the calculation of homogeneous transformation matrices. These were calculated in the first section of this report, so we move directly to step 2, which involves calculating the positions of the centers of mass in order to compute the velocities of the centers of mass and joint velocities.

```

T01=[cos(q1) -sin(q1) 0 0;sin(q1) cos(q1) 0 0; 0 0 1 0; 0 0 0 1];
T12=[cos(q2) -sin(q2) 0 sym_l1;sin(q2) cos(q2) 0 0; 0 0 1 0; 0 0 0 1];
T23=[cos(q3) -sin(q3) 0 sym_l2;sin(q3) cos(q3) 0 0; 0 0 1 0; 0 0 0 1];
T34=[cos(q4) -sin(q4) 0 sym_l3;sin(q4) cos(q4) 0 0; 0 0 1 0; 0 0 0 1];
T02=simplify(T01*T12);
T03=simplify(T01*T12*T23);
T04=simplify(T01*T12*T23*T34);

c1=[sym_l1g 0 0 1]';
c2=[sym_l2g 0 0 1]';
c3=[sym_l3g 0 0 1]';
c4=[sym_l4g 0 0 1]';

c01=T01*c1;
c02=T02*c2;
c03=T03*c3;
c04=T04*c4;
```

Now that we have the positions of each centre of mass, we can calculate the speeds of each arm.

```
vc01=simplify(jacobian(c01(1:3),q)*dq);
vc02=simplify(jacobian(c02(1:3),q)*dq);
vc03=simplify(jacobian(c03(1:3),q)*dq);
vc04=simplify(jacobian(c04(1:3),q)*dq);
```

With respect to this, we calculate the kinetic and potential energy, making sure to express all quantities in the same basis.

```
T1=simplify((1/2)*vc01'*sym_m1*vc01+(1/2)*sym_I1*dq1^2);
T2=simplify((1/2)*vc02'*sym_m2*vc02+(1/2)*sym_I2*(dq1+dq2)^2);
T3=simplify((1/2)*vc03'*sym_m1*vc03+(1/2)*sym_I3*(dq1+dq2+dq3)^2);
T4=simplify((1/2)*vc04'*sym_m1*vc04+(1/2)*sym_I4*(dq1+dq2+dq3+dq4)^2);
T=simplify(T1+T2+T3+T4);

Ep1=sym_m1*g*c01(2);
Ep2=sym_m2*g*c02(2);
Ep3=sym_m3*g*c03(2);
Ep4=sym_m4*g*c04(2);
Ep=Ep1+Ep2+Ep3+Ep4;
```

The inertial and kinetic energy terms can then be grouped together in a single matrix called the mass matrix, denoted M. This matrix represents a generalisation of the mass in the case of systems with several degrees of freedom. The mass matrix is always symmetrical and positive definite. This is logical because the kinetic energy of a system is always a non-negative quantity.

```
    M_temp=simplify(gradient(T,dq));
    M=simplify(jacobian(M_temp,dq));
```

The Coriolis matrix plays an important role in the dynamics of complex mechanical systems, such as multi-articulated robots. It captures the dynamic effects attributable not only to acceleration but also to the interaction of the velocities of the system's components. In the following, this matrix is referred to as $C(q, \dot{q})$, and contains terms that depend on the q configuration and $\dot{q}$ velocities of the system. These terms represent the Coriolis and centrifugal forces, which allow us to describe the behaviour of moving systems in non-inertial settings. To calculate it, we use the properties of the mass matrix $M(q)$ of the system :

$$C_{ij}(q, \dot{q}) = \sum_{k=1}^{n} \left(\frac{\partial M_{ij}}{\partial q_k} + \frac{\partial M_{ik}}{\partial q_j} - \frac{\partial M_{jk}}{\partial q_i} \right) \frac{\dot{q}_k}{2}$$

where M_{ij} are the elements of the mass matrix and n is the number of degrees of freedom. The Coriolis matrix has the following properties :

— The matrix is generally not symmetrical.

— The matrix is multiplied by the velocity vector $\dot{q}$ to produce a force vector which affects the motion of the system, in addition to the external forces and the forces due to mass.

Summing up what we have just said, in our case the Coriolis matrix is given by :

```
syms C real

for i=1:n
    for j=1:n
        C(i,j)=0*g;
        for k=1:n
            C(i,j)=C(i,j)+(1/2)*(diff(H(i,j),q(k))+diff(H(i,k),q(j))-diff(H(j,k),q(i)))*dq(k);
        end
    end
end
C=simplify(C);
```

Then, the matrix $G(q)$, or vector of gravitational forces, is essential for the dynamics of mechanical systems, particularly in studies of robotic mechanics and structures under the influence of

gravity. The matrix $G(q)$, which depends on the configuration q of the system, represents the gravitational forces acting on each degree of freedom of the system. The matrix is calculated by taking into account the gravitational potential of the system :

$$G(q) = \nabla(E_p(\Sigma/R_0), q)$$

where E_p is the gravitational potential energy of the system, which depends on the configuration q and the acceleration due to gravity. Some important properties of the matrix G include :

- It is configuration specific and can vary according to the relative positions of the system components.
- For a system in equilibrium in a uniform gravitational field, G is constant if the configurations of the system remain static.
- This matrix directly influences the control terms in robotic systems to compensate for the forces due to gravity.

Finally, the dynamic equation is written as $M(q)\ddot{q} + C(q, \dot{q}) + G(q) = \tau$ which is translated in matlab as follows :

```
    Tau=simplify(H*ddq+C*dq+G-Fapp);
```

Question 9. Considering a certain initial configuration, show the results of the forward dynamics when you apply :

- zero Torque
- Torque

System response with zero torque

We want to analyse the response of our system if the torque is zero. To do this, we have implemented the following code, which calculates the direct dynamic model of the robot at each iteration.

```
Robot.l1=0.800;Robot.l2=0.600;
Robot.l3=0.400;Robot.l4=0.200;

tau=[0 0 0 0]';
dt=0.01;
k=290;
q0=[0 0 0 0]';dq0=[0 0 0 0]';ddq0=[0 0 0 0]';ddq0old=[0 0 0 0]';dq0old=[0 0 0 0]';
Q=zeros(4,k);dQ=zeros(4,k);ddQ=zeros(4,k);Xr=zeros(2,k);
for i=1:k
    ddq0old=ddq0;dq0old=dq0;
    i
    Q(:,i)=q0;
    dQ(:,i)=dq0;
    dQQ(:,i)=ddq0;
    ddq0=(eval(H(q0(1),q0(2),q0(3),q0(4)))^-1)*(tau-eval(C(q0(1),q0(2),q0(3),q0(4),dq0(1),dq0
    (2),dq0(3),dq0(4)))*[dq0(1) dq0(2) dq0(3) dq0(4)]'-eval(G(q0(1),q0(2),q0(3),q0(4))));

    dq0=dq0+dt*(ddq0+ddq0old)/2;
    q0=q0+dt*(dq0+dq0old)/2;
    Xr(:,i)=MGD_PEINTRE_SIMPLE(Robot,q0);
end
syms q1 q2 q3 q4;

T01(q1)=[cos(q1) -sin(q1) 0 0;sin(q1) cos(q1) 0 0; 0 0 1 0; 0 0 0 1];
T12(q2)=[cos(q2) -sin(q2) 0 Robot.l1;sin(q2) cos(q2) 0 0; 0 0 1 0; 0 0 0 1];
T23(q3)=[cos(q3) -sin(q3) 0 Robot.l2;sin(q3) cos(q3) 0 0; 0 0 1 0; 0 0 0 1];
T34(q4)=[cos(q4) -sin(q4) 0 Robot.l3;sin(q4) cos(q4) 0 0; 0 0 1 0; 0 0 0 1];
Copy_of_TRACE_PEINTRE(Xr,Q,T01,T12,T23,T34,Robot,0.0005);
```

When no torque is applied to the robot joints, one would initially expect the robot to remain in its stationary state, particularly if it is initially at rest. In the absence of applied torques, the robot should theoretically remain static in its initial configuration. However, this expectation is based on the assumption that no external forces, such as gravity, or disturbances occur. However, we chose to highlight the influence of gravity : thus, the robot, subjected to gravity and with its joints not locked, showed a tendency to move or slowly collapse under the effect of gravity. This behavior depends very much on the configuration of the joints and the distribution of the robot's mass. In this video, we can see that the robot is instable. In this video, we can see that one joint influenced by another like 2,3,4 pendulums but it's normal due to zero torque applied. Video nammed : Zero torque

System response with non zero torque

Now we want to analyse the response of our system if we input a non zero torque. To do this, we have implemented the following code, which calculates the direct dynamic model of the robot at each iteration.

```
    tau=eval(G(-pi/4 ,-pi/4 ,-pi/4 ,-pi/4));
dt=0.01;
k=1000;
q0=[0 -pi/4 -pi/4 -pi/4]';dq0=[0 0 0 0]';ddq0=[0 0 0 0]';ddq0old=[0 0 0 0]';dq0old=[0 0 0 0]';
Q=zeros(4,k);dQ=zeros(4,k);ddQ=zeros(4,k);Xr=zeros(2,k);
for i=1:k
    i
    ddq0old=ddq0;dq0old=dq0;
    Q(:,i)=q0;
    dQ(:,i)=dq0;
    dQQ(:,i)=ddq0;
    ddq0=(eval(H(q0(1),q0(2),q0(3),q0(4)))^-1)*(tau-eval(C(q0(1),q0(2),q0(3),q0(4),dq0(1),dq0
    (2),dq0(3),dq0(4)))*[dq0(1) dq0(2) dq0(3) dq0(4)]'-eval(G(q0(1),q0(2),q0(3),q0(4))));

    dq0=dq0+dt*(ddq0+ddq0old)/2;
    q0=q0+dt*(dq0+dq0old)/2;
    Xr(:,i)=MGD_PEINTRE_SIMPLE(Robot,q0);
end
T01(q1)=[cos(q1) -sin(q1) 0 0;sin(q1) cos(q1) 0 0; 0 0 1 0; 0 0 0 1];
T12(q2)=[cos(q2) -sin(q2) 0 Robot.l1;sin(q2) cos(q2) 0 0; 0 0 1 0; 0 0 0 1];
T23(q3)=[cos(q3) -sin(q3) 0 Robot.l2;sin(q3) cos(q3) 0 0; 0 0 1 0; 0 0 0 1];
T34(q4)=[cos(q4) -sin(q4) 0 Robot.l3;sin(q4) cos(q4) 0 0; 0 0 1 0; 0 0 0 1];
Copy_of_TRACE_PEINTRE(Xr,Q,T01,T12,T23,T34,Robot,0.0005);
```

When a non-zero torque is applied to the joints of a robot, this initiates a movement and the speed of the joints is modified according to the torque applied. Moreover, we can see, in the video, that a control "easy" of the robot that means control of 4 angles is complicated ; in fact each inertia affect a lot the others arms so it is complicated to control this robot. To manage these reactions effectively and maintain the robot's stability, we have chosen to integrate a control system that continuously adjusts the torque to achieve the desired objectives while minimising errors and reacting to external disturbances. video nammed : Torque

Question 10. Consider a desired robot configuration qd. Implement, test and compare the following regulating controllers :

In all this parts, there is some video to show the different methods. The main difference between this 3 codes is the calcul of the torque. In all this simulation, we controlled angle so $\dot{q}d = 0$. Considering now our robot with the full dynamic and our goal is to regulate to obtain asymptotic stabilization in a desired equilibrium state which is a zero velocity and q_d which our desired configuration. So we need to design a control which is able to achieve this task

— First of all we consider a PD regulation with respect to the position error with a gain K_p and K_d with respect to te velocity error K_p and K_d are both positive definite. In order to considerate that we implement this code :

```
1        Kp=[10000 0 0 0;0 10000/2 0 0;0 0 10000/4 0;0 0 0 10000/8];Kd=[500 0 0 0;0 500/2
    0 0;0 0 500/4 0;0 0 0 500/8];
2        tau=Kp*(qd-q0)-Kd*dq0;
3
```

So with this code, we can see that the robot can arrive at the good position, but the error is not null.In our simulation we can see that there are big values of KP due to the compensation of this lack of gravity. Video nammed PD

— PID regulation The PID regulation is very similar, but in the video we can see that this method could perfect reduce the errors. Moreover, we can see that the robot at the end have some problems to compensate the gravity. It's normal because when the robot arrived at the good position the error decrease but not the gravity, one technic for this is to include the gravity (gravity compensation).

```
1        Kp=[10000 0 0 0;0 10000/2 0 0;0 0 10000/4 0;0 0 0 10000/8];Kd=[1000 0 0 0;0
    1000/2 0 0;0 0 1000/4 0;0 0 0 1000/8];Ki=[10000 0 0 0; 0 10000/2 0 0; 0 0 10000/4 0;
    0 0 0 10000/8];
2        res=res+dt*(qd-q0+qd-q0old)/2;
3        tau=Kp*(qd-q0)-Kd*dq0 +Ki*(res);
4
```

Video nammed PID

— PD Gravity regulation In this viedo, we can see that the robot can go at the good position, there are fewer disturbances due to the gravity so it is better. To implement, this method we use this code :

```
1        Kp=[30000 0 0 0;0 10000/10 0 0;0 0 10000/500*10 0;0 0 0 10000/1000*10];Kd=[1000
    0 0 0;0 75 0 0;0 0 7.5 0;0 0 0 0.75];
2        tau=Kp*(qd(:,i)-q0)-Kd*(dqd(:,i)-dq0)+eval(G(q0(1),q0(2),q0(3),q0(4)));
3
```

Video nammed PD gravite

It was pretty fun to go at a desired angle however our goal is to implement trajectory.

Question 11. Consider a desired trajectory for the robot configuration (e.g., a quadratic spline) going from a qd(0) to qd(T), given a trajectory duration T $>$ 0. Implement, test and compare the following controllers to track the desired trajectory :

The goal in this part was to create a part of a circle for the first and give a fix angle to the other arms. It seems easy, however, this includes a good handling of the error due to inertia of each arm. In this part, we could see the difficulties to manage the inertia and so the point of view of FFW and FBL is really important.

— PD controller with gravity compensation In this part, we were really surprised because the command was not good. So we decided to change the coefficient of P and D but it wasn't better. With this controller, we can have a trajectory, but it is difficult to manage all arms correctly. It is totally normal because the effect of all arms of the robot created big perturbations. Video nammed PD gravity trajectory

— PD+FFW We input at the beginning only the PD+FFW, we could see that this method works pretty good. Another method which is a feedback linearization is more attractive because we consider only the desired torque, and we implement the velocity and the Gravity of the robot, so we've got all the advantages of PD+FFW and FBL. So we've implemented this part.

```
1
2 ud=eval(H(qd(1,i),qd(2,i),qd(3,i),qd(4,i)))*ddqd(:,i)+eval(C(qd(1),qd(2),qd(3),qd(4),dqd
    (1),dqd(2),dqd(3),dqd(4)))*[dqd(1) dqd(2) dqd(3) dqd(4)]'+eval(G(qd(1),qd(2),qd(3),qd
    (4)));
3 tau=ud+eval(H(qd(1,i),qd(2,i),qd(3,i),qd(4,i)))*(Kp*(qd(:,i)-q0)-Kd*(dqd(:,i)-dq0)+eval(G
    (q0(1),q0(2),q0(3),q0(4))));
```

— Feedback linearization (FBL) + [PD+FFW] So as we said before, as we can see in the video it works really good. Video=FBLPDFFW.

```
1 tau=eval(H(qd(1,i),qd(2,i),qd(3,i),qd(4,i)))*(ddqd(:,i)+Kp*(qd(:,i)-q0)-Kd*(dqd(:,i)-dq0
    )) +eval(C(q0(1),q0(2),q0(3),q0(4),dq0(1),dq0(2),dq0(3),dq0(4)))*[dq0(1) dq0(2) dq0
    (3) dq0(4)]' +eval(G(q0(1),q0(2),q0(3),q0(4)));
```

Video nammed FBL PD FFW

4 Task control of a dynamic robot

Question 12. Compute a simple trajectory for the end-effector task, e.g., a line or a circle in the robot workspace. You should have the analytical expression of r(t) for t [0, . . . , T], such that you can compute the corresponding derivatives (see point 1.4)

This part was not very different as the question 1.4, we 've implemented the same trajectory.

Question 13. Extend point 1.5.(a) at the second order level (at the acceleration level).

In this way you should be able to compute the desired joint accelerations $\dot{\dot{q}}^d$ to track the desired end-effector trajectory. In order to do that, we've to understand what are the difference between the question 1.5.(a). In the question 1.5, we've calculated $\dot{q} = J\dot{X}$. So now, we've got $\dot{\dot{X}}$. Or $\dot{q} = \dot{J}\dot{X} + J\dot{\dot{X}}$. Or $\dot{\dot{X}} = \dot{J} * \dot{q} + J * \dot{\dot{q}}$, so

$$\dot{\dot{q}} = J(\dot{\dot{X}} - \dot{J} * J\dot{X})$$

. We've to integrate this expression twice, and we can access to $\dot{\dot{q}}d$, $\dot{q}d$ and qd.

Question 14. Integrate $\dot{\dot{q}}^d$ twice to obtain the desired joints velocities and position// We've to integrate this expression twice, and we can access to $\dot{\dot{q}}d$, $\dot{q}d$ and qd. In this part we've developed the code dynamics non optimize cinematic inverse trajectory

```
1 \Jinv=pinv(J);
2 ddq_old=ddq;dq_old=dq_int;
3 ddq=Jinv*(ddX-H*Jinv*dX(:,i)); %H is the derivative of the jacobian
```

Unfortunately, this code doesn't work due to a big complexity, it takes a lot of time due to calcul of syms. So we've decided to use the initial code and derive $\dot{q}d$. It was easier in terms of calcul.

Question 15. Use the previously computed desired trajectory as the input for the dynamic controller developed in point 2.4.

Question 16. Test and comment the results in terms of tracking error
It takes a long time to calculate. This is the reason why we've decided to do a part of a circle. We can see on the video that the robot has a problem of inertia, because it does many moves between each steps due to inertia. So in the real life, we have maybe to linearize the solution in order to not have a peak of current for each controller. However, we can see that the robot succeed to perform the task. The task is not perfect as the kinematic approach. It is totally normal, in fact here we are more in a real life. There is a video for this part too named part of a circle. I advise to not simulate this part because we simulated it during 20min.

5 Conclusion

To conclude, on the project we can see that we can have different approach, however they are similar. First, we can be considerate the kinematic approach we could see that there are many controls in order to avoid obstacles or to do many tasks. On the other hand, we discover the dynamic approach which is more complex, but we have to consider it into real robots. Finally, we've seen that the first methods which seems great are available for a dynamic approach. So it is fascinating to control a robot in a point of view of dynamics, however it is more complicated.