ROB315
Modélisation et Commande des Robots Manipulateurs

TP1 - Direct and inverse kinematics

TP2 - Dynamics and control

Mateus LOPES RICCI
Matheus MELO MONTEVERDE

26 janvier 2021

Table des matières

Table des figures

1 Introduction

We propose to study the geometric and kinematic modeling of a manipulator arm developed by the Interactive Robotics Laboratoryof the CEA List (Figure 1.1). This robot, which kinematic chain is of serial type, has 6 revolute joints (j_i with i = 1,...,6).

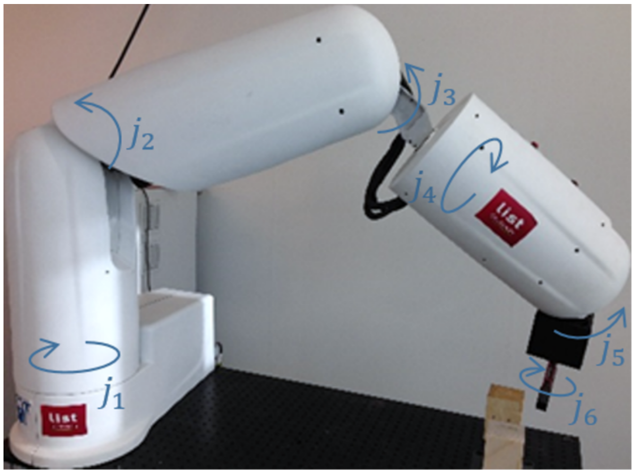


Figure 1.1 – Manipulator arm developed by the Interactive Robotics Laboratory of the CEA List

The numerical values of the robot parameters are specified in the table below.

Parameters	Numerical values	Type of parameter
d_3	0.7m	Geometric parameter
r_1	0.5m	Geometric parameter
r_4	0.2m	Geometric parameter
r_E	0.1m	Geometric parameter

Table 1 – Robot parameters

The main $MatLab^{TM}$ functions used to design the robotic arm in this project can be visualized in Appendix A.

2 Direct geometric model

2.1 Q1. Robot's geometry

First the geometry of the robot was established, identifying the frame of each of its joints according to the MDH convention. In the figure below, the frames attached to each link of the manipulator robot can be seen.

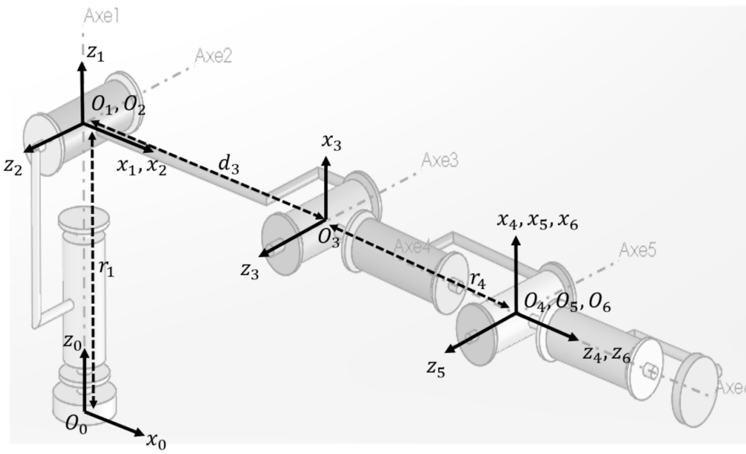


FIGURE 2.1 – Description of the robot's geometry

All robot joints are revolute joints, which means that the relative motion between two bodies is reduced to a pure rotation along a common axis.

2.2 Q2. Geometric parameters

In order to describe the working space of the manipulator arm, each of its joints has its associated geometrical parameters. These parameters can be seen in table 2.

Joint (i)	α_i	$\mathbf{d}_i$	θ_i	$\mathbf{r}_i$
1	0	0	q_1	r_1
2	$\pi/2$	0	q_2	0
3	0	d_3	$q_3 + \pi/2$	0
4	$\pi/2$	0	q_4	r_4
5	$-\pi/2$	0	q_5	0
6	$\pi/2$	0	q_6	0
E	0	0	0	r_E

TABLE 2 – Geometric parameters of the robot

The parameters d and r represent the distance of the j_i joint from the j_{i-1} one in the direction x and z respectively. Meanwhile, the parameters α and θ represent the possible angles of rotation of each joint in relation to the x and z axes respectively.

2.3 Q3. Direct Geometric Model (DGM) of the robot

First, in order to calculate the homogenous transformation matrix $\overline{g}$ between two successive frames, a MatLab function `TransformMatElem(`α_i`,`d_i`,`θ_i`,`r_i`)` has been defined using the following formula.

$$\begin{aligned}\overline{g}_{(i-1)i} &= R_x, \alpha_i Trans(x, d_i) R_z, \theta_i Trans(z, r_i) \\ &= \begin{pmatrix} cos(\theta_i) & -sin(\theta_i) & 0 & d_i \\ cos(\alpha_i)sin(\theta_i) & cos(\alpha_i)cos(\theta_i) & -sin(\alpha_i) & -r_i sin(\alpha_i) \\ sin(\alpha_i)sin(\theta_i) & sin(\alpha_i)cos(\theta_i) & cos(\alpha_i) & r_i cos(\alpha_i) \\ 0 & 0 & 0 & 1 \end{pmatrix} \\ &= \begin{pmatrix} R_{(i-1)i} & p_{(i-1)i} \\ 0_{1x3} & 1 \end{pmatrix}\end{aligned}$$

where $R_{(i-1)i}$ is the rotation matrix representing the frame $R_{(i-1)} = (O_{(i-1)}x_{(i-1)}y_{(i-1)}z_{(i-1)})$ with respect to the frame $R_i = (O_i x_i y_i z_i)$ and $p_{(i-1)i}$ is the column vector that represents the translation from $O_{(i-1)}$ to O_i.

Then a new function `ComputeDGM(`α`,`d`,`θ`,`r`)` was developed to compute the direct geometry model of the system. This function aims to calculate the $\overline{g}_{0N}$ transformation matrix through which it is possible to find the geometrical parameters of a given frame of the manipulator arm link chain. This transformation matrix is obtained by the following recursive formula.

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

where q is the vector of joint variables.

Finally, it was desired to calculate the position and orientation of the robot's end-effector. Thus, in addition to the geometrical parameters of the system, the end-effector geometrical parameters (seen in the last line of table 1) were used as the input arguments of the GDM function. The function `TransformMatElem(`α_i`,`d_i`,`θ_i`,`r_i`)` was used within function `ComputeDGM(`α`,`d`,`θ`,`r`)` to calculate the homogenous transformation matrix of each frame.

2.4 Q4. Position and orientation of end-effector frame

Now the position and orientation of the end-effector frame is to be calculated given two joint configurations $q_i = [-\frac{\pi}{2}, 0, -\frac{\pi}{2}, -\frac{\pi}{2}, -\frac{\pi}{2}, -\frac{\pi}{2}]^t$ and $q_f = [0, \frac{\pi}{4}, 0, \frac{\pi}{2}, \frac{\pi}{2}, 0]^t$. To this end the function to compute the DGM, created in the previous section, is implemented.

2.4.1 For configuration $\mathbf{q}_i$

$$P = [-0.10 \quad -0.70 \quad 0.30]^T$$
$$n = [-0.58 \quad -0.58 \quad 0.58]^T$$
$$\phi = 2.09$$

2.4.2 For configuration $\mathbf{q}_f$

$$P = [0.63 \quad -0.10 \quad 1.13]^T$$
$$n = [0.28 \quad 0.68 \quad -0.68]^T$$
$$\phi = 2.59$$

2.5 Q5. Visualization of position and orientation of end-effector frame

Through the position and orientation results obtained for the end-effector frame in the last section, for both proposed configurations, it was possible to develop a function `PlotFrame(q)` to better visualize the frames R_0 and R_E. The graphs generated by this new function are presented below. The dashed cyan line represents the translation from O_0 to O_E.

2.5.1 For configuration $\mathbf{q}_i$

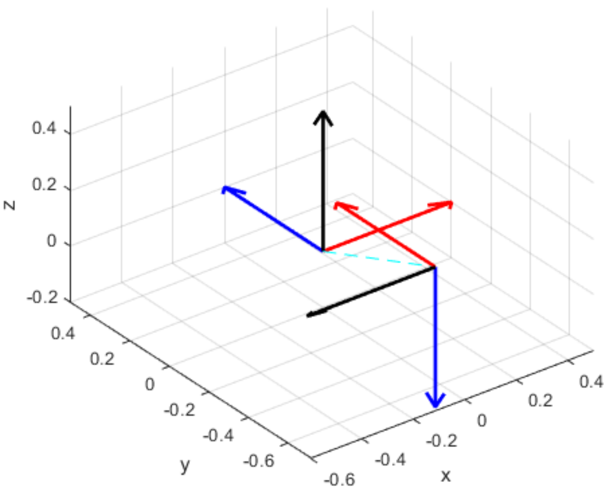


FIGURE 2.2 – Frames R_0 and R_E for the configuration q_i

2.5.2 For configuration $\mathbf{q}_f$

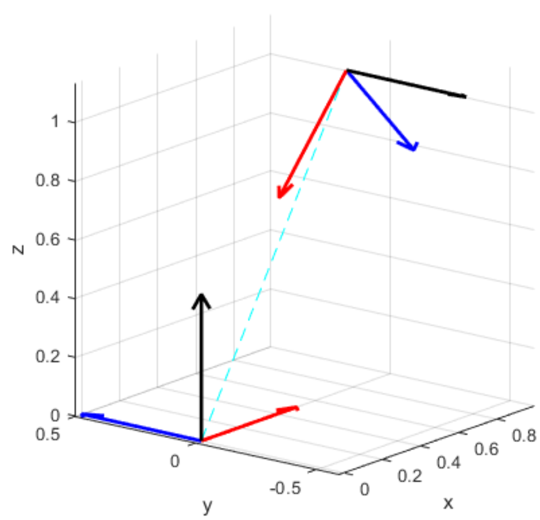


FIGURE 2.3 – Frames R_0 and R_E for the configuration q_f

3 Direct kinematic model

3.1 Q6. Twists calculation by the Jacobian matrix

In this question it is desired to study the direct kinematic model (DKM) of a robot, by which it is possible to determine the velocity of the end-effector $\dot{X}$ according to the velocities of the joints $\dot{q}$. This is done by the following equation.

$$\dot{X} = J(q)\dot{q} \tag{2}$$

where $J(q)$ denotes the Jacobian matrix given by $\frac{\partial X}{\partial q}$.

In order to calculate the linear and angular twist components of the velocity of a given frame (equation 3.1.2), the Jacobian matrix of equation 2 can be written by equation 5.

$$^0\mathcal{V}_{0,E} = \begin{bmatrix} ^0V_{0,E}(O_E) \\ ^0\omega_{0,E} \end{bmatrix} = \begin{bmatrix} ^0J_v(q) \\ ^0J_\omega(q) \end{bmatrix} \dot{q} = \, ^0J(q)\dot{q} \tag{3}$$

$$^0J(q) = [^0J_0(q), ...,^0 J_i(q), ...,^0 J_N(q)] \tag{4}$$

where $^0J_i(q)$ is the cartesian velocity due to the ith joint given in the $\mathcal{R}_0$ frame.

The $^0J_i(q)$ of each joint can be calculated as the following :

$$^0J(q) = \begin{cases} \begin{bmatrix} R_{0i}Z_i \\ 0_{3x1} \end{bmatrix} & \text{if the i}^{th} \text{ joint is prismatic} \\ \\ \begin{bmatrix} R_{0i}(Z_i \times p_{iN}) \\ R_{0i}Z_i \end{bmatrix} & \text{if the i}^{th} \text{ joint is revolute} \end{cases}$$

As in the case under study all joints are revolute, the calculation of all the Jacobian matrices will be performed by the second method, where R_{0i} is the rotation matrix of frame R_i, Z_i is the unit vector of the ith joint and p_{iN} is the position vector from origin of frame R_i.

Thus a function `ComputeJac(α,d,θ,r)` was created to calculate the Jacobian matrix $^0J(q)$, by which it was possible to find the twists on the end-effector frame, using the q_i and q_f of the previous sections in addition to the joint velocities $\dot{q}$ = [0.5,1.0,0.5,0.5,1.0,0.5].

3.1.1 For configuration $\mathbf{q}_i$

$$^0\mathcal{V}_{0,E} = [0.35,\ -0.10,\ 0.60,\ -0.00,\ -1.00,\ 0.00]^T$$

3.1.2 For configuration $\mathbf{q}_f$

$$^0\mathcal{V}_{0,E} = [-0.55,\ 0.32,\ 0.46,\ 1.06,\ 0.00,\ 0.15]^T$$

3.2 Q7. Transmission of velocities

In this question the study of the transmission of velocities between the joint and the task space is proposed, only taking into account the linear velocity component ${}^{0}J_v(q)$. Therefore, the decomposition in singular values is used to calculate the velocities by the following equation :

$$J = U\Sigma V^t \tag{5}$$

where the columns of U represent the principal axis of the ellipsoid and Σ contain the singular values σ_i of ${}^{0}J_v(q)$, which represent the length of the ellipsoid at each direction U_i.

It is also possible to calculate the preferred direction of the ellipsoid by taking the first column of U. In order to calculate the e velocity manipulability of a certain configuration, it is necessary to find the volume of the ellipsoid, which is done by the equation below.

$$\mathcal{W} = \sqrt{det(J(q)J^t(q))} = \prod_{i=1}^{r} \sigma_i \geq 0 \tag{6}$$

3.2.1 For configuration $\mathbf{q}_i$

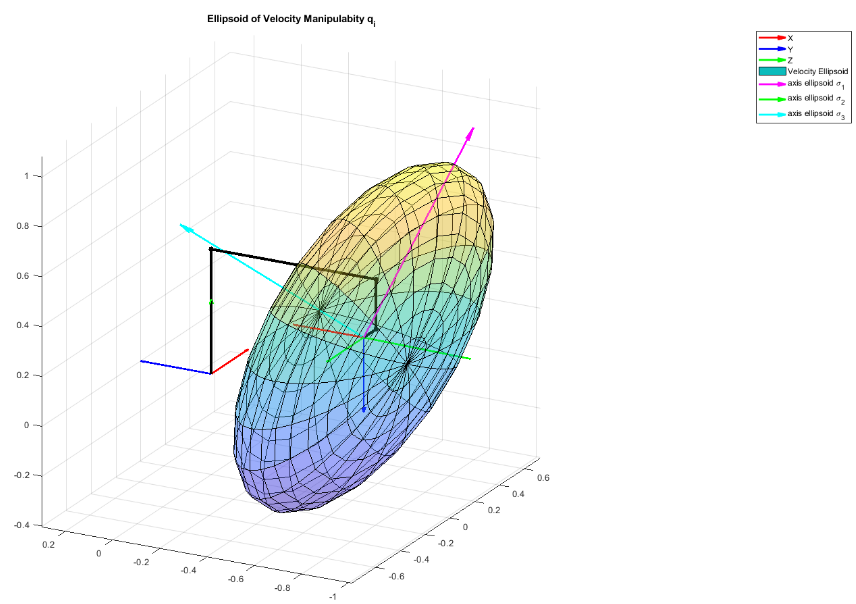


FIGURE 3.1 – Ellipsoid of Velocity Manipulability for q_i

$$\mathcal{W} = 0.1116 \qquad \text{and} \qquad U_1 = [-0.7114,\ -0.0975,\ 0.6959]^T$$

3.2.2 For configuration $\mathbf{q}_f$

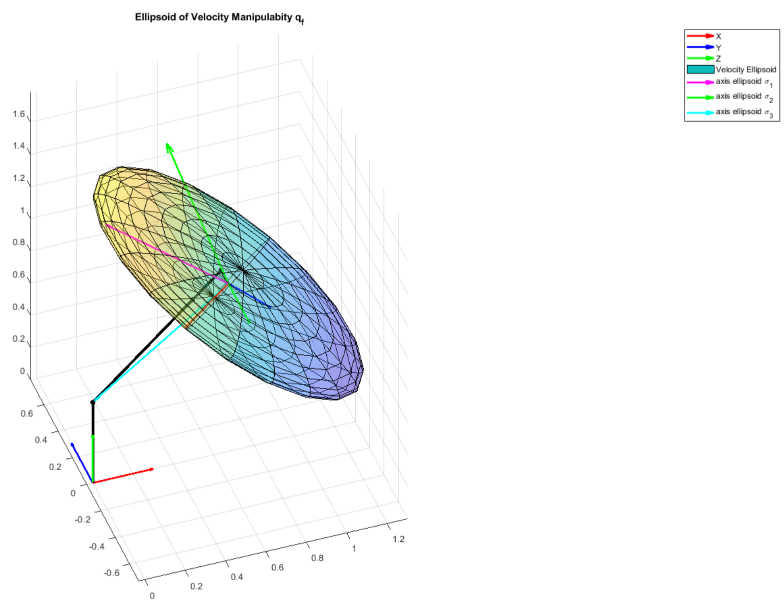


FIGURE 3.2 – Ellipsoid of Velocity Manipulability for q_f

$$\mathcal{W} = 0.0590 \qquad \text{and} \qquad U_1 = [0.3760,\ -0.3263,\ 0.8715]^T$$

4 Inverse geometric model

4.1 Q8. Joint configuration by inverse geometric model

In this question the inverse geometrical model will be analysed by which it is possible to find a joint configuration q that allows the robot to position its tool in a desired position X.

To do this two different methods will be studied, the Newton-Raphson method and the Gradient method. The first one uses a Taylor series approximation to find the best solution. The second aims to minimize an objective-function H(q).

To implement both methods the function `q*=ComputeIGM(`X_d`,`q_0`,`k_{max}`,`ϵ_x`)` was developed based on the pseudo-algorithm illustrated by figure 4.1. X_d, q_0, k_max and ϵ_x represent, respectively, the desired task position, the initial configuration, the maximum number of iterations and the norm of the tolerated Cartesian error. The last two parameters are used as stop criteria.

To study the performance of this model, two sets of parameters were proposed as seen below :

1. $X_d = X_{di}$ = (-0.1, -0.7, 0.3)^t, q_0 = [-1.57,0.00,-1.47,-1.47,-1.47,-1.47,-1.47], k_{max} = 100, ϵ_x = 1mm
2. $X_d = X_{df}$ = (0.64, -0.10, 1.14)^t, q_0 = [0, 0.80, 0.00, 1.00, 2.00, 0.00], k_{max} = 100, ϵ_x = 1mm

$k \leftarrow 0$
while $\left\| J^t(q_k) \right\| > \epsilon$ **do**
$\quad k \leftarrow k + 1$
$\qquad \qquad \triangleright$ *Case of the Gradient-based method*
$\quad q_k \leftarrow q_{k-1} + \alpha J^t(q_{k-1}) \left[X_d - f(q_{k-1}) \right]$
$\quad \triangleright$ *Case of the Newton-Raphson-based method*
$\quad q_k \leftarrow q_{k-1} + J^{-1}(q_{k-1}) \left[X_d - f(q_{k-1}) \right]$
end while
$q^* \leftarrow q_{k+1}$
return (q^*)

FIGURE 4.1 – Pseudo-algorithm for computing the inverse geometric model

The following table shows the results for each case using both methods mentioned earlier.

Configuration	Method	Error (mm)	CPU time (ms)
q_i	Newton-Raphson	$5.59 \cdot 10^{-4}$	6.45
q_i	Gradient	$9.23 \cdot 10^{-4}$	7.79
q_f	Newton-Raphson	$4.16 \cdot 10^{-4}$	10.98
q_f	Gradient	$57.00 \cdot 10^{-4}$	19.50

TABLE 3 – Geometric parameters of the robot

From the results obtained, it can be seen that for both cofigurations the Newton-Raphson method had a lower error and a lower CPU time than the gradient method. Thus, for the development of the following questions, the method used to calculate the Inverse Geometric Model will be that of Newton-Raphson.

5 Inverse kinematic model

5.1 Q9. Following a trajectory with inverse kinematic model

The next step is to impose a desired trajectory to be followed by the end effector. In this case, we will impose as trajectory a straight line from the initial position, X_{di}, to the final position, X_{df}. The rectilinear motion is carried out at a constant speed $V = 1m/s$ and is sampled at period $T_e = 1\ ms$.

The initial condition and final condition are :

1. X_{di} = (-0.10, -0.70, 0.30)^t
2. X_{df} = (0.64, -0.10, 1.14)^t

The trajectory is discretized in n points with a corresponding desired position Xdk.

The discretization is done as follows :

$$temp = \frac{||X_{df} - X_{di}||}{V}$$

$$n = ceil(temp/Te)$$

Then, we determine the setpoints values q_{dk} utilizing the function `q* = ComputeIGM(`X_d,q_0,k_{max},ϵ_x`)`, developed in question **8**. We utilize as the initial approximation for q_{dk}, $q_0 = q_{d(k-1)}$.

The values computed for q_{dk} are saved in a vector and the sequence of positions of the frame of the end frame was plotted. The figure below shows the trajectory perfomerd :

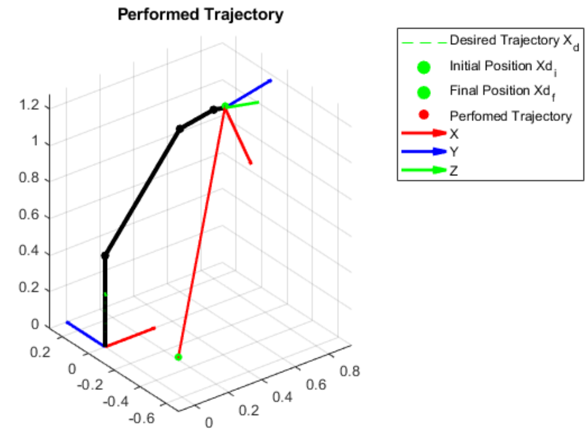


FIGURE 5.1 – Trajectory of the end effector

5.2 Q10. Temporal evolution of the joint variables

In this question, the temporal evolution of each joint variables q_1 to q_6 has been plotted, the maximum values (q_{max}) and minimum values (q_{min}) limitations for each one were overlayed in the plot.

$$q_{min} = [-\pi, -\frac{\pi}{2}, -\pi, -\pi, -\frac{\pi}{2}, -\pi]^t$$

$$q_{max} = [0, \frac{\pi}{2}, 0, \frac{\pi}{2}, \frac{\pi}{2}, \frac{\pi}{2}]^t$$

The following figure shows the temporal evolution of each variable.

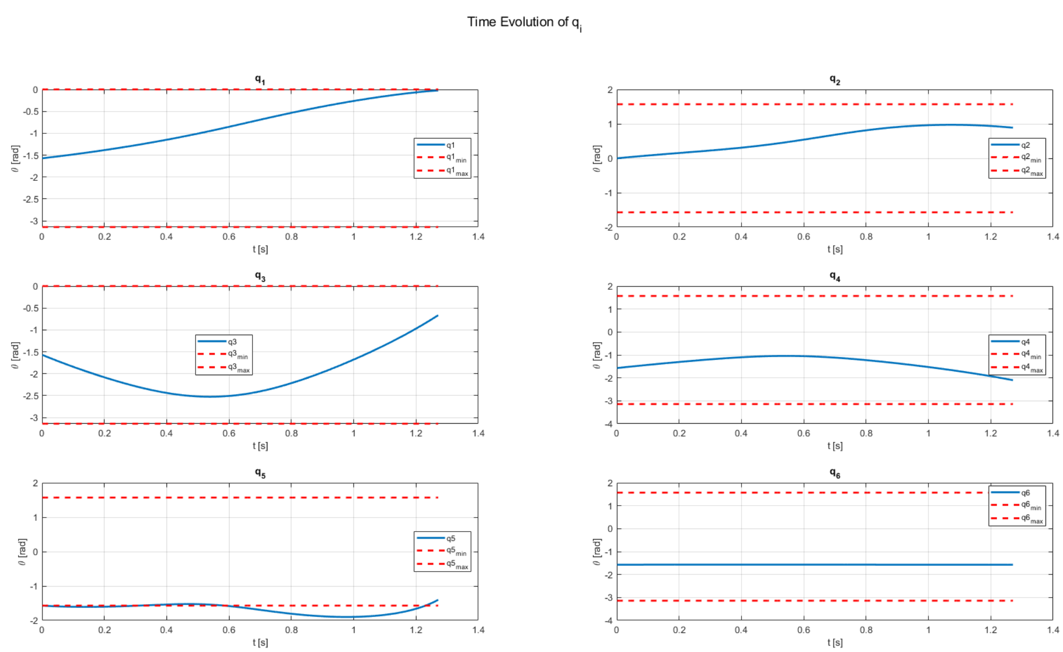


FIGURE 5.2 – Temporal evolution of q_i

5.3 Q11. Secondary task with inverse kinematic model

Given that in the previous question we saw that the angular limitations of each board were not very well respected, it is now desired to compute the inverse cinematic model taking into account a secondary task. Thus, a new function `ComputeIKMlimits(`X_{di}`,`X_{df}`,`V`,`Te`,`q_i`,`q_{min}`,`q_{max}`)`, based on the one created for question Q9, would limit the joints to approach the maximum and minimum values allowed. To that end, the following equation can be used to find the solution to a problem of the reverse kinematic model of a redundant case :

$$\dot{q}^* = J^{\#}\dot{X}_d + (I_n - J^{\#}J)\dot{q}_0 \tag{7}$$

where the first term of the sum is the particular solution whereas the second is the orthogonal projection of $\dot{q}_0$ in the null space (represents the internal motion in the joint space).

To calculate the preferred velocity, the projected gradient technique was used and is expressed by the equation below :

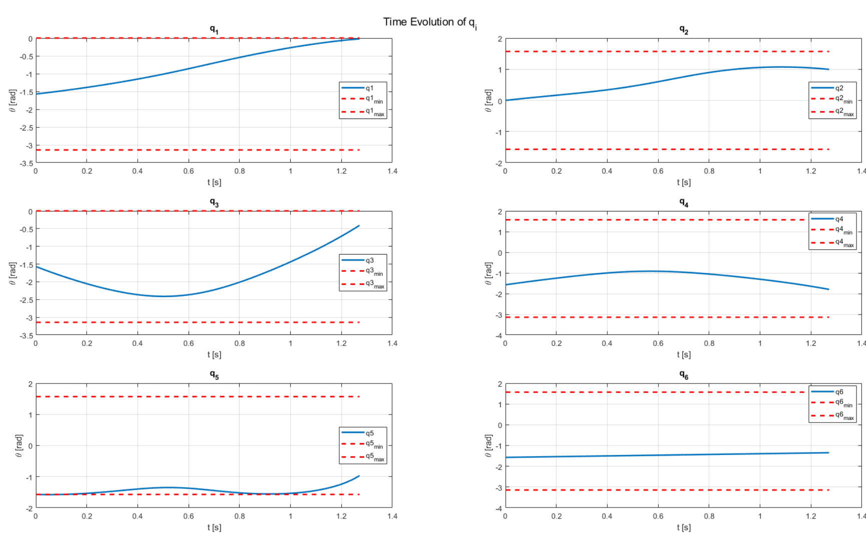
$$\dot{q}_0 = -\alpha \nabla_q H(q) = \begin{pmatrix} \frac{\partial H}{\partial q_1} \\ \vdots \\ \frac{\partial H}{\partial q_n} \end{pmatrix} \tag{8}$$

where $H(q)$ is the objective function and α is the tradeoff between the minimisation objectives of $\frac{1}{2}\left\|\dot{q}\right\|^2$ and H(q).

Since, in this case, it is desired to limit the maximum and minimum angular rotation of the joints during the arm manipulation, the following objective function was used :

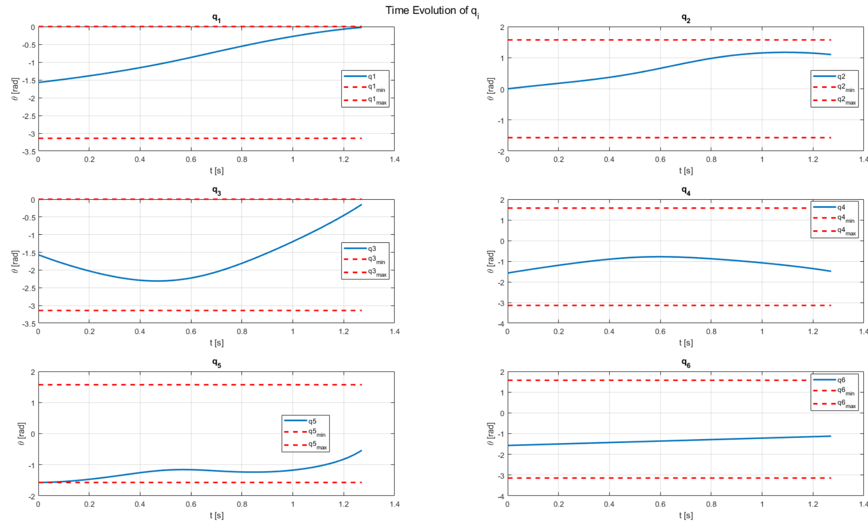
$$H_{lim.}(q) = \sum_{i=1}^{n}\left(\frac{q_i - \overline{q}_i}{q_{max} - q_{min}}\right)^2 \quad \text{where} \quad \overline{q}_i = \frac{q_{max} - q_{min}}{2} \tag{9}$$

By applying this new function to the same case performed in the previous question, new curves are obtained with respect to the angles of each joint in time. These curves can be seen in the figure below.



(a) Trade-off $\alpha = 0.5$, error $\epsilon = 1.46$

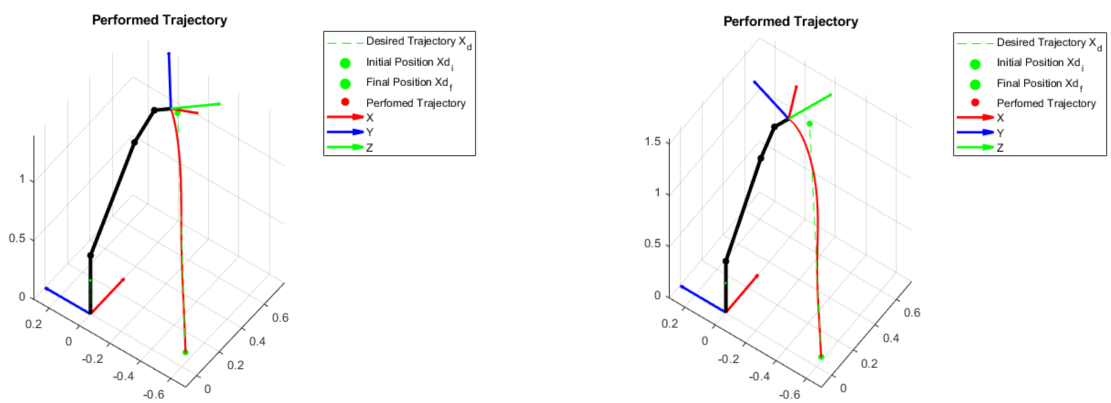
FIGURE 5.3 – Time Evolution for the angle of the joints for two different configurations



(b) Trade-off $\alpha = 1.0$, error $\epsilon = 3.40$

FIGURE 5.3 – Time Evolution for the angle of the joints for two different configurations

The error between the achieved trajectory and the target trajectory was calculated for two different α values and the trajectory of the robotic arm is shown in the figure below.



(a) Trade-off $\alpha = 0.5$, error $\epsilon = 1.46$ (b) Trade-off $\alpha = 1.0$, error $\epsilon = 3.40$

FIGURE 5.4 – Trajectory of the end effector with a imposed secondary task

By analysing figures 5 and 6, we can see the influence of the value of tradeoff on the results. While the tradeoff value is low ($\alpha = 0.5$), the angular restrictions are not well respected but the new trajectory has a minor error compared to the target trajectory. On the other hand, for the tradeoff = 1, the error between the trajectories increases but the angular variables of each joint do not exceed the established limits.

6 Dynamic model

Now it is proposed to study the dynamic model of the robotic arm. The matrix form of the inverse dynamic model for rigid robot manipulator is presented below :

$$A(q)\ddot{q} + C(q,\dot{q})\dot{q} + G(q) + \Gamma_f(\dot{q}) = \Gamma \tag{10}$$

where $A(q)$ is the inertia matrix, $C(q,\dot{q})\dot{q}$ is the vector of joint torques due to the Coriolis and centrifugal forces, $G(q)$ is the vector of joint torques due to gravity, $\Gamma_f(\dot{q})$ is the vector of joint friction torques, Γ is the vector of the joint torques and q, $\dot{q}$ and $\ddot{q}$ are respectively the vectors of joint positions, velocities and accelerations

6.1 Q12. Velocity of the center of mass and rotation speed of all the rigid bodies

First of all, the velocity of the centre of mass and the speed of rotation of each rigid body of the system are calculated. To do this a new function `ComputeJacGi(`α`,d,`θ`,r,`x_G`,`y_G`,`z_G`)` has been created. Within this function the distance between the centre of mass of a rigid body and the frame R_0 is first calculated and then the Jacobian matrix of this position is computed. Then from the Jacobian matrix the velocity of the centre of mass and the speed of rotation of the target body are obtained following the same method as in question **Q6**.

$${}^0J_{Gi}(q) = \begin{bmatrix} {}^0J_{v_{Gi}}(q) \\ {}^0J_{\omega_{Gi}}(q) \end{bmatrix} \tag{11}$$

6.2 Q13. Inertia Matrix of the robot

A function to calculate the inertial matrix of the robot was then created. In order to perform its calculation, the following formula was applied.

$$A(q) = \sum_{i=1}^{N} (m_i \, {}^0J^t_{Gi}(q) \, {}^0J_{Gi}(q) + \, {}^0J^t_{\omega i}(q) \, {}^0I_i \, {}^0J_{\omega i}(q)) \tag{12}$$

where ${}^0J_{Gi}$ is the velocity of the centre of mass of the body, ${}^0J_{\omega i}$ is the speed of rotation of the body, m_i is the mass of the body and ${}^0I_i$ is the inertia tensor evaluated at the centre of mass of the body transported to the frame R_0.

To calculate the matrix ${}^0I_i$, the Generalized Huygens theorem was used to first calculate ${}^0I_{Gi}$. Equation 13 represents this theorem.

$$I_{Oi} = I_{Gi} + m_i \begin{bmatrix} XX_i & XY_i & XZ_i \\ YX_i & YY_i & YZ_i \\ ZX_i & ZY_i & ZZ_i \end{bmatrix} \tag{13}$$

Then the I_{Gi} matrix was transported to the frame R_0, ${}^0I_i = R_0 I_{Gi} R_0^T$. Finally the motor inertia of each actuator calculated by $r^2_{red} * J_m$ is added the diagonal of matrix $A(q)$.

6.3 Q14. Lower and upper bounds of the inertia matrix

Since it is possible to calculate the inertial matrix of the robot for a given configuration of joint angles q, an `A_bounds(`q_{min}`, `q_{max}`, `n_joints`)` function has been developed to verify the maximum and minimum value of A within the angular restrictions q_{min} and q_{max}. Thus $A(q)$ can be defined as follows :

$$\mu_1 \leq A(q) \leq \mu_2$$

As the robot is composed only by revolute joints, μ_1 and μ_2 values are constant and do not depend on the joint configuration. To find these boundary values, a discretization is made for each joint angle q_i of the robot, within the angle restrictions, and the value of $A(q)$ is calculated for several combinations of joint angle configurations.

The minimum and maximum values of $A(q)$ are respectively :

$$\mu_1 = 0.0589 \qquad \text{and} \qquad \mu_2 = 10.1985$$

6.4 Q15. Gravitational torque

The next step is to calculate the torque due to the gravitational force $G(a)$by function `ComputeGravTorque(q)`. Through the gradient of the potential energy E_p due to gravitational force, the gravitational torque can be expressed by the following equation.

$$G(q) = -({}^0J_{vG1}^t(q)m_1g + ... +^0 J_{vG6}^t m_6 g) \tag{14}$$

where ${}^0J_{vGi}$ is the velocity of the centre of mass of the body, m_i is the mass of the body and g is the gravity vector.

6.5 Q16. Upper bound of the gravitational torque

In this question the maximum value of gravitational torque was checked among the various possibilities of angular configurations q of the robot joints. This was done through the function `G_bound(`q_{min}`,` q_{max}`,` n_joints`)` and the method used was the same as the function `A_bounds(`q_{min}`,` q_{max}`,` n_joints`)`. As a result, the maximum value of $G(q) = 116.8160$ was obtained.

6.6 Q17. Inverse dynamic model

Then the last function `ComputeFrictionTorque(`$\dot{q}$`)` was created. It returns the torque due to friction of the joints, which can be calculated from equation 15.

$$\Gamma_{f_i}(q_i) = diag(\dot{q}_i)F_{vi} \tag{15}$$

where $\Gamma_{f_i}(q_i)$ is the torque due to friction, $\dot{q}_i$ is the velocity vector of the joint i and F_{vi} is the joint viscous frictions.

Now, from all the functions created above, together with function `ComputeCCTorques(`q`,` $\dot{q}$`)` which calculates the torque of the joints due to the *Coriolis* and centrifugal effects, already provided, it was possible to develop in *Simulink*^{TM} the inverse dynamic model (equation 10) which has as input the total torque of the system applied by the motors and as output the vector acceleration of the joints.The block built can be seen in the figure below.

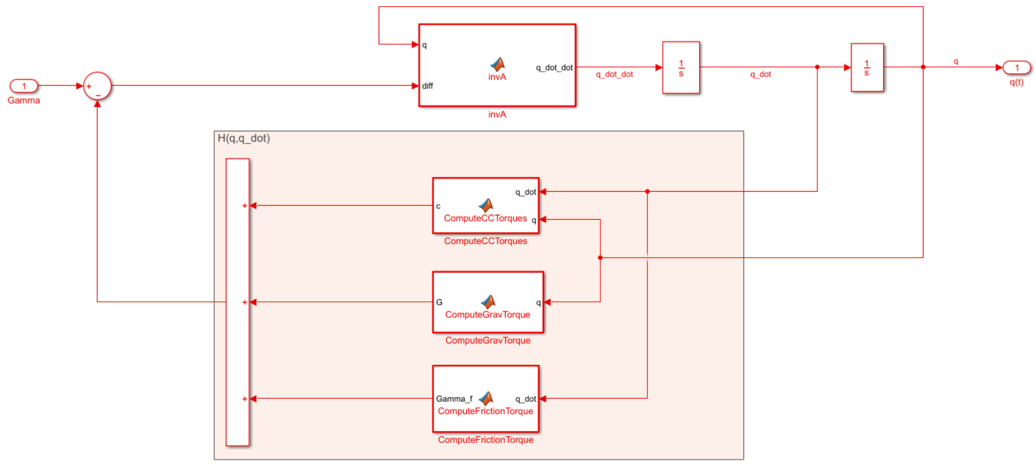


FIGURE 6.1 – Dynamic System Block Diagram

7 Trajectory generation in the joint space

7.1 Q18. Minimal time to follow trajectory

Given a starting configuration q_{d_i} and a final configuration q_{d_f} it is desired to calculate the minimum time t_f required to follow a polynomial trajectory of degree 5 in the joint space, performed at zero initial and final velocities and accelerations. The time is sampled at a period T_e = 1ms. Both initial and final configurations are as follow :

— 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

To calculate the angular configuration of the robot as a function of the time $q(t)$, the following expressions can be used :

$$q(t) = q_{d_i} + r(t)D \tag{16}$$

$$r(t) = 10(\frac{t}{t_f})^3 - 15(\frac{t}{t_f})^4 + 6(\frac{t}{t_f})^5 \tag{17}$$

where $D = q_{d_f} - q_{d_i}$

Taking into account only the k_{a_j} vector of maximum acceleration of the joints (calculated using equation 19), the minimum time T_{f_i} of the j^{th} joint to reach the final configuration is given by the equation below.

$$t_{f_i} = \sqrt{\frac{10\ |D|}{\sqrt{3}k_{a_j}}} \tag{18}$$

$$k_{a_j} = \frac{\tau_{max,j}\ r_{red,j}}{\mu_2} \tag{19}$$

where $\tau_{max,j}$ is the maximal torque of joint j and $r_{red,j}$ is the reduction factor of joint j.

Applying all the values provided by the problem in the above equations results in the minimum time $t_{f_i} = 0.4102\ s$.

7.2 Q19. Generating a desired joint trajectory point

Now through *Simulink*^{TM} the previous equations are implemented within an function that generates the trajectory q_c = `GenTraj(`q_{d_i}`,`q_{d_f}`,`t`)`, having as input the initial and final configuration of the joints and the minimum time. The diagram created can be seen as the following.

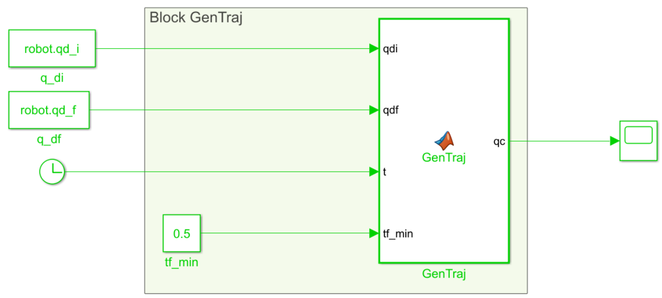


FIGURE 7.1 – Trajectory Generation Block Diagram

Using q_{d_i} and q_{d_f} from the previous question and $t_f = 0.5$, the angles of each joint j as a function of time, for the trajectory found, are presented below.

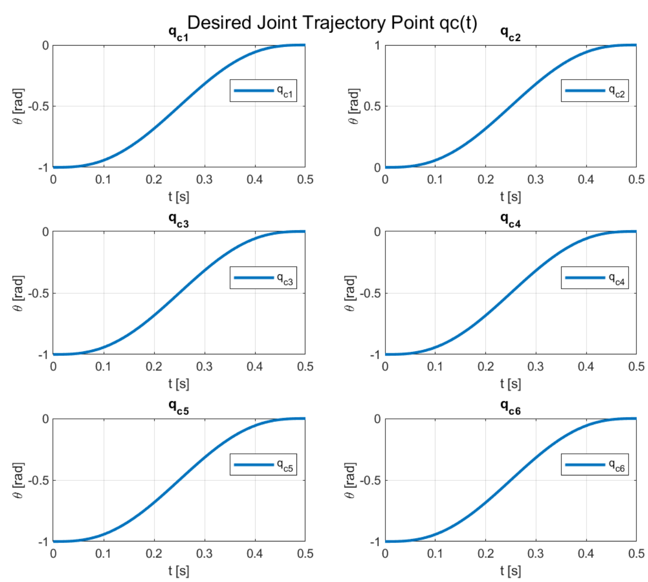


FIGURE 7.2 – Evolution of Joint Trajectories $q_i(t)$

7.3 Q20. P.D. controller with gravity compensation

Finally, in order to finish the robotic arm project, a block to perform the position control is added to the system. This is done through a P.D controller with gravitational compensation, and can be described by the following expression :

$$\Gamma = K_p(q_d - q) + K_d(\dot{q}_d - \dot{q}) + \hat{G}(q) \tag{20}$$

The controller block built in *Simulink*^{TM} is schematized below.

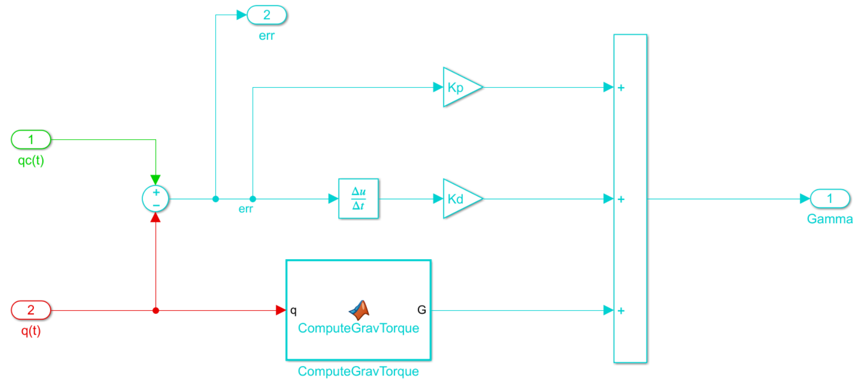


FIGURE 7.3 – Control Block Diagram

Now, by linking this block to the inverse dynamic model block and the trajectory generation block, we obtain the following system :

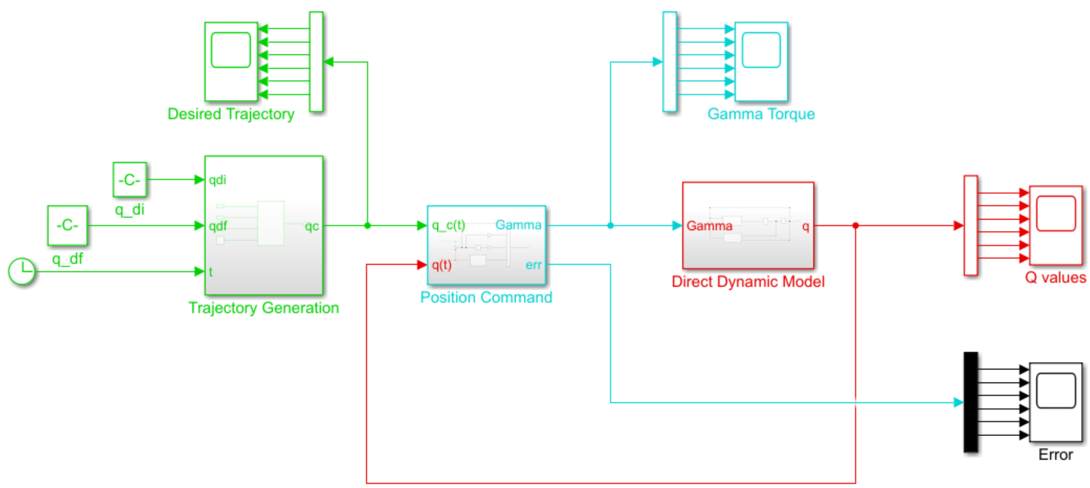


FIGURE 7.4 – Position-Control block Diagram

Through the analysis of velocity, acceleration, error and admissible torques, it was possible to define the values of K_d and K_p of the controller as the following :

— $q_{d_i} = [5000, 8000, 5500, 800, 5500, 2500]^t$
— $q_{d_f} = [1000, 1000, 500, 100, 100, 500]^t$

With the implementation of these values on the controller it was possible to obtain the graphs below regarding the temporal evolution of joint trajectories $q_i(t)$, tracking errors e (t), and control joint torques $\Gamma_i(t)$ of each of the robot joints.

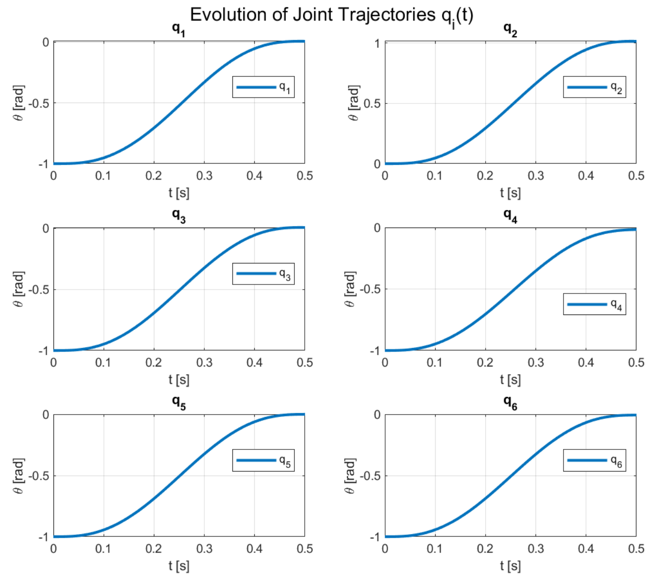


FIGURE 7.5 – Evolution of Joint Trajectories $q_i(t)$

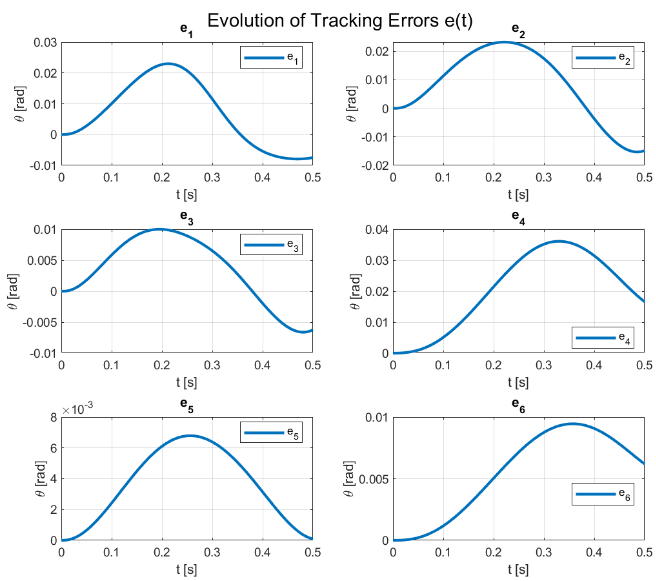


FIGURE 7.6 – Evolution of tracking errors $e_i(t)$

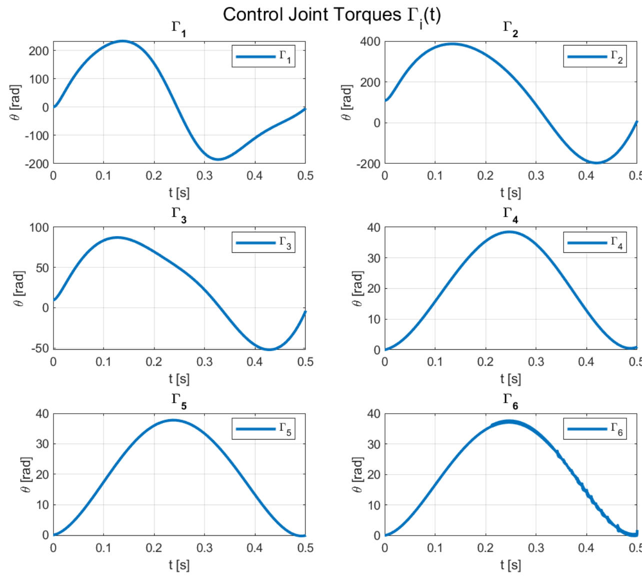


FIGURE 7.7 – Evolution of Control Joint Torques $\Gamma_i(t)$

From the figure 7.6, it is possible to visualize that the joints are able to follow the desired joint trajectory q_c with a tracking error below the limits of the 0.05, for the Kp and Kd chosen.

Regarding the maximum torque, we see that the joints are all below the limits of the maximum torque from the motor of each joint. To verify that, it is possible to calculate the torque of the motor after the reduction and compare with the values obtained to the torque command by the control parameters. The torque limits of the motor after the reduction is given by : $\Gamma_{max} = [500, 500, 500, 350, 350, 350]^T$ and from the figure 7.7, we verify that for all the joints the requested torque is below the limits.

A Appendix - MatLab functions

A.1 Main Function

```
clear all
close all
clc
addpath('functions')
%%%%%%%%%%%%%%%%%%%%%%%%%



%% Problem specifications
%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%
% Geometric Parameters
%%%%%%%%%%%%%%%%%%%%%%%%%

d3   = 0.7;
r1   = 0.5;
r4   = 0.2;
r_e  = 0.1;

%%%%%%%%%%%%%%%%%%%%%
% Analyse Parameters
%%%%%%%%%%%%%%%%%%%%%

% Joint configuration qi
qi = [-pi/2, 0, -pi/2, -pi/2, -pi/2, -pi/2]';

% Joint configuration qf
qf = [0, pi/4, 0, pi/2, pi/2, 0]';

% Joint velocity q_dot
q_dot = [0.5, 1, -0.5, 0.5, 1, -0.5]';

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Question 2 - Robot Reference Parameters
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

robot.qmin = [-pi, -pi/2, -pi,  -pi, -pi/2,   -pi]';
robot.qmax = [  0,   pi/2,   0, pi/2,  pi/2,  pi/2]';

robot.n_joints = 6;
robot.n_frames = 7;
robot.alpha = [0, pi/2, 0, pi/2, -pi/2, pi/2, 0]';
robot.d     = [0, 0, d3, 0, 0, 0, 0]';
robot.theta = [0, 0, 0 + pi/2, 0, 0, 0, 0]';
robot.r     = [r1, 0, 0, r4, 0, 0, r_e]';

% Mass of each arm of the robot
robot.m = [15.0, 10.0, 1.0, 7.0, 1.0, 0.5]';

% Robot's Joints Inertia Matrix
robot.I = zeros(3,3,robot.n_joints);
robot.I(:,:,1) = [0.80, 0.00, 0.05 ;
                  0.00, 0.80, 0.00 ;  % Inertial tensor of the body 1
                  0.05, 0.00, 0.10];  % in the R1 frame[kg*m^2]

robot.I(:,:,2) = [0.10, 0.00, 0.10 ;
                  0.00, 1.50, 0.00 ;  % Inertial tensor of the body 2
                  0.10, 0.00, 1.50];  % in the R2 frame[kg*m^2]

robot.I(:,:,3) = [0.05, 0.00, 0.00 ;
                  0.00, 0.01, 0.00 ;   % Inertial tensor of the body 3
                  0.00, 0.00, 0.05];   % in the R3 frame[kg*m^2]

robot.I(:,:,4) = [0.50, 0.00, 0.00 ;
                  0.00, 0.50, 0.00 ;  % Inertial tensor of the body 4
                  0.00, 0.00, 0.05];  % in the R4 frame[kg*m^2]

robot.I(:,:,5) = [0.01, 0.00, 0.00 ;
                  0.00, 0.01, 0.00 ;  % Inertial tensor of the body 5
                  0.00, 0.00, 0.01];  % in the R5 frame[kg*m^2]

robot.I(:,:,6) = [0.01, 0.00, 0.00 ;
                  0.00, 0.01, 0.00 ;  % Inertial tensor of the body 6
                  0.00, 0.00, 0.01];  % in the R5 frame[kg*m^2]


% Vectors of Gi given in frame Ri
```

```
robot.cg = [ 0.00, 0.35,  0.00, 0.00, 0.00, 0.00 ;
              0.00, 0.00, -0.10, 0.00, 0.00, 0.00 ; % Coordinates of Gi given
             -0.25, 0.00,  0.00, 0.00, 0.00, 0.00]; % in frame Ri [m]

% Moment of inertia of the actuator rotor [kg*m^2]
robot.Jm = (10^-5)*[1; 1; 1; 1; 1; 1];

% Reduction ratio
robot.red = [100; 100; 100; 70; 70; 70]; % [ ]

% Joint viscous frictions
robot.Fv = [10; 10; 10; 10; 10; 10]; % [N*m*(rad^-1)*s]

% Maximal motor torques
robot.tau_max = [5; 5; 5; 5; 5; 5];  % [N*m]

%Acceleration of gravity in the reference frame
robot.g = [0.00; 0.00; -9.81];  % [m*s^-2]
%%%%%%%%%%%%%%%%%%%%%%%%%


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Problem design
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Questions 3 and 4
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Analyse position
theta_i = robot.theta + [qi;0];
theta_f = robot.theta + [qf;0];

% Transformation Matrix
g_0Ei = ComputeDGM(robot.alpha, robot.d, theta_i, robot.r);
g_0Ef = ComputeDGM(robot.alpha, robot.d, theta_f, robot.r);

% Position Vector of End frame at qi and qf
Pi = g_0Ei(1:3,4);
Pf = g_0Ef(1:3,4);

% Rotation Matrix of End frame at qi and qf
Rot_i = g_0Ei(1:3,1:3);
Rot_f = g_0Ef(1:3,1:3);

% Unitary vector and angle of the transformation
[phi_i,n_i] = inv_Rot(Rot_i);
[phi_f,n_f] = inv_Rot(Rot_f);


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Question 6
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Jacobian of End frame

% Joint configuration qi
J0_Ei = ComputeJac(robot.alpha, robot.d, theta_i, robot.r, Pi, robot.n_joints);
% Joint configuration qf
J0_Ef = ComputeJac(robot.alpha, robot.d, theta_f, robot.r, Pf, robot.n_joints);

% Velocties of end frame in i and f
V0_0Ei = J0_Ei*q_dot;
V0_0Ef = J0_Ef*q_dot;


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Question 7
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Single value decomposition
[U_i, Sig_i, V_i] = svd(J0_Ei(1:3,:));
[U_f, Sig_f, V_f] = svd(J0_Ef(1:3,:));

% Velocity Manipulability
Vel_Man_i = prod(diag(Sig_i));
Vel_Man_f = prod(diag(Sig_f));


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Question 8
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Joint configuration qi
```

```matlab
Xd_i = [-0.1, -0.7, 0.3]';
q0_i = [-1.57, 0.00, -1.47, -1.47, -1.47, -1.47]';

% Joint configuration qf
Xd_f = [0.64, -0.1, 1.14]';
q0_f = [0, 0.80, 0.00, 1.00, 2.00, 0.00]';

% Max number of iterations
k_max = 100;
% Max error
eps_max = 0.001;

% IGM by Newton-Raphson Method
[f_in, err_in, q_star_in] = ComputeIGM(robot, Xd_i, q0_i, k_max, eps_max, 'NewtonRaphson');
[f_fn, err_fn, q_star_fn] = ComputeIGM(robot, Xd_f, q0_f, k_max, eps_max, 'NewtonRaphson');

% Gradient Method
[f_ig, err_ig, q_star_ig] = ComputeIGM(robot, Xd_i, q0_i, k_max, eps_max, 'Gradient');
[f_fg, err_fg, q_star_fg] = ComputeIGM(robot, Xd_f, q0_f, k_max, eps_max, 'Gradient');


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Question 9 and 10
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Linear velocity
V_d = 1;
% Time sample
Te = 0.001;

% IKM
[Xdk, thetadk, qdk] = ComputeIKM(robot, Xd_i, Xd_f, V_d, Te, qi, k_max, eps_max, 'NewtonRaphson');


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Question 11
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% IKMLimits for tradeoff = 0.5
% tradeoff = 0.5;
% [X_star, theta_star, q_star] = ComputeIKMLimits(robot, Xd_i, Xd_f, V_d, Te, qi, robot.qmin, robot.qmax,
     k_max, eps_max, tradeoff, 'NewtonRaphson');
% err_tradeoff_05 = norm(X_star - Xdk);

% IKMLimits for tradeoff = 1
tradeoff = 1;
[X_star, theta_star, q_star] = ComputeIKMLimits(robot, Xd_i, Xd_f, V_d, Te, qi, robot.qmin, robot.qmax,
     k_max, eps_max, tradeoff, 'NewtonRaphson');
err_tradeoff_10 = norm(X_star - Xdk);


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Question 12
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

Joint_test_JacGi= 6;

% Jacobian matrix of centre of mass of rigid body
[Jv, Jw, p_cg_00, g] = ComputeJacGi(robot.alpha, robot.d, [qi;0] + robot.theta, robot.r, robot.cg,
     Joint_test_JacGi);


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Question 13
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Inertia matrix
Ai = ComputeMatInert(robot, qi);
Af = ComputeMatInert(robot, qf);


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Question 14
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Min and Max values for the inertia matrix
[mu_2,mu_1] =  A_bounds(robot);


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Question 15
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Torque due to the gravity
```

```
Gi = ComputeGravTorque(robot, qi);
Gf = ComputeGravTorque(robot, qf);


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Question 16
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Max value of torque due to the gravity
gb = G_bounds(robot);


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Question 17
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Torque due to joint friction
Gamma_f =  ComputeFrictionTorque(robot,q_dot);


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Question 18
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Initial and final cofigurations
robot.qd_i = [-1; 0; -1; -1; -1; -1];    % [rad] Initial position
robot.qd_f = [ 0; 1;  0;  0;  0;  0];    % [rad] Final position

% Minimal time to follow trajectory
Dj = robot.qd_f - robot.qd_i;
k_aj = robot.tau_max.*robot.red/mu_2;
tf_min = max( sqrt( 10*abs(Dj)./(sqrt(3)*k_aj) ));


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Question 19
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Trajectory generation
qc = GenTraj(qdi, qdf,t, tf_min);


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Question 20
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Control parameters
Kp = [5000; 8000; 5500; 800; 5500; 2500];
Kd = [1000; 1000; 500; 100; 100; 500];


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Simulink
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

open_system('main_simulink');
sim('main_simulink.slx');



%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Plots
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%
figure(1)
PlotFrame(g_0Ei) % Q5
hold on
create_ellipsoide(Pi,Sig_i,U_i) % Q7
PlotRobot(robot,qi)
legend('X','Y','Z','Velocity Ellipsoid','axis ellipsoid \sigma_1','axis ellipsoid \sigma_2','axis
    ellipsoid \sigma_3')
title('Ellipsoid of Velocity Manipulabity q_i')


%%
figure(2)
PlotFrame(g_0Ef) % Q5
hold on
create_ellipsoide(Pf,Sig_f,U_f) % Q7
PlotRobot(robot,qf)
```

```
legend('X','Y','Z','Velocity Ellipsoid','axis ellipsoid \sigma_1','axis ellipsoid \sigma_2','axis ellipsoid \sigma_3')
title('Ellipsoid of Velocity Manipulabity q_f')


%%  Question 9
figure(3)
title('Performed Trajectory qdk')
AnimationTrajectory(robot,thetadk,qdk,Xd_i,Xd_f,Xdk,'q9.avi')


%% Question 10
figure(4)
suptitle('Time Evolution of q_i')
JointTemporalPlot(robot.qmin, robot.qmax, Te, qdk)

%% Question 11
figure(5)
suptitle('Time Evolution of q_i with Secondary Task')
JointTemporalPlot(robot.qmin, robot.qmax, Te, q_star)

figure(6)
title('Performed Trajectory with Secondary Task')
AnimationTrajectory(robot,theta_star,q_star,Xd_i,Xd_f,X_star,'q11.avi')
%% Q19
figure(7)
suptitle('Desired Joint Trajectory Point qc(t)')
simdata_plot(sim_qc, 'q_c_')
%saveas(gcf,'q19_qc.png')

%% Q20
figure(8)
suptitle('Control Joint Torques \Gamma_i(t)')
simdata_plot(sim_Gamma, '\Gamma_')
%saveas(gcf,'q20_Gamma.png')

figure(9)
suptitle('Evolution of Joint Trajectories q_i(t)')
simdata_plot(sim_q, 'q_')
%saveas(gcf,'q20_q.png')

figure(10)
suptitle('Evolution of Tracking Errors e(t)')
simdata_plot(sim_error, 'e_')
%saveas(gcf,'q20_error.png')

figure(11)
suptitle('Evolution of q_i(t) and q_c(t)')
simdata_plot(sim_qc, 'qc_')
hold on
simdata_plot(sim_q, 'q_')
%saveas(gcf,'q20_qc_q.png')
```

Code 1 – Main

A.2 Direct geometric model

A.2.1 TransformMatElem

```
function g = TransformMatElem(alpha, d, theta, r)
%TransformMatElem
% Calculates the transformation matrix of a given grame
% with respect to the reference frame

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
```

Code 2 – TransformMatElem

A.2.2 ComputeDGM

```
function [g_ON] = ComputeDGM(alpha, d, theta, r)
%ComputeDGM
% Calculates the succesive transformations from the
% Reference Frame R0 to another Reference Frame Rn
%
% Returns the transformation matrix g_ON, 4x4.

    n_frames = length(alpha);
    g_ON = eye(4);
    for i=n_frames:-1:1
        g = TransformMatElem(alpha(i), d(i), theta(i), r(i));
        g_ON = g*g_ON;
    end
end
```

Code 3 – ComputeDGM

A.3 Direct kinematic model

A.3.1 ComputeJac

```
function J = ComputeJac(alpha, d, theta, r, p_e, n_joints)
%ComputeJac
% Calculates the jacobian matrix of a given point
%
% Returns the jacobian matrix of size n_joints x n_joints.

    J = zeros(6,n_joints);
    g_ON = eye(4);
    % Khalil-Kleinfinger Convention
    Zi = [0; 0; 1];

    for i=1:n_joints
        g = TransformMatElem(alpha(i), d(i), theta(i), r(i));
        g_ON = g_ON*g;

        % Revolution Joint
        R_0i = g_ON(1:3,1:3);
        p_in = g_ON(1:3,4);
        p = p_e - p_in;

        J(:,i) = [cross(R_0i*Zi,p);
                  R_0i*Zi;];
    end

end
```

Code 4 – ComputeJac

A.4 Inverse geometric model

A.4.1 ComputeIGM

```
function [Xd_estimated, err, q_star] = ComtputeIGM(robot, Xd, q_0, k_max, eps_x, method)
%ComputeIGM
% Calculates the estimated position from a desired position
% and an initial joint configuration by an iterative method
%
% Returns the estimated position, the error and the estimated
% joint configuration

    % tic
    step_IGM = 2;
    it = 1;
    err = 5000;
    f = [0,0,0]';
    while (it < k_max && err > eps_x)
        theta_0 = [q_0;0] + robot.theta;
        g = ComputeDGM(robot.alpha, robot.d, theta_0, robot.r);
        Xd_estimated = g(1:3,4);
        % J_full = ComputeJac(robot.alpha, robot.d, robot.theta + q_0, robot.r, Xd);
        J_full = ComputeJac(robot.alpha, robot.d,  theta_0, robot.r, Xd, robot.n_joints);
        J = J_full(1:3,:);
```

```
        if strcmp(method, 'NewtonRaphson')
            J_star = pinv(J);
        end
        if strcmp(method, 'Gradient')
            J_star =  step_IGM*J';
        end
        q_k = q_0 + J_star*(Xd-Xd_estimated);
        err = norm(Xd-Xd_estimated);
        it = it + 1;

        q_0 = q_k;
    end


    q_star = q_0;
    % fprintf("Method %s: ",method); toc
end
```

Code 5 – ComputeIGM

A.4.2 ComputeIKM

```
function  [Xdk, thetadk, qdk] = ComputeIKM(robot, Xd_i, Xd_f, V, Te, q_i, k_max, eps_x, method)
%ComputeIKM
% Calculates the best points and joint configurations of the desired
% trajectory

    delta_X = Xd_f - Xd_i;
    dist = norm(delta_X);
    u_x = (Xd_f - Xd_i)/dist;

    temp = dist/V;
    it=  ceil(temp/Te);
    Xdk = zeros(length(Xd_i),it);
    qdk = zeros(length(q_i),it);

    % iter 1
    Xdk(:,1) = Xd_i;
    qdk(:,1) = q_i;

    for i=2:it-1
        Xdk(:,i) = Xdk(:,i-1) + (V*Te)*u_x;
        [~, ~, qdk(:,i)] = ComputeIGM(robot, Xdk(:,i), qdk(:,i-1), k_max, eps_x, method);

    end

    Xdk(:,it) = Xd_f;
    [~, ~, qdk(:,it)] = ComputeIGM(robot, Xdk(:,it), qdk(:,it-1), k_max, eps_x, method);

    thetadk = [qdk;zeros(1,it)] + robot.theta;

end
```

Code 6 – ComputeIKM

A.4.3 ComputeIKMlimits

```
function [X_star,theta_star, q_star] = ComputeIKMLimits(robot, Xd_i, Xd_f, V, Te, q_i, q_min, q_max,
     k_max, eps_x, tradeoff, method)
%ComputeIKMLimits
% Calculates the best points and joint configurations of the desired
% trajectory while respecting the angle restrictions qmax and qmin
    n = length(q_i);
    delta_X = Xd_f - Xd_i;
    dist = norm(delta_X);
    u_x = (Xd_f - Xd_i)/dist;

    temp = dist/V;
    it=  ceil(temp/Te);
    t_lim = 0:Te:temp;
    Xdk_dot = V*u_x;
    q_star= zeros(length(q_i),it);
    q_star_dot = zeros(length(q_i),it);
    X_star = zeros(3,it);
    % iter 1

```

```
    [Xdk, thetadk, qdk] = ComputeIKM(robot, Xd_i, Xd_f, V, Te, q_i, k_max, eps_x, method);
    qi_dash =  (q_max + q_min)/2;

    for k=1:it
        dH_lim = 2*(qdk(:,k)-qi_dash)./((q_max - q_min).^2);
        q0_dot = -tradeoff*dH_lim;

        J = ComputeJac(robot.alpha, robot.d, thetadk(:,k), robot.r, Xdk(:,k), robot.n_joints);
        rank_J(k) = rank(J);
        Jv =  J(1:3,:);
        J_sharp = pinv(Jv);
        N_j = eye(length(Jv(1,:))) - (J_sharp*Jv);
        q_star_dot(:,k) = J_sharp*Xdk_dot + N_j*q0_dot;
    end

    for i=1:n
        q_star(i,:) = cumtrapz(t_lim,q_star_dot(i,:));
    end

    q_star(:,:) = q_star(:,:) + q_i;

    theta_star = [q_star;zeros(1,it)] + robot.theta;

    for k=1:it
        g = ComputeDGM(robot.alpha, robot.d, theta_star(:,k), robot.r);
        X_star(:,k) = g(1:3,4);
    end

end
```

Code 7 – ComputeIKMlimits

A.5 Inverse dynamic model

A.5.1 ComputeJacGi

```
function [Jv, Jw, p_cg_00, g] = ComputeJacGi(alpha, d, theta, r, cg, ji)
%ComputeJacGi
% Calculates the jacobian matrix of the centre of mass
% of a rigid body
% Returns the jacobian matrix of size ji x ji.

    alpha_cg = alpha(1:ji);
    d_cg = d(1:ji);
    theta_cg = theta(1:ji);
    r_cg = r(1:ji);
    g = ComputeDGM(alpha_cg,d_cg,theta_cg,r_cg);
    p_cg_00 = g(1:3,:)*[cg(:,ji);1];

    J = zeros(6);
    J(:,1:ji) = ComputeJac(alpha_cg,d_cg,theta_cg,r_cg, p_cg_00,ji);

    Jv = J(1:3,:);
    Jw = J(4:6,:);

end
```

Code 8 – ComputeJacGi

A.5.2 ComputeMatInert

```
function A = ComputeMatInert(robot, q)
%ComputeMatInert
% Calculates the inertia matrix of a given configuration
%
% Returns the inertia matrix of size n_joints x n_joints.

    theta_0 =  robot.theta + [q;0];
%     theta_0 =  robot.theta;
%     theta_0(1:6) = theta_0(1:6) + q;
    A = zeros(6);

    for i=1:length(q)

        [Jvi, Jwi, p_cg_00, g] = ComputeJacGi(robot.alpha, robot.d, theta_0, robot.r, robot.cg, i);
```

```
        IG_i = robot.I(:,:,i) - robot.m(i)*skew(robot.cg(:,i));
        R0_i = g(1:3,1:3);
        I0_i = R0_i*IG_i*R0_i';
        A = A + (robot.m(i)*(Jvi'*Jvi) + Jwi'*I0_i*Jwi);
    end
    A = A + diag(robot.red.^2.*robot.Jm);


function M = skew(x)

    X=[    0, -x(3),  x(2) ;
        x(3),     0, -x(1) ;
       -x(2),  x(1),    0 ];
    M = -X*X;
```

Code 9 – ComputeMatInert

A.5.3 ComputeGravTorque

```
function G = ComputeGravTorque(robot,q)
%ComputeGravTorque
% Computes the resistives torques in the joints due to
% gravitational effects.
% Returns a vector 6x1, with the corresponding resisitve torque

    theta_0 =  robot.theta + [q;0];
    G = zeros(length(q),1);

    for i=1:length(q)
        [Jv, ~]  = ComputeJacGi(robot.alpha, robot.d, theta_0, robot.r, robot.cg, i);
        G = G - robot.m(i)*(Jv'*robot.g);
    end

end
```

Code 10 – ComputeGravTorque

A.5.4 ComputeFrictionTorque

```
function Gamma_f =  ComputeFrictionTorque(robot,q_dot)
%ComputeFrictionTorque
% Computes the Friction Torque from the viscous frictions
% and the current speed
% Returns a vectir Gamma_f 6x1.

    Gamma_f = diag(q_dot)*robot.Fv;
end
```

Code 11 – ComputeFrictionTorque

A.5.5 GenTraj

```
function qc  = GenTraj(qdi, qdf,t, tf_min)
%GENTRAJ
%   Generates a 5th degree polynomial trajectory from
%   the initial and final values qdi and qdf.

D = qdf - qdi;
t_star = t/tf_min;
r = 6*t_star^5 -15*t_star^4 +10*t_star^3;
qc = qdi + r*D;
end
```

Code 12 – GenTraj