Rapport sur le Contrôle et la Simulation du Robot UR10

Auteur : Curtis Martelet

Date : 29/11/2024

Table des matières

1. Introduction

Ce rapport sert de conclusion au travail réalisé sur la simulation et le contrôle du robot UR10. Nous y développons le calcul et l'implémentation d'un système de commande avec la méthode de Denavit-Hartenberg modifiée.

Nous décrivons également les étapes de simulation sous **CoppeliaSim** ainsi que les résultats obtenus.

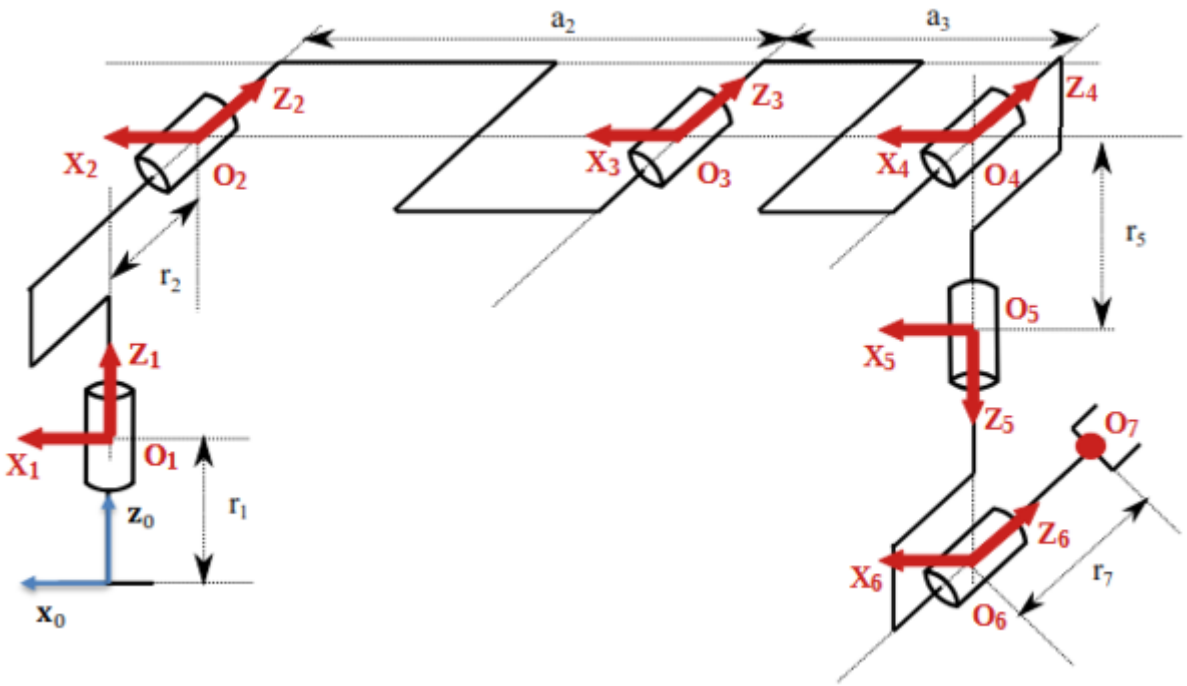
2. Modélisation du Robot UR10

2.1. Présentation de la méthode DH

La méthode Denavit-Hartenberg (DH) permet de décrire la géométrie d'un robot manipulateur. Chaque articulation est modélisée par une matrice de transformation homogène qui exprime la position et l'orientation d'un repère local par rapport au repère précédent.

Dans le cadre de ce TP, nous avons utilisé la méthode Denavit-Hartenberg Modifié. Cette méthode place le repère d'origine du robot sur la première articulation, simplifiant la définition des paramètres.

2.2. Schéma du robot



2.3. Paramètres DH

Simulation

Dans le cas de la simulation dans **CoppeliaSim**, les paramètres DH sont modifiés pour aligner les axes des repères du simulateur.

Joint	σ_j	α_{j-1} (rad)	a_{j-1} (m)	θ_j (rad)	d_j (m)
1	0	0	0	θ_1	r_1
2	0	$\pi/2$	0	$\theta_2 - \pi/2$	r_2
3	0	0	$-a_2$	θ_3	0
4	0	0	$-a_3$	θ_4	0
5	0	$\pi/2$	0	$\theta_5 - \pi/2$	r_5
6	0	$-\pi/2$	0	θ_6	0

Robot Réel

Pour le robot réel, les repères respectent directement la configuration physique du robot, ce qui entraîne une différence au niveau des angles des joints :

Joint	σ_j	α_{j-1} (rad)	a_{j-1} (m)	θ_j (rad)	d_j (m)
1	0	0	0	θ_1	r_1
2	0	$\pi/2$	0	θ_2	r_2
3	0	0	$-a_2$	θ_3	0
4	0	0	$-a_3$	θ_4	0
5	0	$\pi/2$	0	θ_5	r_5
6	0	$-\pi/2$	0	θ_6	0

Nous avons retiré les décalages de $\pi/2$ sur les articulations 2 et 5.

3. Formules et modélisation mathématique

Dans cette seconde partie du rapport, nous allons détailler les étapes et calculs utilisés dans la commande du robot.
On retrouve dans la loi de commande :

- Un bloc **Générateur de Trajectoire**, qui nous donne la vitesse et position désirée au cours du temps.
- Un bloc **Modèle Géométrique**, qui nous donne la pose de l'organe terminal en fonction des angles des articulations.
- Un bloc **Erreur** qui calcul l'erreur entre la position et l'orientation désirée, et celle réelle.
- Un bloc **Contrôleur**, qui calcul la vitesse angulaire désirée de chaque articulation.

Dans le cas de la simulation sur **CoppeliaSim**, il nous faudra également intégrer la vitesse angulaire désirée pour obtenir les angles de chaque articulation : les articulations de la simulation se commandent en position tandis que celles du robot réel se commandent en vitesse.

3.1. Générateur de Trajectoire

La trajectoire est calculée pour définir la position et l'orientation désirées de l'effecteur au cours du temps. Nous avons utilisé un polynôme de degré cinq pour garantir des trajectoires continues en position, vitesse et accélération.

Calcul des positions et vitesses désirées

Les positions et vitesses désirées sont calculées à partir des paramètres de la trajectoire et de la durée de simulation.

On utilise une équation de degré cinq pour calculer r et sa dérivée $\dot{r}$. Ces valeurs sont ensuite utilisées pour calculer la position et la vitesse.

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

$$r_{\text{point}}(t) = 30\frac{t^2}{t_f^3} - 60\frac{t^3}{t_f^4} + 30\frac{t^4}{t_f^5}$$

Une fois, on peut calculer la position et la vitesse désirée au cours du temps :

$$x_{\text{désirée}} = x_{\text{init}} + r(t)(x_{\text{final}} - x_{\text{init}})$$

$$x_{\text{point désirée}} = r_{\text{point}}(t)(x_{\text{final}} - x_{\text{init}})$$

Calcul de l'orientation désirée

La génération de l'orientation désirée pour l'effecteur se base sur l'interpolation entre la rotation actuelle et la rotation finale. Le calcul suit les étapes suivantes :

Calcul de la matrice de rotation relative

La matrice de rotation relative RR est calculée avec cette formule :

$$R = R_{\text{init}}^{T} \cdot R_{\text{final}}$$

Elle représente la transformation nécessaire pour passer de l'orientation initiale à l'orientation finale.

Calcul de l'angle de rotation θ

L'angle de rotation θ est extrait de la matrice de rotation relative R.

Il s'obtient en calculant le cosinus et le sinus, puis en utilisant tangente :

$$\cos(\theta) = \frac{\text{Tr}(R) - 1}{2}$$

$$\sin(\theta) = \frac{\sqrt{(R_{32} - R_{23})^2 + (R_{13} - R_{31})^2 + (R_{21} - R_{12})^2}}{2}$$

$$\theta = \tan^{-1}\left(\frac{\sin(\theta)}{\cos(\theta)}\right)$$

- $Tr(R)$ représente la trace de la matrice R (somme des éléments de la diagonale de la matrice).
- Les éléments $R_{i,j}$ sont les composantes individuelles de la matrice de rotation R.

Calcul de l'axe de rotation $\boldsymbol{u}$

Une fois θ calculé, on peut déterminer l'axe de rotation u.

Il est donné par :

$$\mathbf{u} = \frac{1}{2\sin(\theta)} \begin{bmatrix} R_{32} - R_{23} \\ R_{13} - R_{31} \\ R_{21} - R_{12} \end{bmatrix}$$

Avec la matrice R obtenue avec cette équation :

$$R = R_{\text{init}}^{T} \cdot R_{\text{final}}$$

Avec :

- R_{final} la rotation finale désirée de l'organe terminal.
- ${R_{Init}}^{T}$ la transpose de la matrice de rotation de la pose initiale.

Interpolation de la rotation désirée

La rotation désirée $R_{désirée}$ est obtenue en appliquant une interpolation en fonction du facteur rr, calculé à partir de la trajectoire temporelle :

$$\mathbf{R}_{\text{désirée}} = \mathbf{R}_{\text{init}} \cdot \mathbf{rot}(\mathbf{u}, r \cdot \theta)$$

La matrice $rot(u, r, \theta)$ est une matrice de rotation autour de l'axe u d'un angle $r \cdot \theta$, donnée par :

$$\mathbf{rot}(\mathbf{u}, r \cdot \theta) = \begin{bmatrix} u_x^2(1-\cos(r\cdot\theta)) + \cos(r\cdot\theta) & u_x u_y(1-\cos(r\cdot\theta)) - u_z \sin(r\cdot\theta) & u_x u_z(1-\cos(r\cdot\theta)) + u_y \sin(r\cdot\theta) \\ u_x u_y(1-\cos(r\cdot\theta)) + u_z \sin(r\cdot\theta) & u_y^2(1-\cos(r\cdot\theta)) + \cos(r\cdot\theta) & u_y u_z(1-\cos(r\cdot\theta)) - u_x \sin(r\cdot\theta) \\ u_x u_z(1-\cos(r\cdot\theta)) - u_y \sin(r\cdot\theta) & u_y u_z(1-\cos(r\cdot\theta)) + u_x \sin(r\cdot\theta) & u_z^2(1-\cos(r\cdot\theta)) + \cos(r\cdot\theta) \end{bmatrix}$$

Dans cette formule :

- u_x, u_y, u_z sont les composantes de l'axe de rotation $\boldsymbol{u}$.
- $r \cdot \theta$ est l'angle interpolé pour le temps actuel.

La rotation désirée est ensuite multipliée avec l'orientation actuelle pour obtenir $R_{désiré}$. Cette dernière matrice sera utilisée dans le calcul de de l'erreur.

3.2. Modèle Géométrique

Le modèle géométrique permet de déterminer des positions cartésiennes à partir des angles de chaque articulation.

Les coordonnées cartésiennes réelle du robot seront utilisées pour calculer l'erreur de position et orientation entre la pose désirée et actuelle.

Les Matrices de Transformation

La formule générale d'une matrice de transformation homogène selon DH modifié est la suivante :

$$T_j = \begin{bmatrix} \cos\theta_j & -\sin\theta_j \cos\alpha_{j-1} & \sin\theta_j \sin\alpha_{j-1} & a_{j-1} \\ \sin\theta_j & \cos\theta_j \cos\alpha_{j-1} & -\cos\theta_j \sin\alpha_{j-1} & -r_j \sin\alpha_{j-1} \\ 0 & \sin\alpha_{j-1} & \cos\alpha_{j-1} & r_j \cos\alpha_{j-1} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Une fois les matrices de transformations de chaque articulation déterminée, nous devons calculer la matrice de transformation entre l'origine et la dernière articulation du robot.

$$T_{06} = T_1 \cdot T_2 \cdot T_3 \cdot T_4 \cdot T_5 \cdot T_6$$

La position de l'effecteur final peut être déterminée en multipliant cette matrice par le vecteur homogène représentant l'effecteur :

$$\mathbf{O_{07}} = T_{06} \cdot \mathbf{O_{67}}$$

Où :

- T_{06} est la matrice de transformation de la base au repère final.

- O_{67} est le vecteur décrivant la position de l'effecteur par rapport à l'articulation 6.

Avec cette matrice O_{07} , nous avons la position de l'effecteur. Cette donnée est utilisée pour trouver l'erreur de position et d'orientation entre la position désirée et la position actuelle de l'effecteur.

Calcul de la Jacobienne

Calcul de la Jacobienne J_{06}.

La Jacobienne fait une taille $6 \times n$ avec n le nombre d'articulations. Chaque colonne de notre jacobienne correspond à une articulation du robot.
La méthode pour déterminer ces colonnes dépend de la nature des articulations (rotatives ou prismatiques).

Articulation Rotoïde	Articulation Prismatique
$J_p = Z_i \times (P_n - P_i)$	$J_p = Z_i$
$J_o = Z_i$	$J_o = 0_{3\times3}$

avec :
- Z_i: Vecteur de l'axe de rotation z pour l'articulation i. Il s'agit de la dernière colonne de la matrice de rotation de l'articulation i.
- P_n: Vecteur position de la dernière articulation.
- P_i: Vecteur position de l'articulation i.

Dans le cas du robot UR10, nous n'avons pas d'articulation prismatique.

Calcul de la matrice D

Dans le cas du robot UR10, il existe un décalage r_7 entre la dernière articulation et le centre de l'effecteur. Ce décalage est pris en compte à l'aide de la matrice D, qui est calculée à partir des éléments de la matrice de rotation R_{06} et de la matrice O_{67}.

$$\mathbf{D} = \begin{pmatrix} 0 & a_n x_z + r_{n+1} z_z & -a_n x_y - r_{n+1} z_y \\ -a_n x_z - r_{n+1} z_z & 0 & a_n x_x + r_{n+1} z_x \\ a_n x_y + r_{n+1} z_y & -a_n x_x - r_{n+1} z_x & 0 \end{pmatrix}$$

avec :
- $a_n$et $r_n$trouvable dans O_{67}.
- - $x_x,\ x_y,\ x_z,\ z_x,\ z_y,\ z_z$ trouvable en suivant la convention suivante :

$${}^0R_n = \begin{pmatrix} x_x & y_x & z_x \\ x_y & y_y & z_y \\ x_z & y_z & z_z \end{pmatrix}$$

Calcul de le Jacobienne Finale

Une fois J_P, J_{06}, et D calculés, on peut finalement calculer la Jacobienne finale J.

$$J = \begin{bmatrix} I_3 & D \\ 0_{3\times3} & C \end{bmatrix} \cdot J_{06}$$

avec :

- I_3 une matrice identité de taille 3×3.
- $O_{3\times3}$ une matrice de zéros de taille 3×3.
- C une matrice que nous avons choisi identité.
- D la matrice que l'on a obtenue plus tôt.
- J_{06} la jacobienne calculé plus tôt.

3.3. L'Erreur

Le calcul de l'Erreur permet de calculer la différence entre la position et l'orientation actuelle du robot et celle désirée à un moment t.
Les deux types d'erreurs sont ainsi calculées :

1. Erreur de position :

Cette erreur représente la différence entre la position actuelle de l'effecteur (x_{actuel}) et la position désirée ($x_{désiré}$) :

$$e_p = x_{\text{désiré}} - x_{\text{actuel}}$$

2. Erreur d'orientation :

Cette erreur exprime la différence entre l'orientation actuelle de l'effecteur (R_{actuel}) et l'orientation désirée ($R_{désirée}$). L'erreur est calculée comme une combinaison des vecteurs de base des deux orientations :

$$e_o = \frac{1}{2}\left(R_{\text{actuel}}[:,0] \times R_{\text{désiré}}[:,0] + R_{\text{actuel}}[:,1] \times R_{\text{désiré}}[:,1] + R_{\text{actuel}}[:,2] \times R_{\text{désiré}}[:,2]\right)$$

Matrice L :

Pour faciliter le calcul de l'erreur d'orientation, on va calculer une matrice auxiliaire L. Elle relie les rotations souhaitées et actuelles en modifiant leur structure :

$$L = -\frac{1}{2}\left(S(R_{\text{désiré}}[:,0])S(R_{\text{actuel}}[:,0]) + S(R_{\text{désiré}}[:,1])S(R_{\text{actuel}}[:,1]) + S(R_{\text{désiré}}[:,2])S(R_{\text{actuel}}[:,2])\right)$$

Ici, $S(v)$ est la matrice antisymétrique d'un vecteur v.

Pour un vecteur donné $v = \begin{bmatrix} v_x \\ v_y \\ v_z \end{bmatrix}$, la matrice antisymétrique $S(v)$ est définie comme :

$$S(v) = \begin{bmatrix} 0 & -v_z & v_y \\ v_z & 0 & -v_x \\ -v_y & v_x & 0 \end{bmatrix}$$

Les deux erreurs e_p et e_o, ainsi que la matrice L seront par la suite utilisées dans le contrôleur afin de corriger les écarts de trajectoire et d'orientation.

3.4. Le Contrôleur

Le contrôleur de notre chaine de commande sert à calculer les vitesses articulaires ($\dot{q}$) nécessaires pour que le robot suive la trajectoire désirée.

Utilisation de la Jacobienne pseudo-inverse

La commande est ensuite obtenue en effectuant la pseudo-inverse de la Jacobienne.
Nous utilisons la formule ci-dessous pour éviter les singularités :

$$\mathbf{J}^* = \mathbf{J}^T \cdot (\mathbf{J} \cdot \mathbf{J}^T + k^2 \cdot \mathbf{I})^{-1}$$

avec :

- k est une constante de régularisation pour éviter les singularités. Nous l'avons défini à 0.01.
- I la matrice identité.
- J la jacobienne obtenue avec le modèle géométrique.

Composition de la commande

La commande finale combine deux parties :

1. **Commande de position**.
2. **Commande d'orientation**.

$$\mathbf{q}_p^{\text{desire}} = \mathbf{J}^* \cdot \begin{bmatrix} \mathbf{x}_p^{\text{desire}} + K_p \cdot \mathbf{e}_{\text{position}} \\ L^{-1} \cdot \left(L^T \cdot \mathbf{w}_d + K_o \cdot \mathbf{e}_{\text{orientation}}\right) \end{bmatrix}$$

Avec :

- L la matrice auxiliaire calculée en même temps que l'erreur de position et d'orientation.
- $x_p^{désiré}$ la vitesse désirée.
- $e_{orientation}$ et $e_{position}$ les erreurs en position et d'orientation.
- K_o et K_P des matrices identités multiplié par les coefficients des erreurs d'orientation et de position. Nous les avons choisis égaux à 1.5.
- La vitesse angulaire désirée w_d est égal à :

$$\mathbf{w}_d = R_{\text{init}} \cdot (r_{\text{point}} \cdot \theta \cdot \mathbf{u})$$

3.5. Commande des Moteurs

A la différence du robot réel, les articulations de simulation de l'UR10 sur **CoppeliaSim** sont commandées en position et non en vitesse.

Une fois les vitesses articulaires désirées (q_p^{desire}) calculées par le contrôleur, il faut donc les convertir en position articulaire.

Mise à jour des positions articulaires

Les nouvelles positions des articulations sont calculées en intégrant les vitesses articulaires désirées sur un pas de temps donné. Cela revient à utiliser une méthode d'intégration simple pour mettre à jour les positions en fonction de la vitesse et du temps écoulé (Δt) depuis la dernière mise à jour :

$$\mathbf{q}_{\mathbf{new}} = \mathbf{q}_p^{\mathbf{desire}} \cdot \Delta t + \mathbf{q}_{\mathbf{current}}$$

Où :

- q_{new} est le vecteur des nouvelles positions articulaires.
- q_p^{desire} est le vecteur des vitesses articulaires désirées.
- Δt est l'intervalle de temps écoulé depuis la dernière mise à jour.
- $q_{current}$ est le vecteur des positions actuelles des articulations.

4. Implémentation

4.1. Langages et bibliothèques utilisés

Pour programmer la simulation et le contrôle du robot UR10, nous avons utilisé les outils suivants :

- **Python** pour les calculs et la commande.
- **Numpy** pour les manipulations matricielles.
- **Matplotlib** pour afficher les courbes.
- **CoppeliaSim** pour la simulation du robot.

4.2. Étapes de la simulation

Connexion à CoppeliaSim

Une connexion avec **CoppeliaSim** est établie pour contrôler le robot (simulé). Les **handles** des articulations sont récupérés afin de pouvoir les piloter.

Nous avons utilisé le mode `simx_opmode_blocking` pour récupérer l'angle des moteurs.

Génération de la trajectoire

La trajectoire est générée en interpolant entre la position initiale et la position finale en utilisant des polynômes de degré 5.

Calcul des commandes des articulations

Pour chaque instant tt, les étapes suivantes sont effectuées :

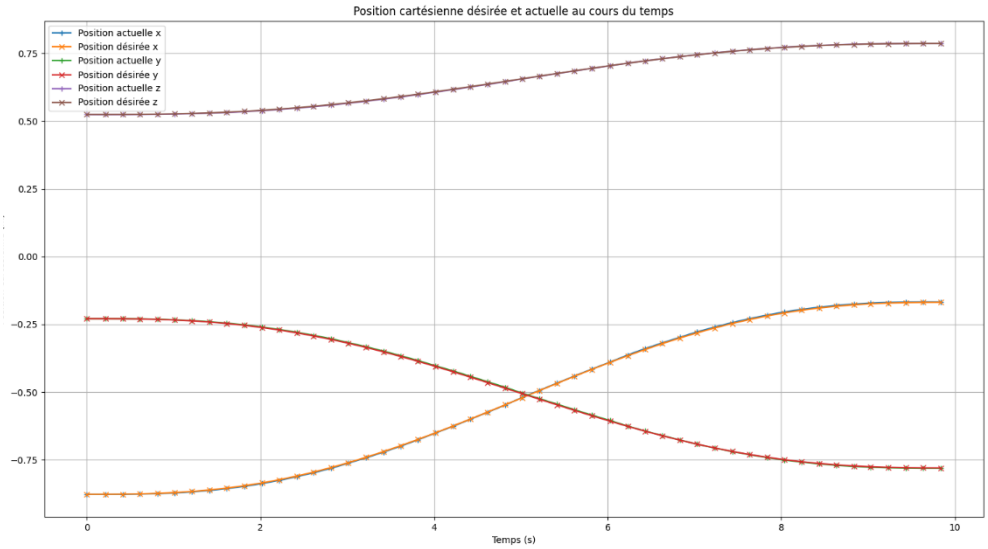
- Calcul de la **géométrique directe** pour déterminer la position actuelle de l'effecteur.
- Comparaison avec la position désirée pour obtenir **l'erreur**.
- Utilisation d'une **loi de commande** pour déterminer les vitesses articulaires désirées.
- **Intégration** pour obtenir les nouvelles positions des joints.

Envoi des commandes à CoppeliaSim

Les positions calculées sont envoyées aux articulations du robot dans **CoppeliaSim** via les commandes API. Nous avons utilisé le mode `simx_opmode_blocking` pour commander les moteurs.

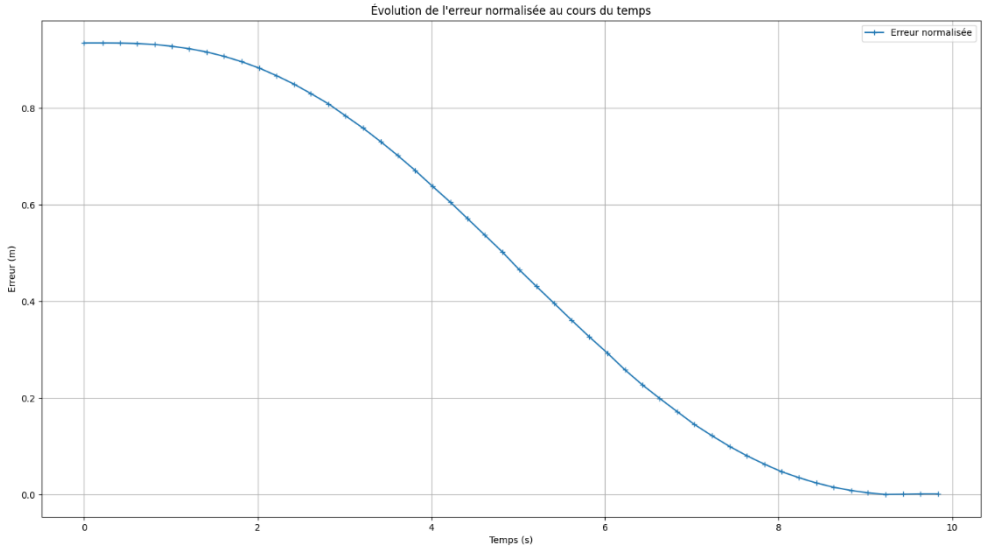
5. Courbes

Position désirées et actuelles des joints au cours du temps



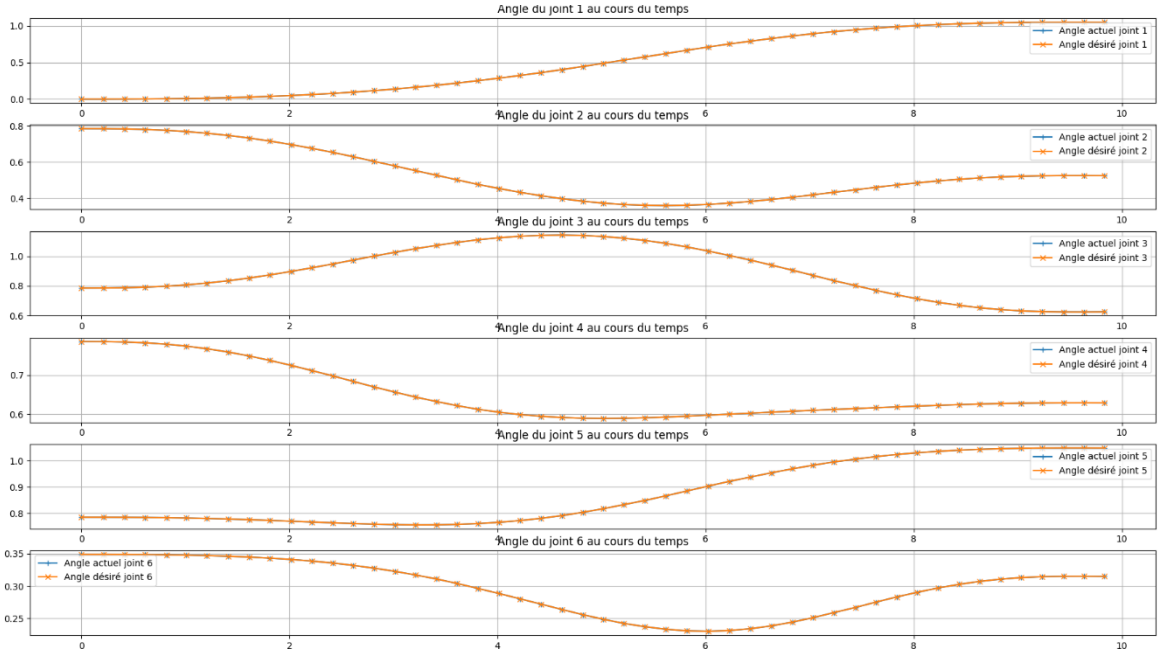
Sur ce graphique, on peut observer que le robot suit parfaitement la trajectoire demandée.

Erreur Normalisé



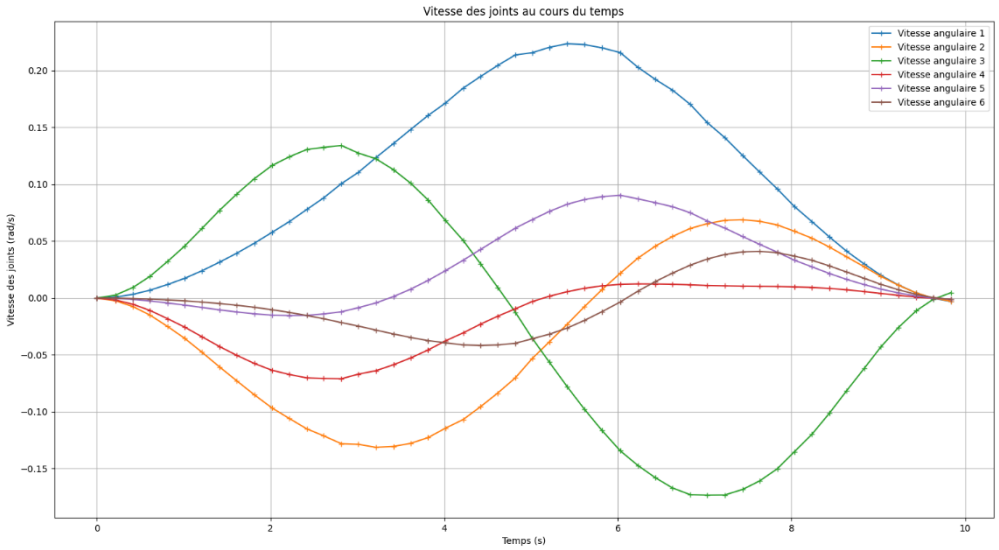
On peut voir sur ce graphique que l'erreur normalisé calculée entre la position initiale de l'organe terminal et la position finale tend vers zéro : le robot se déplace et atteint la position finale qu'on lui a demandé.

Angles désirées et actuelles des joints au cours du temps



Comme pour le premier graphique, les angles des articulations actuels suivent celle désirées.

Vitesses actuelles des articulations au cours du temps



A t_{final}, les vitesses des joints sont nulles.

Conclusion

En conclusion, la simulation du robot UR10 sur **CoppeliaSim** fonctionne. Le robot parvient à suivre la trajectoire définie, même lorsque celle-ci passe par une singularité.

Dans ce projet, j'ai été amené à travailler avec plusieurs outils mathématiques et logiciels nouveaux :

1. La Méthode de Denavit-Hartenberg Modifié m'a permis de plus facilement modéliser mathématiquement un robot 6 degrés de liberté sans me perdre dans les calculs.
2. CoppeliaSim est un puissant outil de simulation et de commande qui m'a permis de tester mon programme sans risquer d'endommager le robot.

Ce TP m'a également servi à comprendre pourquoi l'on avait des singularités de représentation, et comment les éviter, notamment avec l'utilisation de la jacobienne J^* au lieu de sa pseudo-inverse.

J'aurais cependant aimé tester mon programme sur le robot réel, mais un problème dans mon programme (que j'ai pu corriger) m'a causé trop de retard.