

UNIVERSITE CLAUDE BERNARD LYON 1

DEPARTEMENT DU MECANIQUE

(Electronique, Energie Electrique, Automatique)

Projet

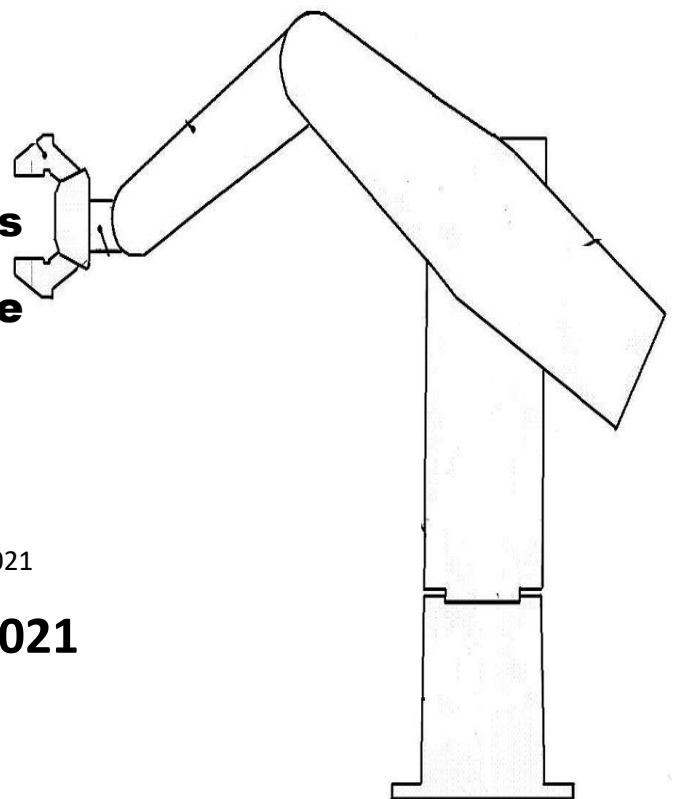
Robotique Appliquée

KUKA KR-I 000 TITAN

Rédigé par :

BELBECIR Riyadh Abbes

M2 GSA Recherche



28/02/2021

2020-2021

Introduction et présentation du Projet :

La robotique dans notre ère est devenue l'une des disciplines technologiques les plus répondus dans le domaine industriel car elle offre la vitesse, la force, la précision et la fiabilité nécessaires pour réaliser les tâches de production les plus ennuyeuses (répétitives), sales, dangereuses et difficiles (4D Dull, Dirty, Dangerous And Difficult) alors que l'être humain ne peut pas les faire.

C'est pour cela dans ce projet on va s'intéresser à l'étude des bras robotiques et précisément celui fabriqué par KUKA modèle KR 1000 titan.



L'objectif final est de planifier la trajectoire de l'effecteur d'un point A vers un point B et de la comparer avec la droite (AB) tout en variant les angles θ_i de nos liens.

La validation du modèle et les résultats sera faite par un script programmé en MATLAB.

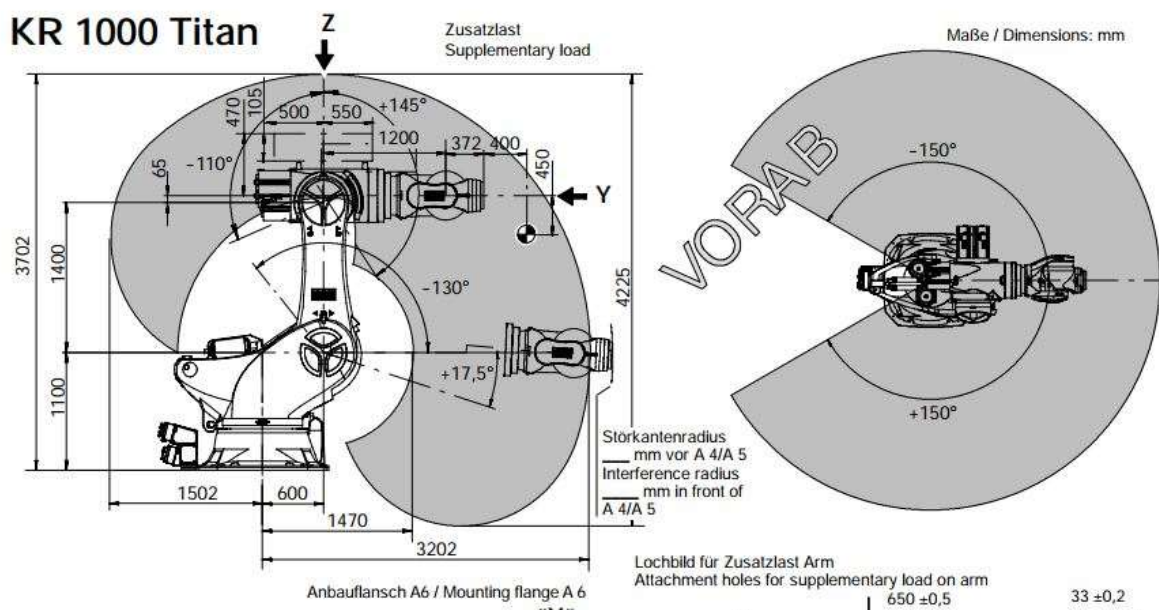
Pour atteindre cet objectif il faut passer par des étapes intermédiaires :

- Établir un schéma cinématique du robot (topologie et paramètres géométriques) ;
- Déterminer les paramètres de Denavit-Hartenberg
- Établir le modèle géométrique direct ;
- Établir une loi de commande pour une trajectoire de type point à point (Cette partie n'est pas développée pour ce Projet) ;
- Déterminer la trajectoire de l'effecteur.

La présentation du robot :

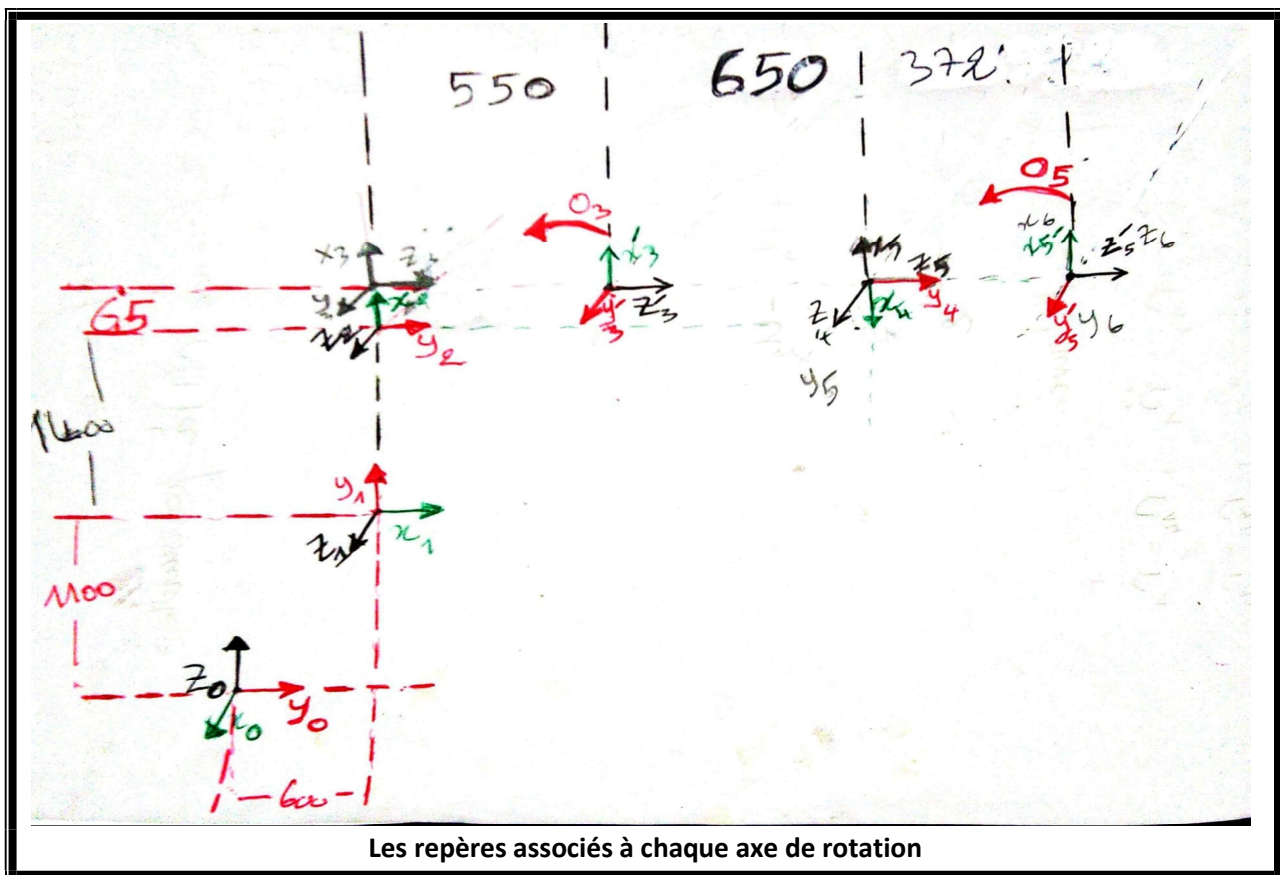
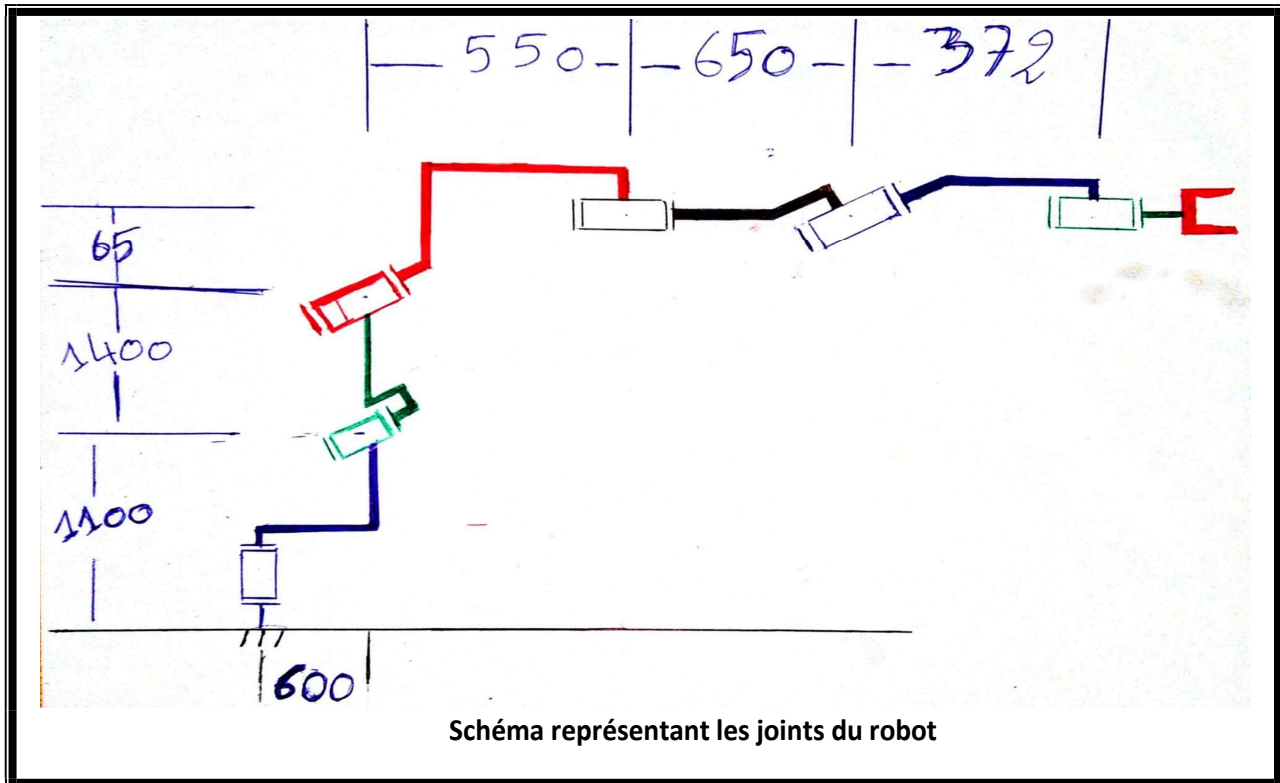
Le KUKA KR1000 Titan est un manipulateur anthropomorphe qui possède six joints rotoides donc six axes de rotation avec 6 ddl et 1 repère associé à chaque joint.

Ce bras robotique pèse 4690 Kg, il occupe un volume fonctionnel de 79.8 m³ avec une charge maximale de 1000Kg. Le reste des informations techniques nécessaires pour ce projet est sur le schéma suivant obtenu du DATASHEET du robot.



Achsdaten / Axis data	Bereich (Software) / Range (software)	Geschwindigkeit / Speed
Achse / Axis 1 (A1)	±150°	58 °/s
Achse / Axis 2 (A2)	+17,5°/-130°	50 °/s
Achse / Axis 3 (A3)	+145°/-110°	50 °/s
Achse / Axis 4 (A4)	±350°	60 °/s
Achse / Axis 5 (A5)	±118°	60 °/s
Achse / Axis 6 (A6)	±350°	84 °/s

Le schéma cinématique du robot :



La définition des repères par la convention DH :

Les conditions DH :

$$DH1 : X_i \perp Z_{i-1}$$

$$DH2 : X_i \cap Z_{i-1} \neq 0$$

- On considère les axes Z_i comme axes de rotation.
- S'il n'y a pas une intersection entre Z_i et Z_{i-1} et ils ne sont pas parallèles alors :
 - o X_i est au long la ligne normale commune (unique)
 - o L'origine O_i est l'intersection entre Z_i et la normale commune.
 - o $(Z_0 \rightarrow Z_1) \& (Z_2 \rightarrow Z_3)$ on a fait une translation des origines vers les intersections comme le montre la figure ci-dessus (Pas forcément que les repères soient sur les joints).
- Si Z_i et Z_{i-1} sont parallèles mais sans intersection alors :
 - o O_i peut être placé n'importe où sur Z_i .
 - o $(Z_1 \rightarrow Z_2)$
- Si Z_i et Z_{i-1} intersectent alors :
 - o $X_i = \pm Z_i \times Z_{i-1}$
 - o O_i est l'intersection entre Z_i et Z_{i-1} .
 - o $(Z_3 \rightarrow Z_4) \& (Z_4 \rightarrow Z_5)$.
- Le repère Z_6 représente le corps final et bouge identiquement que Z_5 .

Pour aller du repère Z_{i-1} vers Z_i il faut faire :

- 1- Translation le long du Z_{i-1} par d_i .
- 2- Rotation autour Z_{i-1} par Θ_i .
- 3- Translation le long du X_i par a_i .
- 4- Rotation autour X_i par α_i .

La détermination des paramètres de Denavit-Hartenberg :

$d_i, \Theta_i, a_i, \alpha_i$: Les paramètres DH :

- $a_i = \text{dist}(Z_{i-1}, Z_i)$ le long de X_i .
- $\alpha_i = \text{Angle}(Z_{i-1}, Z_i)$ autour X_i .
- $d_i = \text{dist}(X_{i-1}, X_i)$ le long de Z_{i-1} .
- $\Theta_i = \text{Angle}(X_{i-1}, X_i)$ le long de Z_{i-1} .

i	a _i	α _i	D _i	Θ _i	Θ _i initial
1	+600	+90	+1100	Θ ₁ +Θ ₁ -in	+90
2	1400	0	0	Θ ₂ +Θ ₂ -in	+90
3	0	+90	-65	Θ ₃ +Θ ₃ -in	0
4	0	+90	+650+550	Θ ₄ +Θ ₄ -in	180
5	0	+90	0	Θ ₅ +Θ ₅ -in	180
6	0	0	+372	Θ ₆ +Θ ₆ -in	0

Le modèle géométrique direct :

Dans l'espace, nous utiliserons le formalisme de **Denavit-Hartenberg**

- 1- Placer les repères (Déjà fait)
- 2- Définir les variables articulaires et les paramètres géométriques (Le tableau précédent)
- 3- Définir les matrices de transformées homogènes :

$$T_i^{i-1} = T_z(\theta_i) T_z(d_i) T_x(a_i) T_x(\alpha_i)$$

$$= \begin{bmatrix} \cos \theta_i & -\sin \theta_i & 0 & 0 \\ \sin \theta_i & \cos \theta_i & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & a_i \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos \alpha_i & -\sin \alpha_i & 0 \\ 0 & \sin \alpha_i & \cos \alpha_i & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} \cos \theta_i & -\cos \alpha_i \sin \theta_i & \sin \alpha_i \sin \theta_i & a_i \cos \theta_i \\ \sin \theta_i & \cos \alpha_i \cos \theta_i & -\sin \alpha_i \cos \theta_i & a_i \sin \theta_i \\ 0 & \sin \alpha_i & \cos \alpha_i & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

4- Multiplier ces matrices :

- Vu qu'on a six repères, on a une multiplication de six matrices de transformées homogènes qui sont paramétrées par le tableau ci-dessus.
- Aussi vu que la multiplication analytique soit fastidieuse, le calcul a été fait par un code MATLAB qui sera expliqué par la suite.

$$T_{Final} = T_1^0 \cdot T_2^1 \cdot T_3^2 \cdot T_4^3 \cdot T_5^4 \cdot T_6^5$$

Planification des trajectoires des angles :

Pour la planification de la trajectoire on a choisi l'interpolation de degré 5 qui permet d'avoir des trajectoires de la forme :

$$q(t) = a_0 + a_1t + a_2t^2 + a_3t^3 + a_4t^4 + a_5t^5$$

L'ordre cinq du polynôme permet d'imposer 2 contraintes sur la vitesse et 2 autres sur l'accélération pour avoir un mouvement lisse et pour que notre actionneur puisse suivre les consignes (Réalisable physiquement) Par Contre cette méthode ne permet pas de fixer les vitesses et les accélérations maximales donc on doit allonger le temps final d'arrivée ou bien minimiser la différence Δq .

Pour déterminer les coefficients a_i il faut résoudre six équations qui contiennent les conditions initiales et finales.

On a comme données : q_{initial} , q_{final} , T_{final} .

Remarque : $q(t)$ est une matrice de six lignes.

Pour avoir la consigne en vitesse et en accélération il suffit de dériver :

$$\begin{cases} q(t) &= q_i + \left(\frac{10\Delta q}{t_f^3}\right) t^3 - \left(\frac{15\Delta q}{t_f^4}\right) t^4 + \left(\frac{6\Delta q}{t_f^5}\right) t^5 \\ \dot{q}(t) &= \left(\frac{30\Delta q}{t_f^3}\right) t^2 - \left(\frac{60\Delta q}{t_f^4}\right) t^3 + \left(\frac{30\Delta q}{t_f^5}\right) t^4 \\ \ddot{q}(t) &= \left(\frac{60\Delta q}{t_f^3}\right) t - \left(\frac{180\Delta q}{t_f^4}\right) t^2 + \left(\frac{120\Delta q}{t_f^5}\right) t^3 \end{cases}$$

Ces matrices seront par la suite les consignes de notre commande.

L'application de ces résultats est donnée par le programme MATLAB suivant :

La programmation

Une description résumée du script :

D'abord on a défini nos seules variables qui sont les angles des joints

```
syms T1 T2 T3 T4 T5 T6; % Les angles symboliques du notre bras robotique.  
T =[T1 T2 T3 T4 T5 T6]; % Le Vecteur Theta's
```

Ensuite on a défini les conditions initiales (décalages des angles par rapport à Z_i), on a choisi la configuration initiale (Référence) de telle façon qu'elle soit identique au schéma cinématique :

```
Tin = [+pi/2 +pi/2 0 pi pi 0]'; % Vecteur de l'initialisation des angles.  
      [%[+90 +90 0 180 180 0]'];
```

Après on a programmé le tableau DH :

```
% La Matrice DH qui contient le tableau des parametres Denavit-Hartenberg  
% du notre robot KUKA KR1000 Titan
```

```
DH = [+600 +pi/2 1100 T(1)+Tin(1);...  
      +1400 0 0 T(2)+Tin(2);...  
      +65 +pi/2 0 T(3)+Tin(3);...  
      0 +pi/2 +650+550 T(4)+Tin(4);...  
      0 +pi/2 0 T(5)+Tin(5);...  
      0 0 +372 T(6)+Tin(6)];
```

Ensuite, on a multiplié les matrices de transformées homogènes pour obtenir MGD :

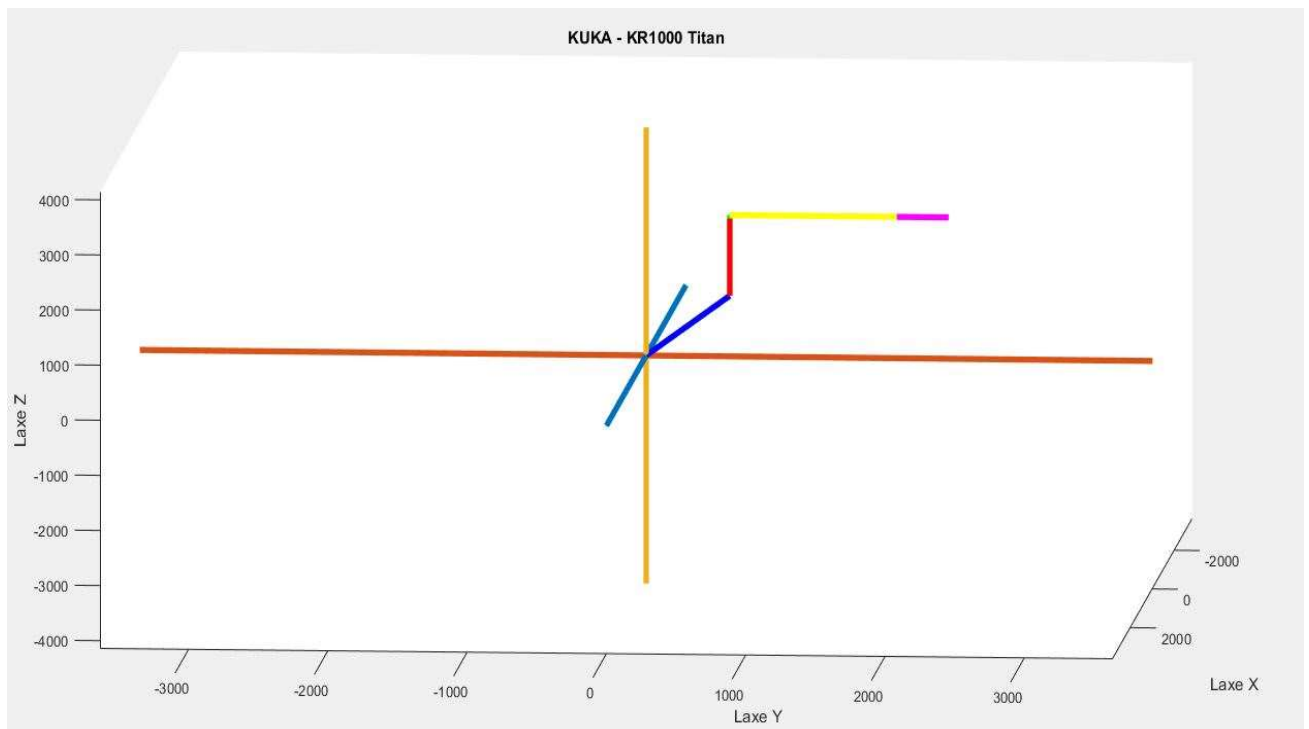
```
Tr = eye(4); % L'initialisation de la multiplication a 1(un)  
Joints_Ends = []; % Les positions des extrémités des liens par rapport au repere fixe  
% Joints_Ends : Utile pour le dessin du robot et l'animation  
% J'ai pas utilisé les orientations des liens pour simplifier le Programme (ce n'est pas le but).  
for i = 1 : length(T)  
    % Les matrices de transformation homogenes pour chaque 2 reperees succesives.  
    L = [cos(DH(i,4)) -cos(DH(i,2))*sin(DH(i,4)) sin(DH(i,2))*sin(DH(i,4)) DH(i,1)*cos(DH(i,4));...  
         sin(DH(i,4)) cos(DH(i,2))*cos(DH(i,4)) -sin(DH(i,2))*cos(DH(i,4)) DH(i,1)*sin(DH(i,4));...  
         0 sin(DH(i,2)) cos(DH(i,2)) DH(i,3);...  
         0 0 0 1];  
  
    Tr = Tr * L; % Le modele Geometrique Direct  
    Joints_Ends = [Joints_Ends Tr(1:3,4)]; % Option d'affichage  
end
```

A ce stade des simulations ont été réalisées pour vérifier la bonne programmation des formules (les lignes qui servent à l'affichage viennent dans la suite du script)

Par exemple pour la configuration initiale :


```
Tin = [+pi/2 +pi/2 0 pi pi 0]'; % Vecteur de l'initialisation des angles.
      % [+90 +90 0 180 180 0]';
```

On a eu la configuration suivante (Exactement ce qu'on veut) :

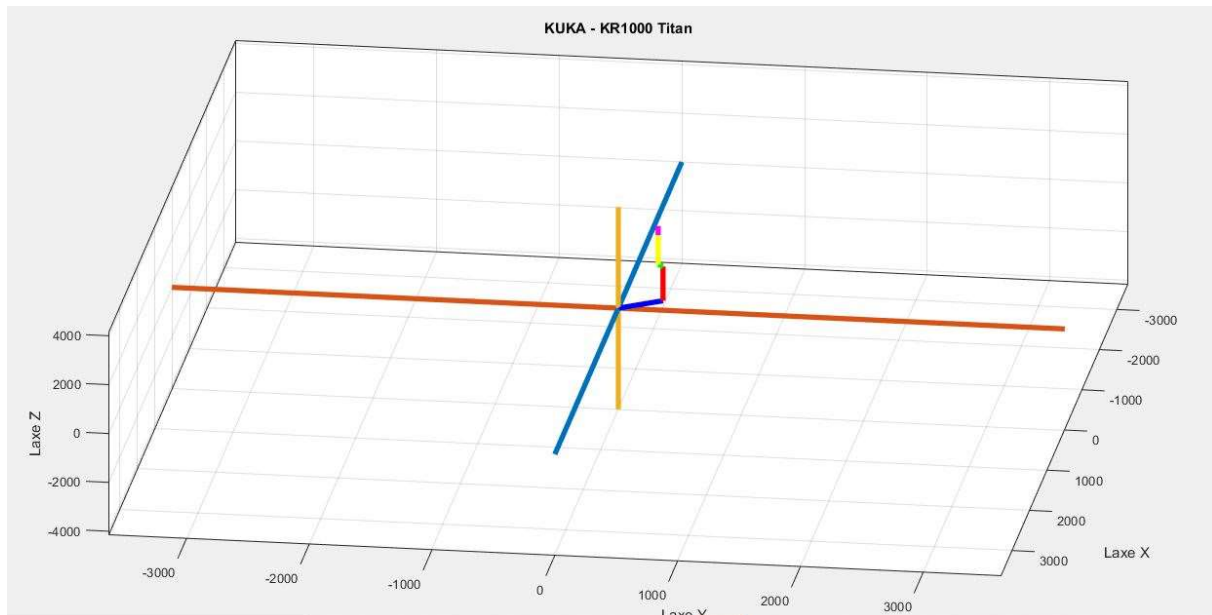


Remarques :

- Entre les liens rouge et jaune il existe un lien vert qui n'est pas bien visible car il a une longueur petite devant les autres liens (65)
- Le lien jaune (1200) contient deux liens (550 et 650) (ils ne sont pas distingués car l'affichage s'intéresse plus aux longueurs des liens et non pas les orientations << une complication non nécessaire pour ce projet >>) (Le lien 650 tourne autour du roulis du lien 550 donc ils seront toujours alignés).

On essaye une autre configuration, on va tourner la première articulation(bleue) par -45° et la troisième (Verte) par $+90^\circ$.

```
Tin = [(+pi/2-pi/4) +pi/2 (0+pi/2) pi pi 0]'; % Vecteur de l'initialisation des angles.
```



On a obtenu les configurations attendues donc jusqu'à maintenant le programme fonctionne correctement.

Après on a fait la planification de la trajectoire dans l'espace articulaire :

```

53  % La planification de la trajectoire
54
55 - Ts = 50; %Time samples : la période d'échantillonnage
56 - Qin = [0 0 0 0 0]; % La configuration de départ ( Nulle C.A.D pareil au schéma cinématique)
57  %Qf = [0.7 -0.4 10 0 1.5 0.5];
58 - Qf = [pi/2 0 0 0 0 0]; % La configuration d'arrivée
59 - DeltaQ = Qf - Qin ; % Constante répétitive dans les formules
60 - TimeI = 0 ; % Temps de départ
61 - TimeF = 10 ; % Temps d'arrivée
62 - t = TimeI : (TimeF-TimeI)/Ts : TimeF; % Le vecteur Temps
63  %
64  % q : Positions Ang ; qdot : Vitesses Ang ; q2dot : Accelerations Ang dim : 6xlength(t)
65 - q = [];
66 - qdot = [];
67 - q2dot = [];
68 - for i = 1:length(DeltaQ)
69 -   q = [q ; Qin(i)+(10*DeltaQ(i)/(TimeF^3))*t.^3-(15*DeltaQ(i)/(TimeF^4))*t.^4+(6*DeltaQ(i)/(TimeF^5))*t.^5];
70 -   qdot = [qdot; (30*DeltaQ(i)/(TimeF^3))*t.^2-(60*DeltaQ(i)/(TimeF^4))*t.^3+(30*DeltaQ(i)/(TimeF^5))*t.^4];
71 -   q2dot = [q2dot; 60*DeltaQ(i)/(TimeF^3)*t-180*DeltaQ(i)/(TimeF^4)*t.^2+120*DeltaQ(i)/(TimeF^5)*t.^3];
72 - end

```

Ensuite on a programmé un script qui extrait les points de départ et d'arrivée ensuite il calcul une droite rectiligne qui passe par ces deux points pour définir la trajectoire opérationnelle :

```

73 % La planification d'une trajectoire rectiligne dans l'espace operationnel
74
75 - P_d = MGD; % P_d : Point de départ (Position de l'effecteur par rapport au repere fixe)
76 - P_a = MGD; % P_a : Point d'arrivée
77
78 % L'extraction des positions de départ et d'arrivée à partir du MGD
79 % c'est l'intersection de la quaterieme colonne et les trois premieres lignes
80 - for i = 1 : 6
81     % Substitution par les valeurs numériques dans le model Symbolic
82     P_d = subs(P_d,T(i),Qin(i));
83     P_a = subs(P_a,T(i),Qf(i));
84 - end
85 - P_d = double(P_d(1:3,4)); % L'extraction
86 - P_a = double(P_a(1:3,4));
87
88 % La définition de la droite operationnelle
89 % Définir les vecteurs de la droite selon X,Y,Z (Projections) Pour que je puisse dessiner la
90 % rectiligne au fur et à mesure en animation avec la trajectoire
91 % articulaire
92 % (Le nombre des echantillons de la trajectoire articulaire = Nombre des Ech Rectiligné)
93
94 - if((P_d(1)-P_a(1))>0)
95 - Line_Oper_i = P_d(1) : -abs(P_d(1)-P_a(1))/Ts : P_a(1);
96 - elseif((P_d(1)-P_a(1))<0)
97 - Line_Oper_i = P_d(1) : +abs(P_d(1)-P_a(1))/Ts : P_a(1);
98 - elseif((P_d(1)-P_a(1))==0)
99 - Line_Oper_i = zeros(1,Ts+1)+P_d(1); % Projection de la droite sur X
100 - end
101
102 - if((P_d(2)-P_a(2))>0)
103 - Line_Oper_j = P_d(2) : -abs(P_d(2)-P_a(2))/Ts : P_a(2);
104 - elseif((P_d(2)-P_a(2))<0)
105 - Line_Oper_j = P_d(2) : +abs(P_d(2)-P_a(2))/Ts : P_a(2);
106 - elseif((P_d(2)-P_a(2))==0)
107 - Line_Oper_j = zeros(1,Ts+1)+P_d(2); % Projection de la droite sur Y
108 - end
109
110 - if((P_d(3)-P_a(3))>0)
111 - Line_Oper_k = P_d(3) : -abs(P_d(3)-P_a(3))/Ts : P_a(3);
112 - elseif((P_d(3)-P_a(3))<0)
113 - Line_Oper_k = P_d(3) : +abs(P_d(3)-P_a(3))/Ts : P_a(3);
114 - elseif((P_d(3)-P_a(3))==0)
115 - Line_Oper_k = zeros(1,Ts+1)+P_d(3); % Projection de la droite sur Z
116 - end
117
118 - Line_Oper = [Line_Oper_i;Line_Oper_j;Line_Oper_k]; % Les coordonnées finales de la droite

```

Finalement on a programmé la partie qui compile tout le calcul et faire l'animation (Vu que le script est long, il n'était pas posé dans le rapport mais il est dans le fichier Matlab)

Aussi on a programmé une partie qui compare entre les deux trajectoires et d'autres fonctionnalités qui seront détaillées par la suite.

La simulation :

Remarque : J'ai pris TimeF identique pour toutes les articulations pour simplifier la programmation dans un premier temps.

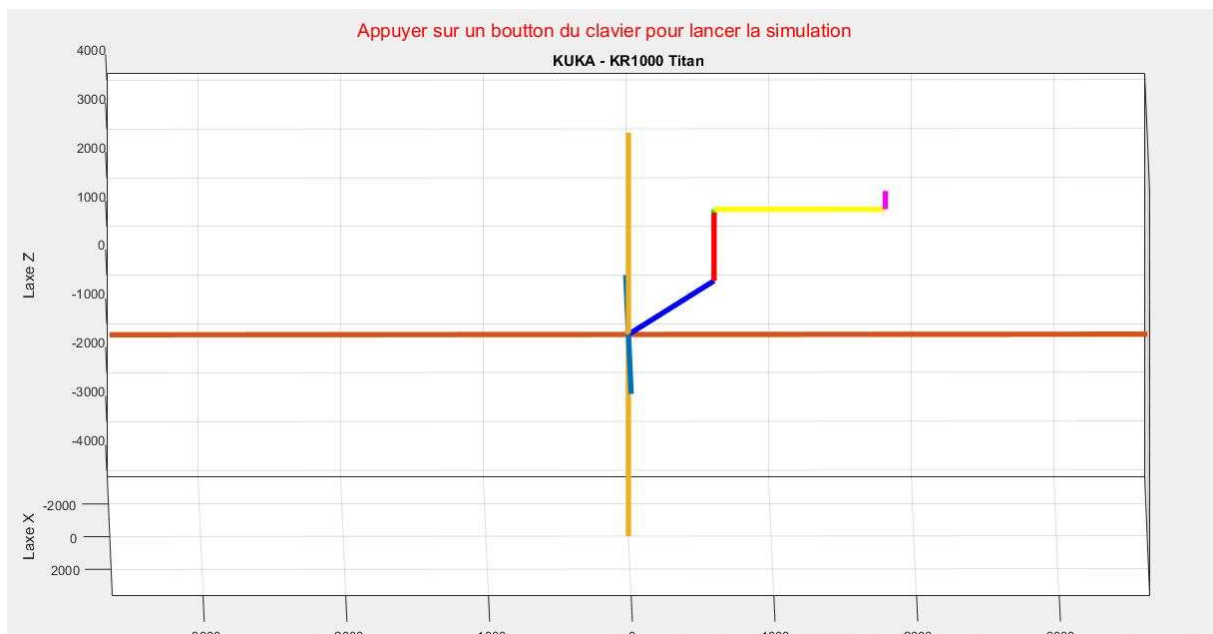
Simulation 1 :

Qin (Configuration de départ) (°)	[0 0 0 0 90 45]
Qf (Configuration d'arrivée) (°)	[90 0 270 0 45 45]
Time_F (Temps d'arrivée) (s)	10
Ts (Sampling Time) (s)	50

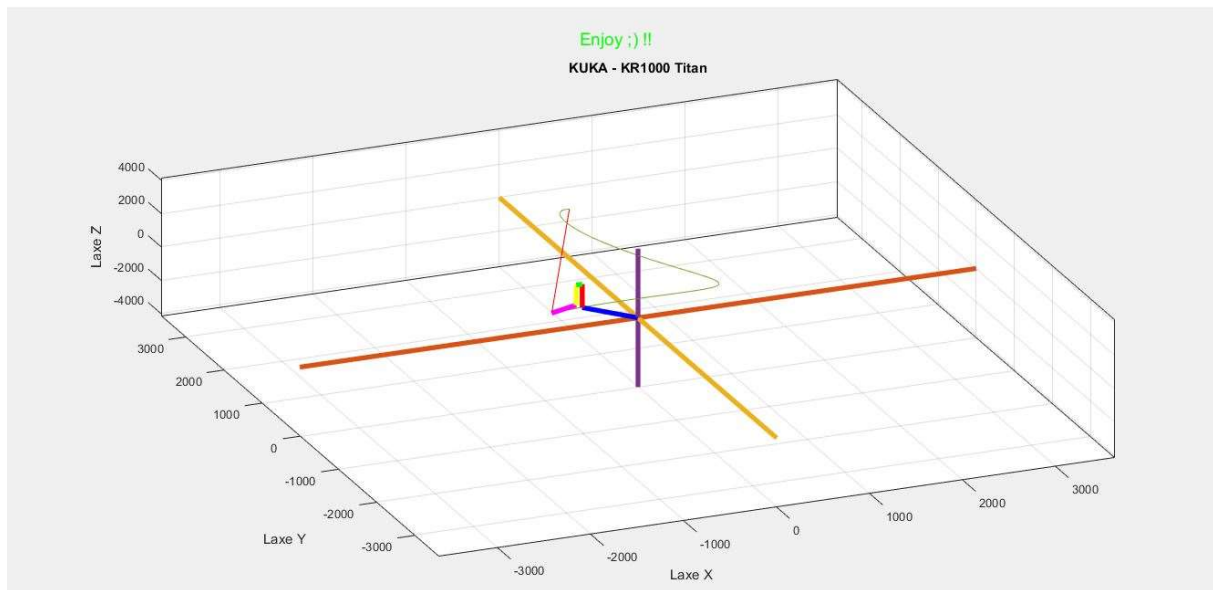
Remarques :

- La configuration de référence est celle du schéma cinématique (C.A.D Qin = [0 0 0 0 0 0] Correspond au schéma cinématique).
- Le sixième angle n'apparaît pas sur la figure car il correspond à l'effecteur final.
- J'ai appliqué des changements seulement sur trois angles donc en principe on génère seulement trois trajectoires (Sera vérifié dans les figures qui suivent).
- J'ai appliqué un angle final (3) qui est en dehors de la zone de fonctionnement.

Le départ :

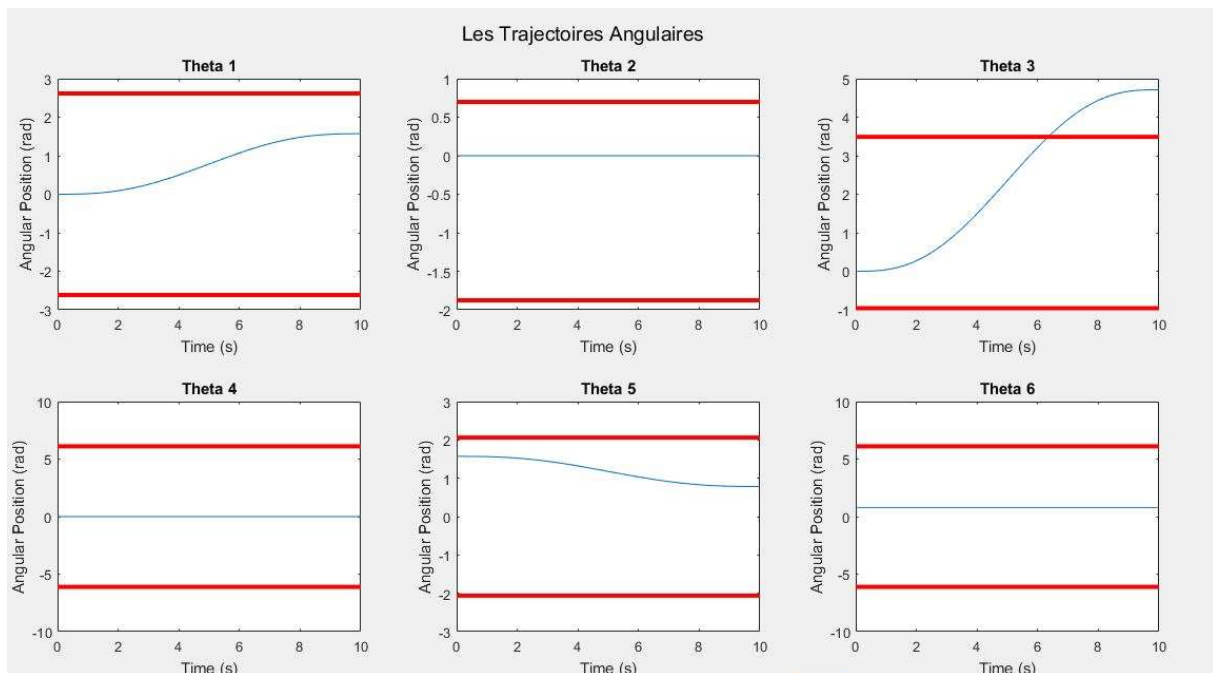


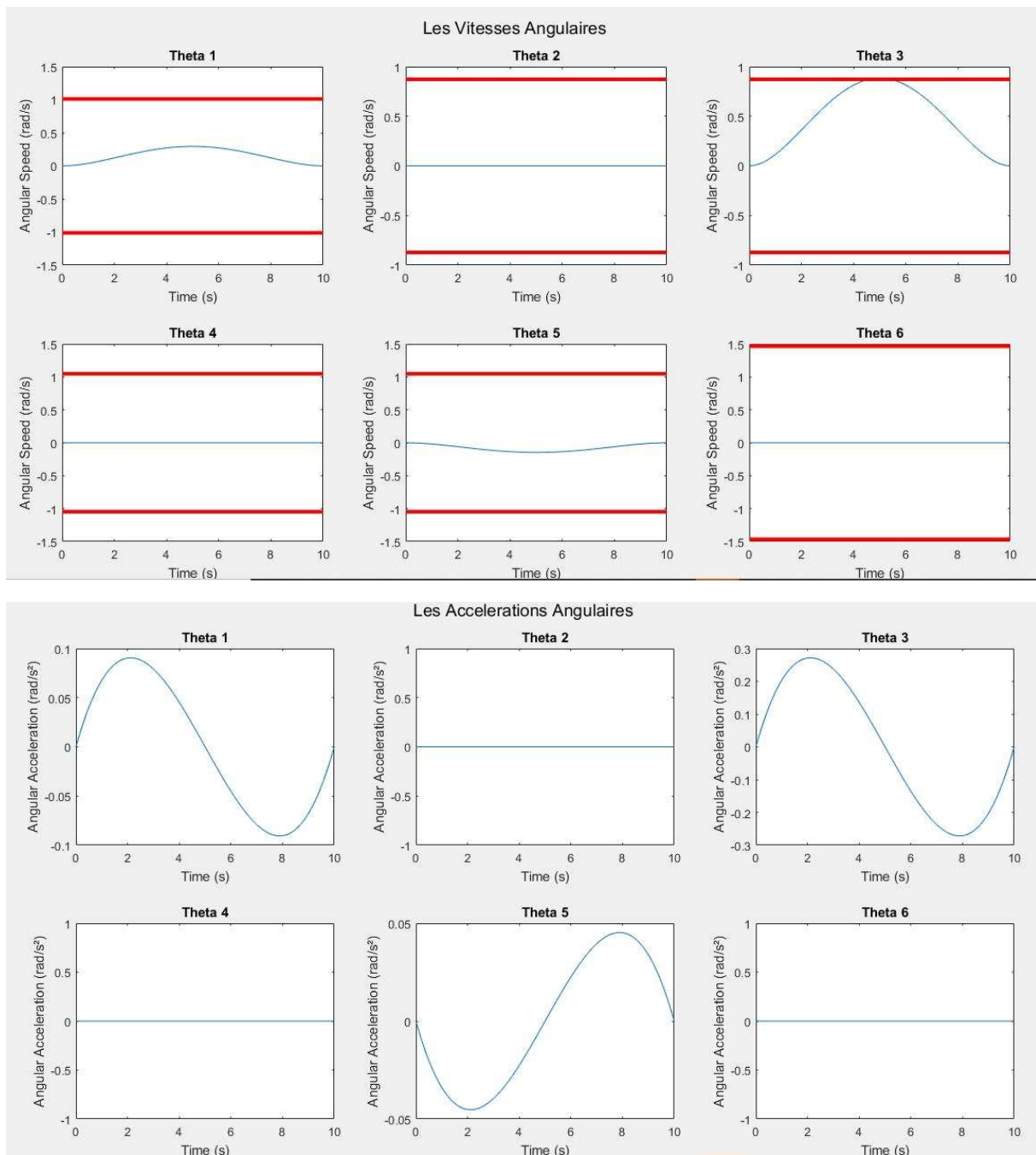
L'arrivée :



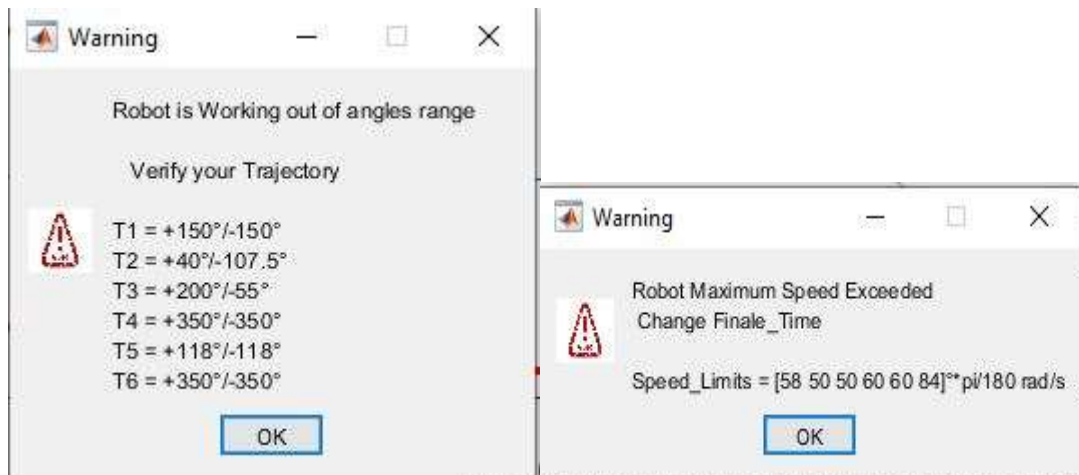
- On voit bien que notre script nous affiche dans la première figure : l'animation, La trajectoire opérationnelle, la trajectoire articulaire, les positions des liens au cours du temps et d'autre informations . . .

Les autres figures de notre script :

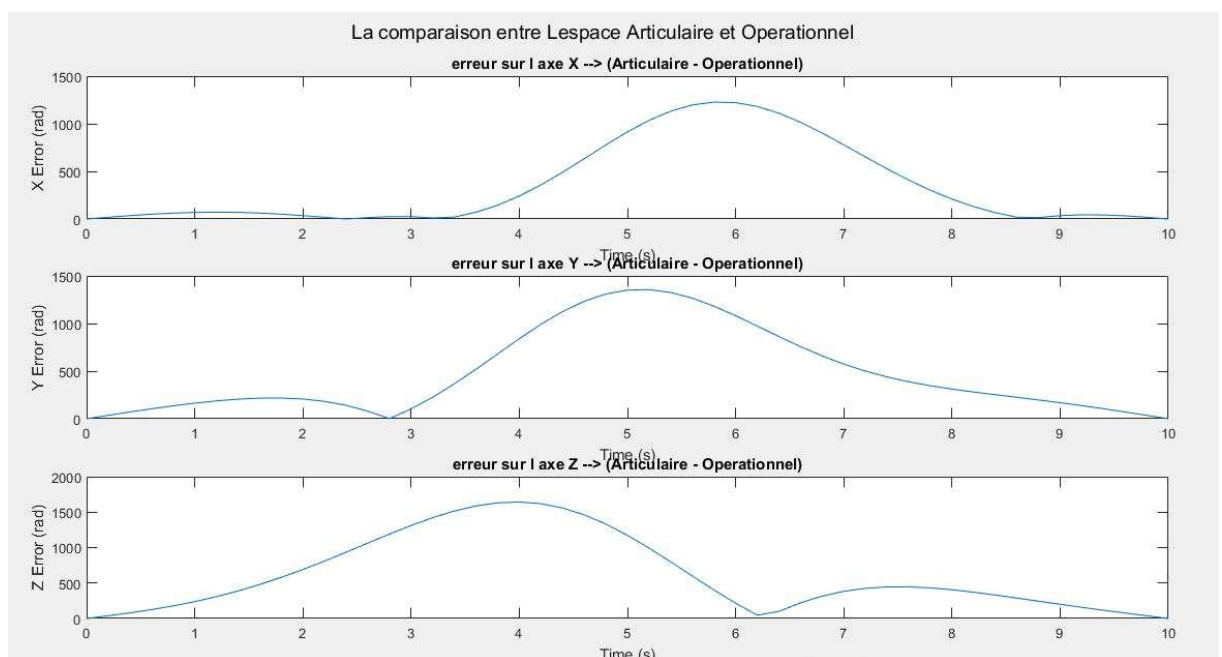




- Notre scripte nous affiche aussi la planification des trajectoires pour les différents angles, les vitesses angulaires, les accélérations angulaires (les consignes) et aussi il trace les limites sur les angles et les vitesses en rouge que notre robot ne peut pas les dépasser, et aussi il nous donne des « Warnings » lorsque on les dépasse, comme dans ce cas on voit bien que la trajectoire et la vitesse du troisième angle dépassent les limites donc on obtient les messages suivants :



- Donc on doit rectifier notre trajectoire en changeant les angles d'arrivée, et nos vitesses en changeant le temps d'arrivée (Le temps d'arrivée est unique pour toutes les angles pour ne pas trop encombrer le script).
- Ces Warnings sont trop utiles pour rester dans la zone de fonctionnement (Ils n'interrompent pas la simulation).
- Aussi, on a des messages qui précisent les limites dépassées dans la commande window.
- On voit bien à partir des graphes des trajectoires, vitesses, accélérations que c'est une planification par interpolation de degré cinq.



- La dernière figure nous donne l'erreur cartésienne entre la trajectoire de l'effecteur articulaire et la trajectoire opérationnelle.
- On voit bien que l'erreur est large car l'espace articulaire permet de suivre et commander les angles des articulations et non pas la position de l'effecteur.

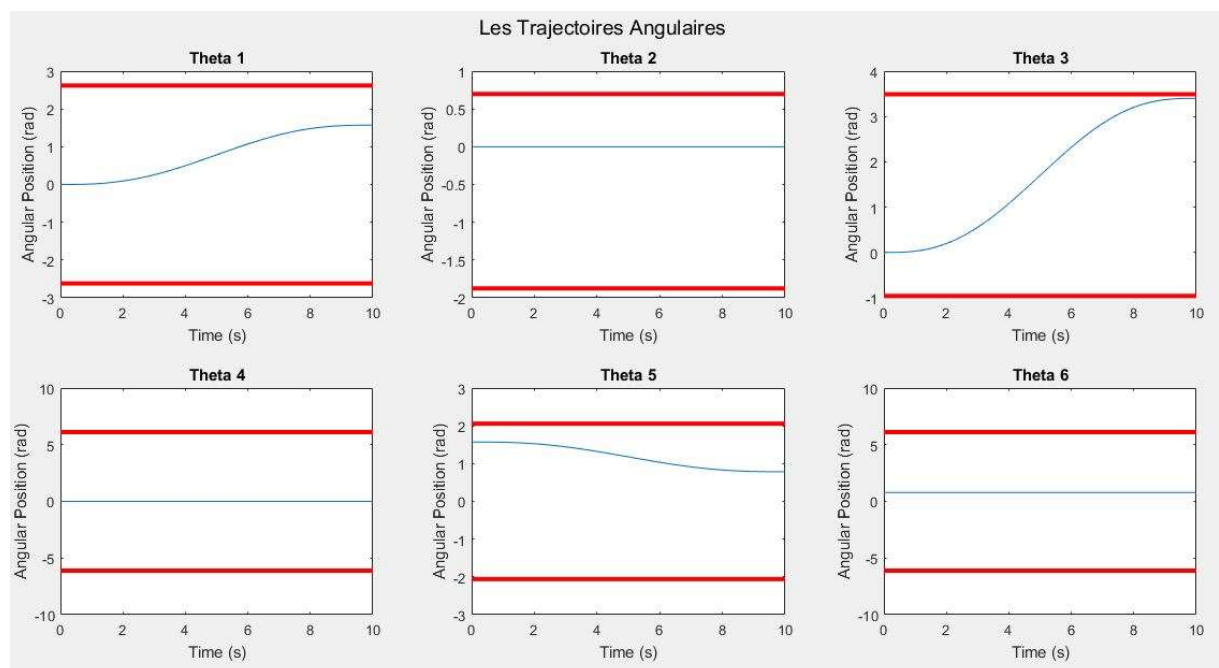
Simulation 2 : (Changement des conditions d'arrivée pour rester dans la zone de fonctionnement) :

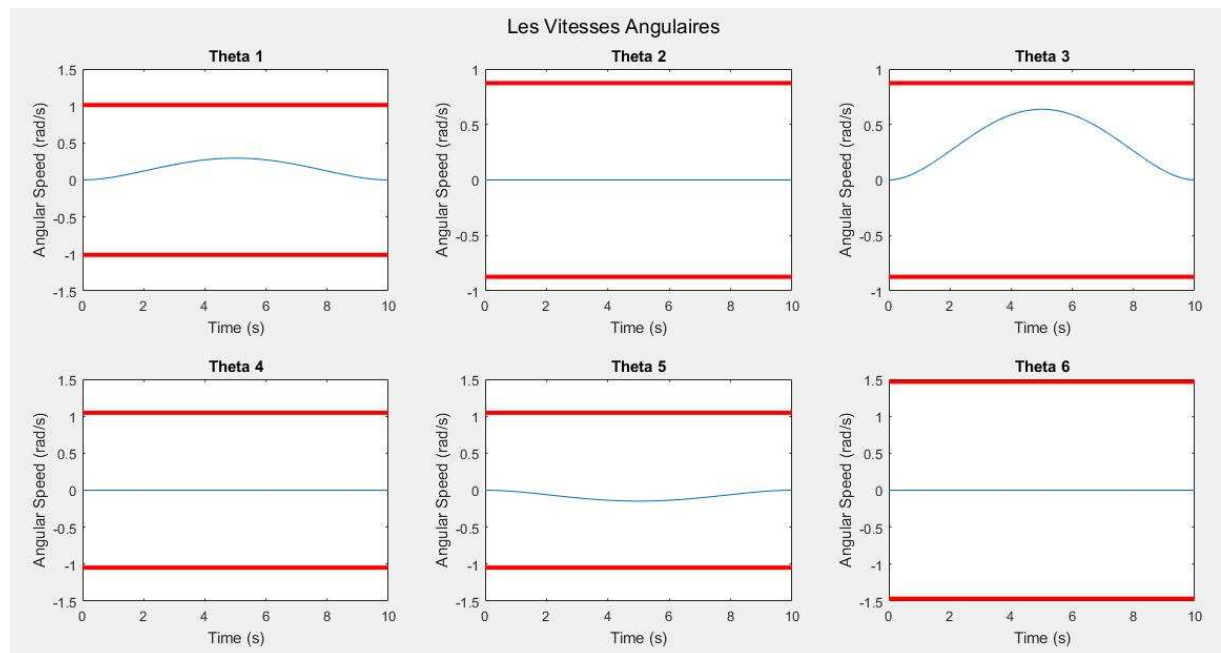
Qin (Configuration de départ) (°)	[0 0 0 0 90 45]
Qf (Configuration d'arrivée) (°)	[90 0 194.8057 0 45 45]
Time_F (Temps d'arrivée) (s)	10
Ts (Sampling Time) (s)	50

Remarques :

- On a changé le troisième angle qui était dans la zone interdite.
- Le réglage de la trajectoire (Point de départ et point d'arrivée) peut régler la trajectoire (Bien évident) et aussi la vitesse interdite mais l'allongement du temps d'arrivée peut régler seulement la vitesse et non pas la trajectoire interdite (Logique 😊).
- Le changement des angles change forcément le point d'arrivée (Car on utilise la planification articulaire) donc notre trajectoire opérationnelle change respectivement.
- Pour faire une planification de trajectoire opérationnelle le travail sera plus compliqué car il faut utiliser le MGD et MGI et des techniques de calcul et de résolution plus compliqués (La logique floue par exemple – Fuzzy Logic 🧠-....)

Les résultats :






- On voit bien que notre trajectoire est maintenant dans la zone acceptée de fonctionnement.
- L'angle 3 est trop proche à la limite, de préférence de ne pas trop rapprocher aux limites car par la suite on sait que nos actionneurs et correcteurs ne sont pas idéaux donc il va avoir toujours une marge d'erreur qui peut amener notre robot dans des zones interdites.
- On peut rendre notre script avec plus de fonctionnalités mais sûrement plus de complexité, Par exemple on ajoute des boucles qui permettent de planifier des points intermédiaires, qui prend des temps d'arrivée différents pour chaque angle, qui permet de vous proposer le type de planification voulu (Interpolation 1,3,5 ; Bang-Bang ; Trapèze ...), la commande des actionneur (J'ai déjà programmé un script avec la commande des actionneurs mais je n'ai pas eu le temps pour l'implémenter dans ce script) ...

Conclusion Finale :

- Dans ce projet, on a étudié la planification d'une trajectoire libre du bras robotique KUKA KR1000 Titan dans l'espace articulaire.
- On a planifié des trajectoires angulaires par interpolation de l'ordre 5.
- On a défini le dimensionnement de notre robot et aussi ses limites en angles et en vitesses.
- On a fait une comparaison entre sa trajectoire articulaire (utilisable pour les trajectoires libres et les trajectoires libres passant par des points intermédiaires) et la trajectoire opérationnelle (utilisable pour les trajectoires contraintes et les trajectoires contraintes passant par des points intermédiaires).
- Finalement on a validé les points développés dans le cours par rapport aux avantages et les inconvénients de chaque espace :
 - L'espace articulaire :
 - Avantages :
 - Mouvement minimal sur chaque articulation : car on les commande directement.
 - Ne nécessite pas le calcul du MGI et MGD : (On a utilisé le MGD juste pour définir les positions des extrémités des liens).
 - Mouvement non affecté par le passage par des configurations singulières : on peut programmer une trajectoire qui ne passe pas par les points singuliers.
 - Contraintes de couples maximum et vitesses maximum sont connues : ils sont précisés par les formules des trajectoires sur chaque angle.
 - Inconvénients :
 - Déplacement non contrôlé du robot dans l'espace opérationnel : Car on ne connaît pas les trajectoires de chaque angle qui permettent d'avoir le mouvement opérationnel de l'effecteur (il faut passer par le MGI donc la procédure opérationnelle forcément).
 - L'espace opérationnel :
 - C'est exactement l'inverse de l'espace articulaire : on peut contrôler la trajectoire de l'effecteur mais on doit faire des grands calculs des MGD et MGI, aussi on risque de passer ou de se rapprocher des points singulières la chose qui délivre des consignes non réalisables pour la commande des actionneurs (Il existe des techniques pour résoudre ces problèmes).
- Ce Projet m'a permis vraiment de s'introduire dans le monde des bras robotiques, j'ai bien assimilé beaucoup de notions essentielles et indispensables pour leur fonctionnement (MGD, MGI, La commande, le positionnement des liens, La planification d'une trajectoire, La convention DH, La représentation 3d, Les repères ...).
- Ces notions ne vont pas seulement me permettre de comprendre le module Robotique appliquée mais aussi de me faciliter le travail sur mes propres projets qui nécessitent forcément la compréhension de la mécanique rationnelle (les repères et le

positionnement), l'automatique (la commande, la cinématique, la dynamique...), la programmation ... tels que la conception des drones (quad-copters, aile fixe ...).

- Et aussi il m'a donné l'inspiration et le courage pour réaliser mon propre bras robotique que je vais programmer ça conception dans les mois prochains (ça sera beaucoup plus compliqué mais rien n'empêche -No Pain No Gain ).

Travail supplémentaire :

- J'ai réalisé un programme qui commande le robot dans l'espace articulaire.
- J'ai modélisé le système par une représentation d'états des pendules qui ne subissent pas à la gravité (pour simplifier le modèle) seulement les moments des actionneurs (J'ai pris les moments cinétiques $J = 1$ aussi pour la simplification).

$$J \frac{d^2 \theta}{dt^2} = U + 0 * P \text{ (Pesanteur)}$$

$$\begin{cases} \dot{X}_{1i} = X_{2i} \\ \dot{X}_{2i} = U_i \end{cases}$$

$X1$: représente les angles

$X2$ les vitesses angulaires

U : les commandes

i : représente le numéro de l'angle

$$U = F (X - X_f)$$

F : Représente le gain de réglage obtenu par retour d'état (La fonction 'place' du Matlab).

- Aussi je n'ai pas imposé une trajectoire (Faute du temps) mais seulement j'ai appliqué l'algorithme de la stabilisation autour d'un point voulu (Il part à partir des angles initiaux vers des angles finals librement en suivant la commande).
- La commande de poursuite de trajectoire que j'ai voulue programmer :
$$U = \ddot{r} + F_1(X_1 - r) + F_2(X_2 - \dot{r})$$
 r : Représente la fonction de la trajectoire
- Il est un peu oscillatoire car je n'ai pas bien réglé les gains de résonance (Tuning).

Remarque : Les codes Matlab sont inclus dans la pièce jointe :

- [Projet_RA_TR1000_TITAN_BELBECIR_Riyadh_GSA_2020_2021.m](#)
 - o Ce fichier est le programme principal qui est demandé pour le Projet.
 - o Il est fonctionnel, vous lui donnez seulement la configuration initiale et la configuration finale et le temps final.
- [Travail_Supplementaire_Commande.m](#)
 - o C'est le travail de plus que le temps ne m'a pas suffi pour l'implémenter dans le programme principal (Il est fonctionnel, il suffit de lancer la simulation).