Sorbonne Université

Compte rendu Projet Robotique

Par :
Touzari Lisa 28617271
Bekdouche Lina Insaf 28620402

Introduction

L'objectif de ce projet est la caractérisation d'un robot de type "bras" fonctionnant dans le plan, et réalisé à partir de moteurs simulés tous identiques.Pour cela on va utiliser un robot simulé "réaliste". Il s'agira en particulier d'établir différents modes de contrôle permettant au bras de réaliser une tâche.

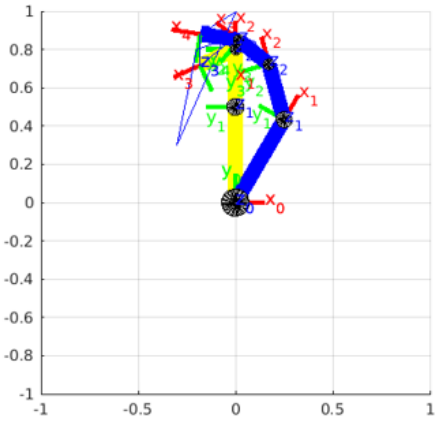
Modélisation et commande géométrique d'un robot

Construction et affichage d'un 3R plan

D'abord on fait appel à la fonction declarerobot définie dans le fichier du même nom afin de déclarer un robot 3R plan, nommé r3, en configuration initiale [0 0 0]T et avec des segments de longueur 0.5m,0.3m et 0.2m respectivement. On déclare un second robot de type 4R plan, nommé r4, en configuration initiale et avec des segments de longueur 0.5m, 0.3m, 0.2m et 0.2m respectivement.

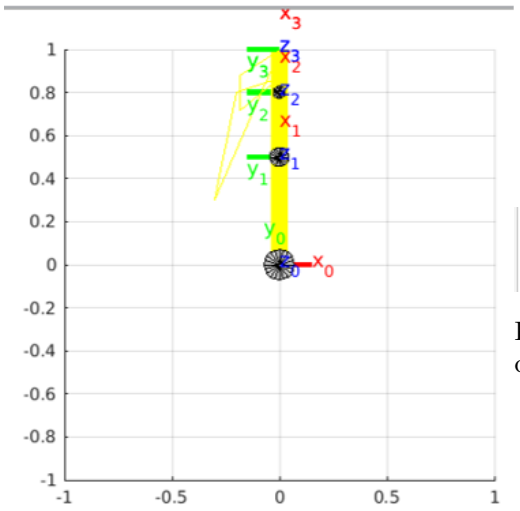
```
n1=3
Trans1=[0.5,0.3,0.2]
q1=[pi/2,0,2]
r3 = declare_robot(n1,Trans1,q1)

n=4
Trans=[0.5,0.3,0.2,0.2]
q=[pi/3,pi/4,pi/5,pi/6]
r4 = declare_robot(n,Trans,q)


affiche_robot(r3,1,1,1,[1,1,0],1)
affiche_robot(r4,1,1,0,[0,0,1],1)
```

Création des robots R3 et R4



Affichage des robots R3 et R4

On affiche le robot 3R dans la configuration $[\pi/2, 0, 0]$



Affichage du robot R3

```
q1=[pi/2,0,0]
r3 = declare_robot(n1,Trans1,q1)
affiche_robot(r3,1,1,1,[1,1,0],1)
```

Programme affichage du robot R3 dans la configuration $[\pi/2, 0, 0]$

Fonctionnalités élémentaires

On écrit une fonction Matlab qui retourne une matrice de rotation d'un angle a autour de l'axe Z :

```
function [R] = rotation_Z_etu(angle)
%  ROTATION_Z retourne la matrice de rotation 3D
% correspondant à une rotation d'angle 'angle' autour de x
%  R = rotation_Z(angle)
R = [cos(angle),-sin(angle), 0;
    sin(angle), cos(angle), 0;
    0,0,1]

end
```

La fonction rotation

```
Rot=rotation_Z_etu(r3.q(1))
```

```
Rot =

    0.0000   -1.0000         0
    1.0000    0.0000         0
         0         0    1.0000
```

Test de la fonction rotation

On écrit une fonction Matlab qui retourne l'angle de rotation autour de l'axe Z à partir d'une matrice de rotation passée comme argument (et dont on suppose qu'elle représente bien une rotation d'axe Z) :

```
function [a] = inv_rotation_Z_etu(R)
% INV_ROTATION_Z retourne l'angle de rotation autour de l'axe Z à partir
% d'une matrice de rotation passée comme argument (et dont on suppose
% qu'elle représente bien une rotation d'axe Z).
%   [a] = inv_rotation_Z(R)
a=acos((R(1,1)+R(2,2)+R(3,3)-1)/2)
end
```

La fonction inverse rotation

```
a=inv_rotation_Z_etu(Rot)
```

```
a =

    1.5708
```

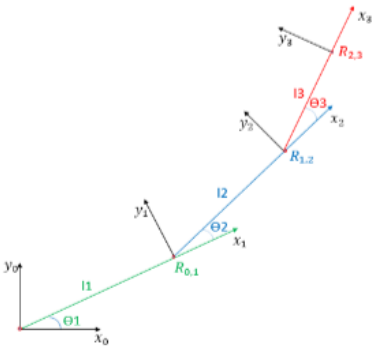
Test de la fonction inverse rotation

Calcul du modèle géométrique direct et inverse

à travers cette section nous allons expliquer et montrer le fonctionnement des outils de calcul des modèles géométrique direct et inverse pour un robot.

modèle géométrique direct

Le modèle géométrique direct est l'ensemble des relations qui permettent d'exprimer la situation de l'organe terminal, c'est-à-dire les coordonnées opérationnelles du robot, en fonction de ses coordonnées articulaires.



Modèle géométrique direct

Pour développer la fonction du modèle géométrique direct on réalise d'abord une fonction qui retourne la matrice de de transformation homogène. Comme suit pour le 3R

```
function [H] = trans_homogene_etu(R,T)
% TRANS_HOMOGENE retourne la matrice de transformation homogene 3D
% correspondant à une rotation R et à une translation T successives.

H = [R(1,:) T(1);
    R(2,:) T(2);
    R(3,:) T(3);
    0 0 0 1];

end
```

Fonction qui rend la matrice de transformation homogène

Tel que R représente la matrice suivante :

```
function [R] = rotation_Z_etu(angle)
%  ROTATION_Z retourne la matrice de rotation 3D
% correspondant à une rotation d'angle 'angle' autour de x
%  R = rotation_Z(angle)
R = [cos(angle),-sin(angle), 0;
    sin(angle), cos(angle), 0;
    0,0,1]

end
```

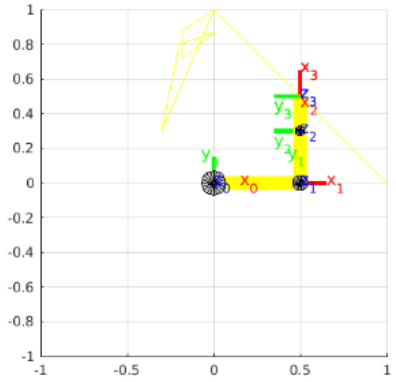
Fonction qui réalise la rotation

Et T représente le vecteur de translation.
Après avoir tester le bon fonctionnement des fonctions on viens exprimer notre modèle géométrique direct :

```
function [TH1, TH1p] = mod_geo_dir_etu(robot,segment)
% MOD_GEO_DIR retourne les matrices de transformation homogène associées au
% corps 'segment' d'un robot 'robot'
%   [TH1, TH1p] = mod_geo_dir(robot,segment)

TH1 = eye(4);
TH1p = eye(4);
for i=1:segment

    TH1p = trans_homogene_etu(rotation_Z_etu(robot.q(i)),zeros(3,1))*TH1
    TH1 = trans_homogene_etu(eye(3),[robot.T(i);0;0])*TH1p

end

end
```

Fonction du modèle géométrique direct

On exécute un test pour vérifier le bon fonctionnement de notre fonction. On choisie une configuration :



Configuration de notre robot

On obtient :

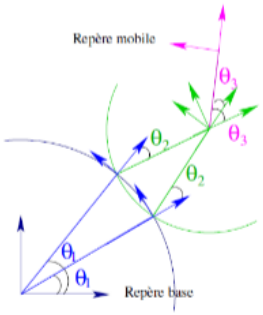
```
Ths3 =

    0.0000   -1.0000         0    0.5000
    1.0000    0.0000         0    0.5000
         0         0    1.0000         0
         0         0         0    1.0000
```

La matrice de transformation homogène du derniers segment

On peut déduire à partir de la matrice de transformation homogène du derniers segment les coordonnées opérationnelles du robot.

modèle géométrique inverse

Le modèle géométrique inverse, quant à lui, consiste à trouver les coordonnées articulaires qui amènent l'organe terminale dans une situation désirée spécifiée par ses coordonnées opérationnelles.



$$\cos\alpha = \frac{YZ - \epsilon X\sqrt{X^2+Y^2-Z^2}}{X^2+Y^2}$$

$$\sin\alpha = \frac{XZ + \epsilon Y\sqrt{X^2+Y^2-Z^2}}{X^2+Y^2}$$

avec $\epsilon = +/-1$.

Fonction du modèle géométrique inverse

Notre modèle géométrique inverse s'écrit comme suit :

```
function [q1,q2] = mod_geo_inv_3Rplan(robot,x,y,theta)
% MOD_GEO_INV_3RPLAN retourne les configurations du robot correspondant
% à une position cartésienne 'x', 'y' et à une orientation 'theta' dans le plan
%   [q1,q2] = mod_geo_inv_3Rplan(robot,x,y,theta)


q1 = zeros(3,1);
q2 = zeros(3,1);

l1=robot.T(1);
l2=robot.T(2);
l3=robot.T(3);

w1=x-l3*cos(theta);
w2=y-l3*sin(theta);

cq2=(w1*w1+w2*w2-l1*l1-l2*l2)/(2*l1*l2);

if abs(cq2)>1.0
    cq2=sign(cq2);
end

sqa2=sqrt(1.0-cq2*cq2);
q1(2)=atan2(sqa2, cq2);
sqb2=-sqa2;
q2(2)=atan2(sqb2, cq2);

k1a=l1+l2*cos(q1(2));
k2a=l2*sin(q1(2));
cqa1 =(w1*k1a + w2*k2a)/(k1a*k1a + k2a*k2a);
sqa1 = (w1*k2a + w2*k1a)/(k1a*k1a + k2a*k2a);
q1(1)=atan2(sqa1,cqa1);

k1b = l1 + l2*cos(q2(2));
k2b = l2*sin(q2(2));
cqb1 = (w1*k1b + w2*k2b)/(k1b*k1b+k2b*k2b);
sqb1 = (-w1*k2b + w2*k1b)/(k1b*k1b+k2b*k2b);
q2(1)= atan2(sqb1, cqb1);

q1(3) = theta- q1(2) - q1(1);
q2(3) = theta- q2(2) - q2(1);




end
```

Cette fonction nous donnera deux solutions possible Qa et Qb deux vecteur qui représente les coordonnées articulaires qui amènent l'organe terminale vers notre cible, les deux sont juste. On exécute un test pour vérifier le bon fonctionnement de notre fonction. On choisie une configuration :

```
[qA2,qB2] = mod_geo_inv_3Rplan_etu(r,0.7,0.2,pi/3)
```

Test du modèle géométrique inverse

```
qA2 =

    0.5433
    1.5017
   -0.9978
```

Résultats obtenue

Teste de la fonction pour la position [1.0 1.0] et l'orientation $3\pi/2$:

```
[qA3,qB3] = mod_geo_inv_3Rplan_etu(r,1.0,1.0,3*pi/2)
r = declare_robot(n,Tr,qA3)
[Ths, Thsp]=mod_geo_dir_etu(r,3)
```

```
qA3 =

    0.8761
         0
    3.8363
```

```
qB3 =

    0.8761
         0
    3.8363
```

```
Ths =

   -0.0000    1.0000         0   -0.4146
   -1.0000   -0.0000         0   -0.5121
         0         0    1.0000         0
         0         0         0    1.0000
```

Résultats du modèle direct

Test des modèles direct et inverse pour l'orientation $3\pi/2$

On remarque que les résultats du modèle géométrique ne correspond pas à la position désirée, cela est dû au fait que la position désirée ne peut pas être atteinte par notre robot et les angles donnée par le modèle inverse sont faux.

Caractérisation des actionneurs

D'abord on a exprimer la relation mathématique reliant les valeurs des pas moteurs lus depuis l'actionneur en valeurs angulaires exprimées en radian. On introduit une valeur d'offset permettant de régler le 0 de la mesure. On en a déduit une fonction Matlab (step2angle) qui permet de convertir une valeur de pas moteur (variable step) en un angle q, exprimé en radian. Ensuite, On a inverser la relation obtenue précédemment pour obtenir l'expression reliant cette fois un angle exprimé en radian à la valeur de pas moteur correspondante et ainsi on en a déduit une fonction Matlab (angle2step) qui permet de convertir un angle q exprimé en radian en une valeur de pas moteur p pour un offset donné. Enfin on a tester ces fonctions et on a vérifier leur bon fonctionnement :

```
% Question 1
% r=5*pi/(3*1024);
% q=r*(step-offset);

%Question 3
%r=5*pi/(3*1024);
%p=(q/r)+ offset;

%Question 5
a=step2angle(512,0)
b=step2angle(1024,512)
c=angle2step(150,0)
d=angle2step(150,512)
```

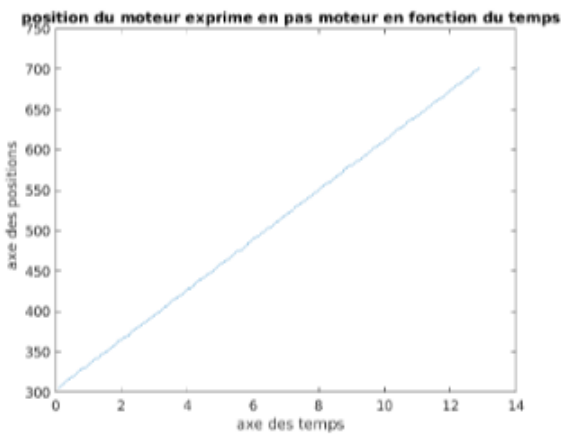
Test des fonctions

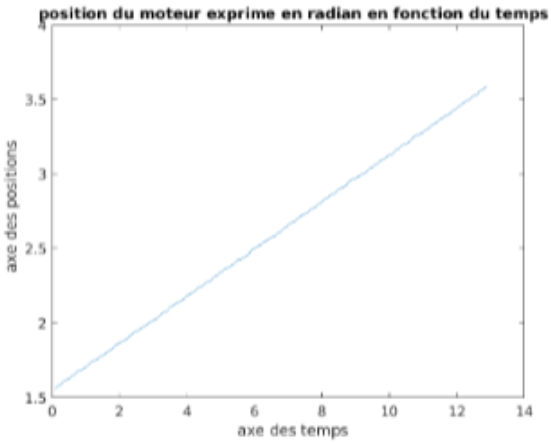
```
a =

    2.6180

b =

    2.6180

c =

   512

d =

        1024
```

Résultat Obtenu

On crée alors un robot 1R, Le robot simulé peut être commandé à l'aide de fonctions Matlab simples qui permettent de reproduire le mouvement du robot en fonction des paramètres de commande, en particulier la vitesse de déplacement et la position articulaire cible.

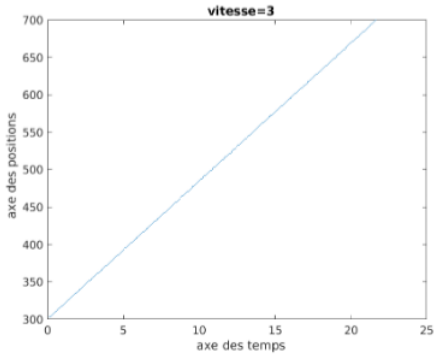
Pour un mouvement allant de la position moteur 300 à 700 avec une vitesse de 5 m/s,on a tracer l'évolution de la position du moteur en fonction du temps. Cette position sera exprimée tout d'abord en "pas moteur", puis en degrés ou radians.





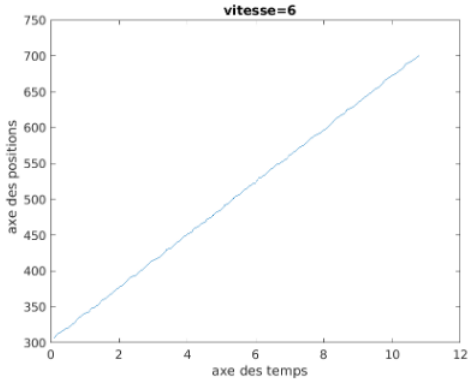
l'évolution de la position du moteur

On a effectuer ce tracé pour différentes valeurs de consignes de vitesse On a obtenue les résultats suivant :
Pour V=3 m/s :



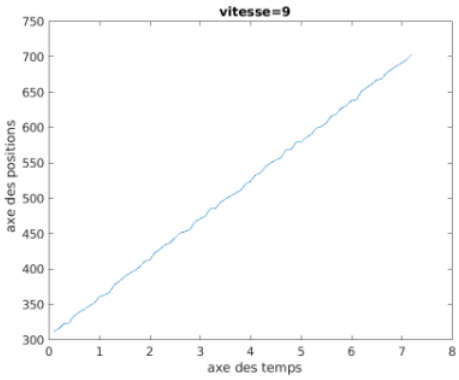
l'évolution de la position du moteur pour une vitesse = 3

Pour V=6m/s :



l'évolution de la position du moteur pour une vitesse = 6

Pour V=9 m/s :

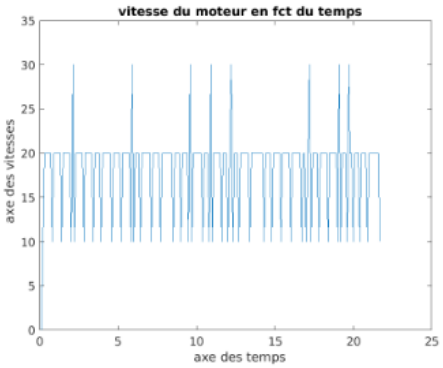


l'évolution de la position du moteur pour une vitesse = 9

On se basant sur les tracés en position obtenus pour différents réglages en vitesse,on a déterminer la vitesse angulaire effectivement obtenue la vitesse correspond à la pente de notre droite. exprimée en pas moteur par seconde, puis en tours par minute on a :

	Vitesse angulaire en métre par seconde	Vitesse angulaire en métre par minute
Pour V= 5 m/s	30,6982	1,7987
Pour V= 3 m/s	18,3797	1.0796
Pour V= 6 m/s	37,0094	2.1630
Pour V= 9 m/s	55,2851	3,2181

Les tracés de la position articulaire en fonction du temps permettent également d'estimer la vitesse de rotation du moteur par différence finie. on a tracer cette vitesse pour les différentes vitesse de déplacement et on a obtenue :



Vitesse de rotation du moteur en fonction du temps pour v= 3

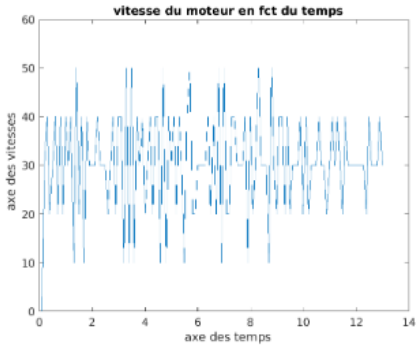


FIGURE 1 – Vitesse de rotation du moteur en fonction du temps pour v= 5

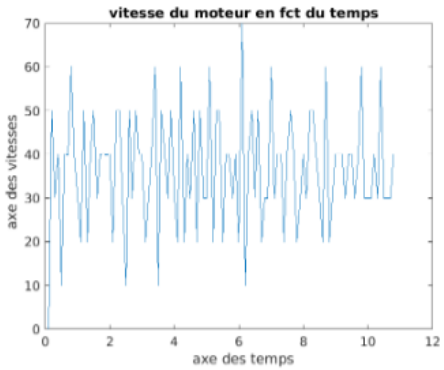


FIGURE 2 – Vitesse de rotation du moteur en fonction du temps pour v= 6

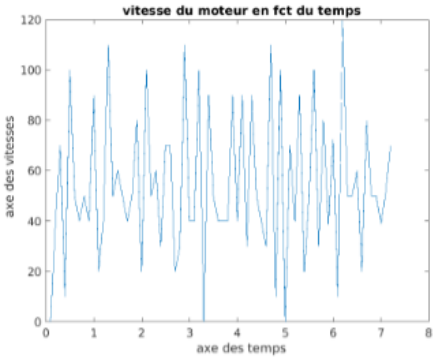


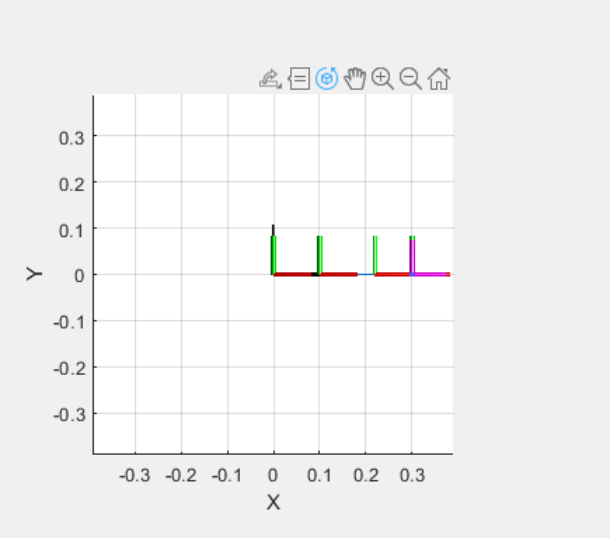
FIGURE 3 – Vitesse de rotation du moteur en fonction du temps pour v= 9

On remarque des perturbations on voit que les courbe tourne au tour de la valeur estimer dans la question précédente par exemple pour la vitesse= 5 on voit bien que les valeurs dans le graphes varient au tour de 30. on remarque aussi que plus la vitesse des moteurs augmente, plus l'oscillation de la vitesse augmente aussi, donc l'erreur est proportionnelle à la vitesse.

Robot 3R plan

Il s'agit maintenant de réaliser un robot 3R plan que nous allons commander à l'aide des fonctions spécifiques au robot simulé qu'on viens d'utiliser. mais également manipuler les modèles cinématiques direct et inverse d'un robot sériel simulé.

Pour commencé on va crée notre robot simulé 3R



le robot obtenue

A l'aide de la fonction modgeodir3Rplan on détermine la pose (x, y,) de l'organe terminal pour 3 commandes articulaire q1, q2 et q3 qu'on a choisi qi=[π/2,0,-π/2], exprimées en rad. on trouve la matrice de transformation homogène suivante :

```
T =

    1.0000         0         0    0.0393
         0    1.0000         0    0.1609
         0         0    1.0000         0
         0         0         0    1.0000
```

matrice de transformation homogène du pose d'organe terminale

On en conclue x = 0.0393 y= 0.1609 et $\theta = 0$

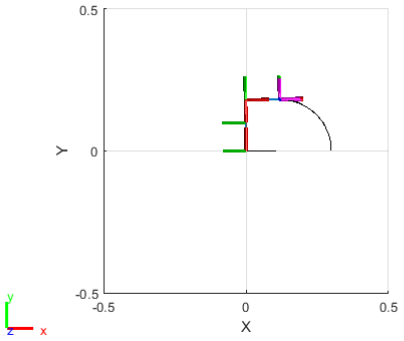
On applique ces commandes au robot simulé, pour différentes vitesse, et on vérifie que la pose atteinte corresponde bien à celle obtenue sur le robot simulé.

```
Theorie_v1 =

    0.7071   -0.7071         0    0.2121
    0.7071    0.7071         0    0.2121
         0         0    1.0000         0
         0         0         0    1.0000


Reel_v1 =

    0.7057   -0.7086         0    0.2117
    0.7086    0.7057         0    0.2126
         0         0    1.0000         0
         0         0         0    1.0000
```

Matrice de transformation théorique et réel pour v=3

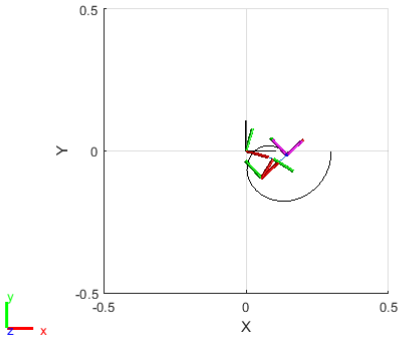
```
>> Theorie_v2 =

    0.7071   -0.7071         0    0.2121
    0.7071    0.7071         0    0.2121
         0         0    1.0000         0
         0         0         0    1.0000


Reel_v2 =

    0.7020   -0.7122         0    0.2106
    0.7122    0.7020         0    0.2136
         0         0    1.0000         0
         0         0         0    1.0000
```

Matrice de transformation théorique et réel pour v=6



Positionnement de notre robot

On remarque une éventuelle différence entre les valeurs théorique et réel, plus la vitesse des moteurs augmente, plus cette erreur augmente aussi on en conclue que la différence est une conséquence des perturbations observé dans la vitesse angulaire du moteur.



Positionnement de notre robot pour un offset=150

En partant d'une position initiale à une position quelconque, on voit bien qu'on arrive pas à maîtriser la trajectoire du robot dans l'espace opérationnel.

On s'est décider d'une pose (x, y,)= [[0.2, 0.2,π/6] cible dans l'espace opérationnel. A l'aide de la fonction modgeoinv3Rplan on a déterminer les ordres moteurs à appliquer pour atteindre cette pose.On exploite alors ces ordres moteurs pour commander le robot simulé :

```
%% Question 11
target=[0.2,0.2,pi/6];
[x,y] = mod_geo_inv_3Rplan(l, target)

x=angle2step(x,512)
R3.setPosition(x);
a=R3.getPosition()

y=angle2step(y,512)
R3.setPosition(y);
b=R3.getPosition()
```

Commande du robot simulé

On obtient avec la fonction modgeoinv3Rplan deux vecteurs d'angles qu'on convertit en pas moteur :

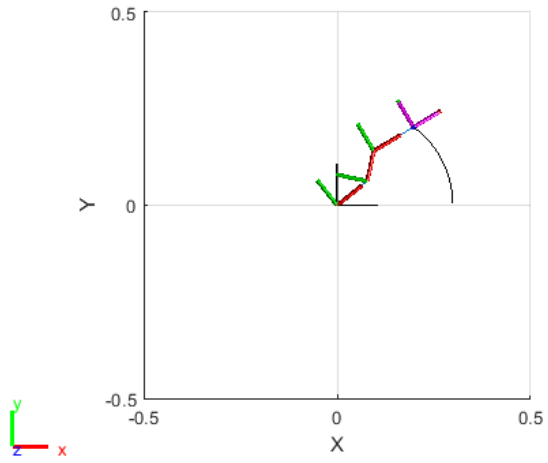
```
x =

  642.6760  645.1890  350.5350
```

```
a =

   644
   645
   351
```

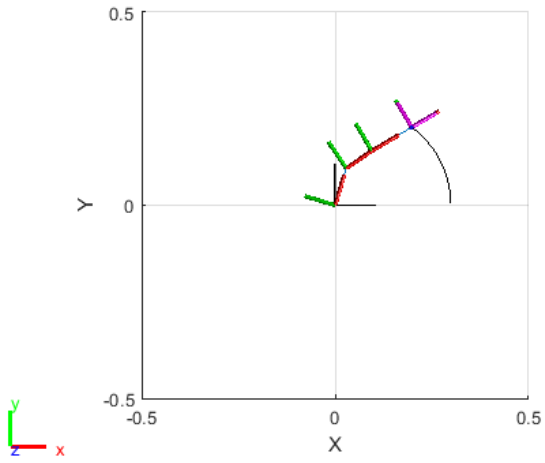
Position atteinte avec le premier vecteur d'angles x

```
b =

   761
   381
   500
```

Position atteinte avec le deuxième vecteur d'angles y



Positionnement du robot avec la solution x



Positionnement du robot avec la solution y

Si on compare la pose atteinte par l'organe terminal avec la consigne on remarque qu'il y a une légère différence entre les deux qui est du aux erreurs sur la vitesse moteur. Sur un robot réel, on limite traditionnellement

le mouvement de chaque moteur à un débattement angulaire suffisant pour la tâche à réaliser. On écrit alors une fonction limite :
Fonction limite :

```
function new_pos = limit(pos,val_min,val_max)
if (pos(1) <= val_min)
new_pos(1) =val_min;
elseif  (pos(1)>= val_max)
 new_pos(1) =val_max;
else
     new_pos(1)= pos(1);
end
if (pos(2) <= val_min)
new_pos(2) =val_min;

elseif  (pos(2)>= val_max)
 new_pos(2) =val_max;
else
     new_pos(2)= pos(2);
end
if (pos(3) <= val_min)
new_pos(3) =val_min;

elseif  (pos(3)>= val_max)
 new_pos(3) =val_max;
else
     new_pos(3)= pos(3);
end
end
```

On test le bon fonctionnement de notre fonction :

```
% %%Question 12==============
%tester la fonction limit
pos=[pi -3*pi/2  pi/4]
n_pos=limit(pos,-pi/2,pi/2)
```

Test fonction limite

```
pos =

    3.1416   -4.7124    0.7854


n_pos =

    1.5708   -1.5708    0.7854
```

Résultat du test fonction limite

Génération de trajectoires

On souhaite maintenant faire en sorte que le robot soit capable de suivre une trajectoire. Et du coup deux types de calcul de trajectoire sont envisageables :

Trajectoire dans l'espace opérationnel :

Soit une cible opérationnelle (x, y,) et une pose initiale de l'organe terminal. On peut alors déterminer par interpolation les pose intermédiaires menant en (x, y,), et en déduire les ordres moteurs qi à appliquer. Dans

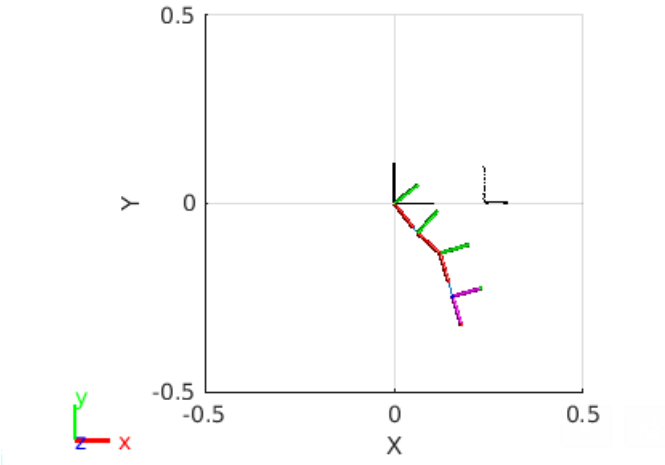
ce qui suit on vas commander notre robot dans l'espace opérationnel et on va voir comment on peut écrire des lettres avec cette méthode.

La structure de notre programme est comme suit : Afin d'écrire notre lettre on va choisir des segments. Ce qui veut dire que notre lettre contient plusieurs segments. Nous allons maintenant travailler sur un seul segment : Un segment représente une trajectoire précise => un target (un point cible) qu'on va approcher par interpolation (boucle). On obtient :

```
for i=0:0.01:0.1 %interpolation
  target1=[0.24,i,pi/4]; % definir target finale
[qa1,qb1]=mod_geo_inv_3Rplan(l,target1);
qc1=choose(qa1,qb1);
new_qc1=limit(qc1,-pi/2,pi/2);      %limiter les depacements angulaires
    Qc1= angle2step(new_qc1,512);    %conversion angle-pas
  R3.setPosition(Qc1,1);     %vers la position init

end
```

Boucle d'un seule segment

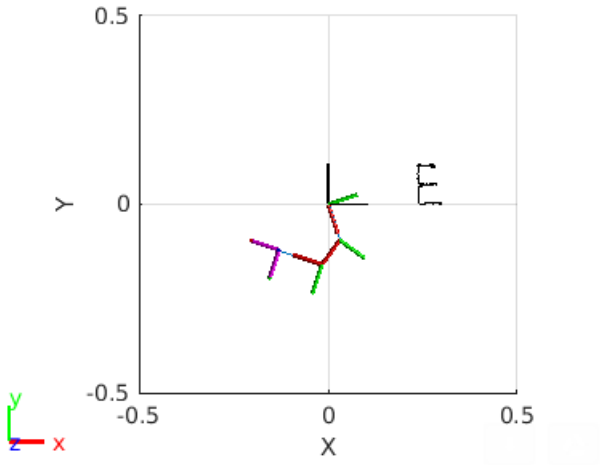
ici on a choisi d'écrire la lettre E notre premier segment est celui-ci :



Premier segment de la lettre E

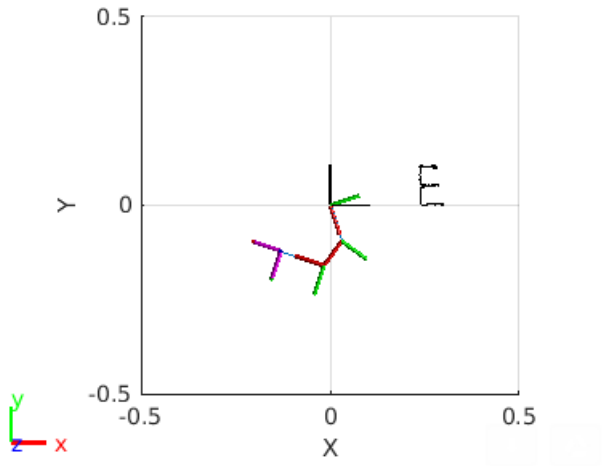
On peut voir clairement notre choix de target on a d'abord le x qui va vers 0.24, ensuite on a notre y qui va de 0 à 0.1 avec un angle de pi/4. Pour expliqué la boucle : on approche notre target avec i ensuite avec la fonction modgeoinv3Rplan on trouve nos ordres moteurs qi à appliquer on aura deux solutions à chaque fois, on en choisit une ensuit en la limite avec notre fonction limit pour ne pas avoir un souci après. Et enfin avec R3.setPosition on fait bouger notre robot avec qi. notre boucle se répète jusqu'à l'en arrive à notre target.

Maintenant on refait la même chose avec les autres segments afin d'écrire notre lettre. continuant avec la lettre E, la suite du programme est comme suit :



Le reste du programme pour la lettre E

En simulant on aura ça :



La lettre E

Trajectoire dans l'espace articulaire :

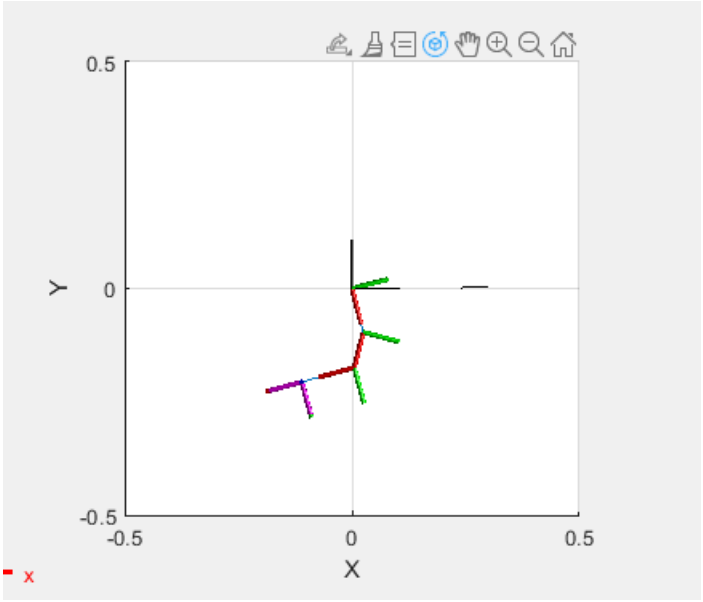
soit une cible opérationnelle (x, y,) et une position du robot caractérisée par sa position articulaire initiale. On peut alors déterminer les commandes motrices qi permettant d'atteindre la cible. Il s'agit ici de calculer par interpolation les commandes motrices intermédiaires.

Dans ce qui suit on vas commander notre robot dans l'espace articulaire et on va voir comment on peut écrire des lettres avec cette méthode. La structure de notre programme est comme suit : Afin d'écrire notre lettre on va choisir des segments comme dans la commande opérationnelle. Ce qui veut dire que notre lettre contient plusieurs segments. Nous allons maintenant travailler sur un seul segment : Un segment représente une trajectoire précise => un target (un point cible) qu'on va approcher par interpolation (boucle). On obtient :

```
Target=[0.24,0,pi/4] % definir target finale
[qa1,qb1]=mod_geo_inv_3Rplan(l,Target);
qc1=choose(qa1,qb1);
new_qc1=limit(qc1,-pi/2,pi/2);
for i=0.1:0.1:1 %interpolation
 new_qc1_int=i*new_qc1;
  Qc1= angle2step(new_qc1_int,512);    %conversion angle-pas
R3.setPosition(Qc1,1);     %vers la position init
end
```

Boucle d'un seule segment

ici on a choisi d'écrire la lettre L notre premier segment est celui-ci :



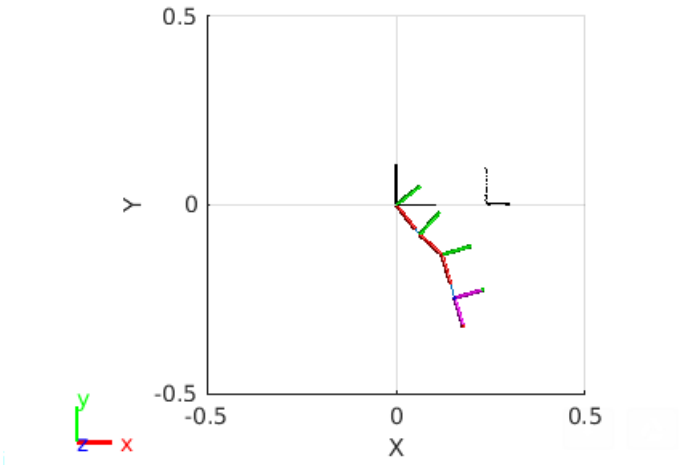
Premier segment de la lettre L

On peut voir clairement notre choix de target on a d'abord le x qui va vers 0.24, ensuite on a notre y qui reste à 0 avec un angle de pi/4. D'abord on prends notre target avec modgeodir3Rplan on trouve nos ordres moteurs qi0 initials en choisit une ensuit en la limite avec notre fonction limit pour ne pas avoir un souci après. On fait ensuit une boucle ou on interpole nos ordre moteurs pour aller de l'initial jusqu'a la position finale . on Et enfin avec R3.setPosition on fait bouger notre robot avec qi. Maintenant on refait la même chose avec les autres segments afin d'écrire notre lettre. continuant avec la lettre L, la suite du programme est comme suit :

```
Target2=[0.24,-0.1,-pi/4] % definir target finale
[qa2,qb2]=mod_geo_inv_3Rplan(1,Target2);
qc2=choose(qa2,qb2);
new_qc2=limit(qc2,-pi/2,pi/2);
for i=0.1:0.1:1 %interpolation
 new_qc2_int=i*new_qc2;
  Qc2= angle2step(new_qc2_int,512);    %conversion angle-pas
R3.setPosition(Qc2,1);     %vers la position init
end
```

Le reste du programme pour la lettre L

En simulant on aura ça :



la lettre L

Conclusion

les outils méthodologiques présentés constituent les bases pour la modélisation, la génération de mouvement et la commande des robots, bases nécessaires avant d'aborder l'étude des cinématiques plus complexes, notamment à chaînes fermées ou parallèles.