Compte Rendu

Projet HMEE 322

Modélisation et commande du robot Poppy

WANG Weili

18/12/2020

Sommaire

Introduction

L'objectif de ce projet est de réaliser la modélisation et la commande du bras gauche du robot Poppy sous Vrep, pour qu'il fasse un geste "Hello" comme suivant :

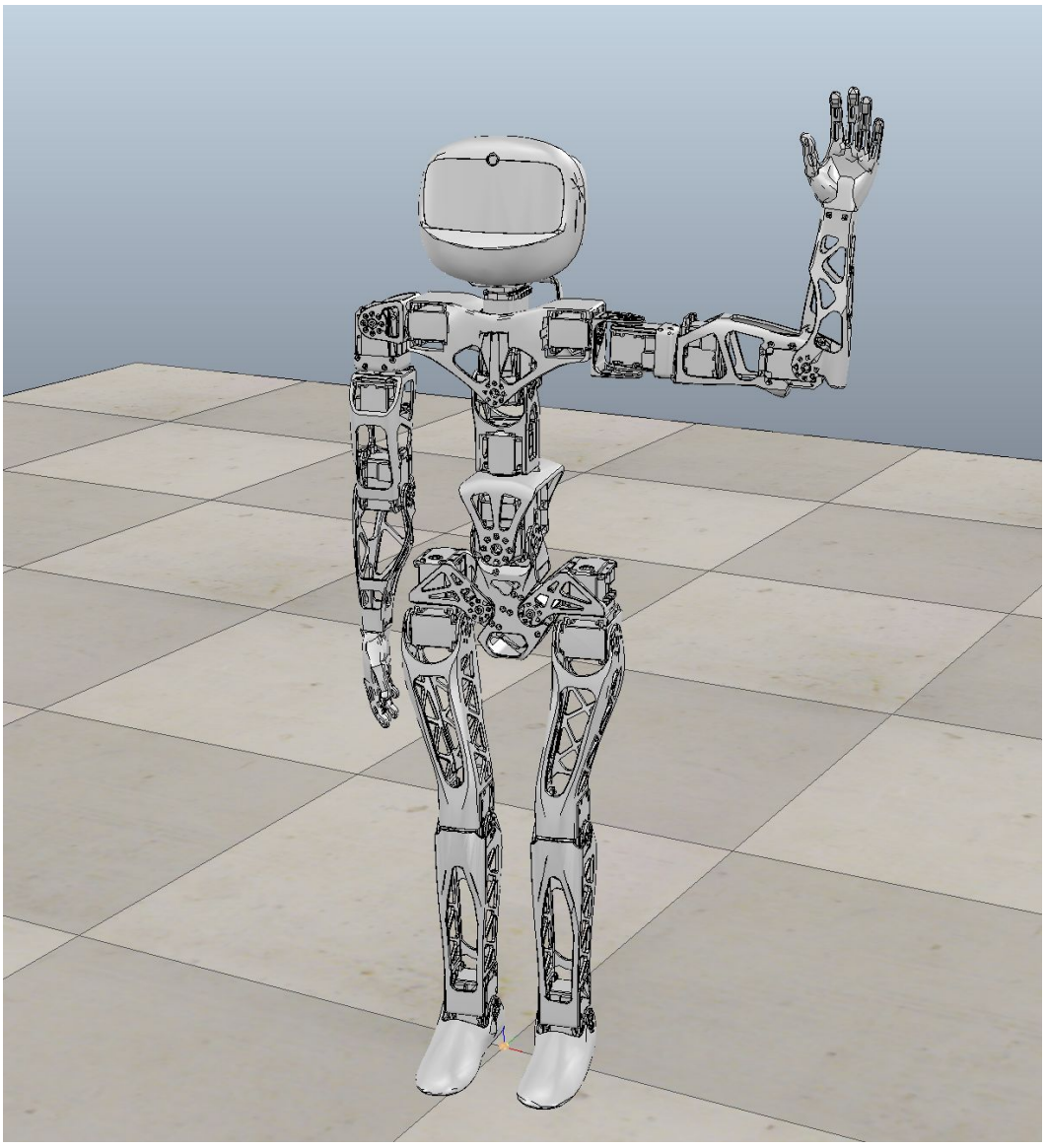

***Fig. 1** : Démonstration du geste à réaliser*

Les parties principales sont :

- Communication entre V-rep et Python
- Modélisation géométrique directe
- Command cinématique
- Génération de trajectoire

Communication entre V-rep et Python

Pour commencer, nous devons créer un lien entre notre simulateur et notre code de commande à l'aide de la librairie officielle de Poppy : **pypot**.

Nous allons d'abord créer un une entité du robot (en indiquant le simulateur utilisé) :

```
poppy = PoppyHumanoid(simulator='vrep')
```

Pour tester facilement, c'est mieux de fermer la connection de V-rep :

```
poppy.reset_simulation()
```

La librairie pypot nous offre de nombreux API pour contrôler notre Poppy. Par exemple pour les moteurs qui nous intéressent, les 4 moteur sont dans un groupe **l_arm** :

```
l_arm_group = poppy.l_arm
```

Et voici quelques fonctions utiles pour le projet suivant :

```
l_arm_group[i].present_position
```
pour retirer la position présente du moteur

```
poppy.nom_dumoteur.goto_position(angle_en_degre,1,wait=True)
```
tourner le moteur à une position désirée
Il existe une conversion radians -> degrés

Problem shooting :

Récupérer les positions des 4 moteurs du geste initial (rester au bout) en utilisant la fonction `l_arm_group[i].present_position`, le résultat donne :[-0.3, 0.3, 0, 0]. Mais utiliser la fonction `poppy.nom_dumoteur.goto_position` pour maintenir le même geste n'est pas possible.

En plus, d'après la documentation officielle (voir dans la partie de référence), pour la position initiale, tous les moteurs sont zéro.

C’est-à-dire la relation suivante n’est pas valable:

***Fig. 2** : Problem shooting*

Modélisation Géométrique Directe

Avec le tableau dans le document :

α	d	θ	r
0	0	θ_1	0
$\frac{-\pi}{2}$	0	θ_2	0.185
$\frac{\pi}{2}$	0	θ_3	-0.1480
$\frac{-\pi}{2}$	-0.01	θ_4	0
$\frac{-\pi}{2}$	0	θ_5	0.215

***Tab. 1** : Paramètres D-H du bras gauche du Poppy*

Et la formule générale :

$$ {}^{i-1}\mathbf{T}_i = \left[\begin{array}{ccc:c} \mathrm{c}\theta_i & -\mathrm{s}\theta_i & 0 & d_i \\ \mathrm{c}\alpha_i\,\mathrm{s}\theta_i & \mathrm{c}\alpha_i\,\mathrm{c}\theta_i & -\mathrm{s}\alpha_i & -r_i\,\mathrm{s}\alpha_i \\ \mathrm{s}\alpha_i\,\mathrm{s}\theta_i & \mathrm{s}\alpha_i\,\mathrm{c}\theta_i & \mathrm{c}\alpha_i & r_i\,\mathrm{c}\alpha_i \\ \hdashline 0 & 0 & 0 & 1 \end{array}\right] $$

***Eq. 1** : Matrice de transformation entre deux articulations*

Nous pouvons facilement enregistrer les 5 matrices de transformations dans une **liste** de Python, et puis **parcourir** cette liste pour la **multiplication accumulée** afin d'avoir les matrices $_0\mathbf{T}_1$, $_0\mathbf{T}_2$, $_0\mathbf{T}_3$, $_0\mathbf{T}_4$, $_0\mathbf{T}_5$. Notamment, le bras gauche de notre Poppy consiste à 4 moteurs, et la dernière matrice de transformation représente la transformation entre la dernière articulation (coude gauche) et l'organe terminal. Sachant que la main de Poppy et son coude sont une pièce entière d'impression en 3D, θ_5 est donc toujours 0, et r_5 = 0.215 m est la longueur de cette pièce entière.

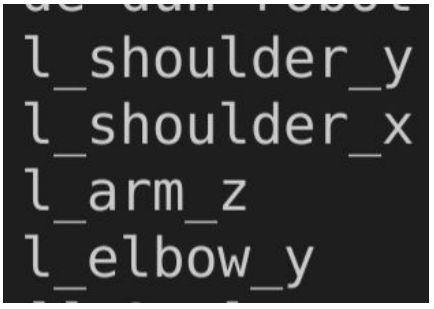

***Fig. 3** : Les 4 moteurs à commander (sortie du programme)*

En vérifiant avec mon programme, le tableau D-H est faux.

Command Cinématique

Pour cette partie, nous devons utiliser les 5 matrices de transformation sauvegardées dans la liste créée dans la partie MGD pour trouver la matrice **Jacobienne**.

$$J = \begin{bmatrix} I & D \\ O & C \end{bmatrix} \cdot {}^{0}J_{5}$$

***Eq. 2** : Matrice Jacobienne*

A l'aide de la fonction linalg de **Numpy**, nous sommes capable d'obtenire la **pseudo-inverse** de la matrice jacobienne : J^{+}.

Vu que les résultats précédents sont faux, la matrice jacobienne est fausse aussi.

Mais il y a autre chose à faire attention, différent que le TP que nous avons fait, pour ce projet, le dernier repère se pose directement sur l'organe terminal, et il n'y a pas de rotation dans ce repère.

Une fois nous avons le bon résultat, nous pouvons donc commander les vitesses des moteur grâce à la fonction :

```
poppy.nom_dumoteur.goto_position(angle_en_degre,1,wait=True)
```

d'où son deuxième paramètre c'est le temps (en seconde) pour atteindre la position commandée.

Génération de Trajectoire

Vu que le tableau DH donné est faux, cette partie est donc la seule partie que nous pouvons réaliser la demande, soit le geste “Hello”. Nous allons d’abord trouver à la main les 4 positions des moteurs pour ce geste. Il sont :

l_shoulder_y = -1.5
l_shoulder_x = 1.5
l_arm_z = 0
l_elbow = -1.5

Et la position initiale :

l_shoulder_y = 0
l_shoulder_x = 0
l_arm_z = 0
l_elbow = 0

Nous sommes proposé à utiliser le **polynôme de degré 3** pour la génération de trajectoire, mais pour évider le problème qui peut être causé par la **discontinuité d’accélération** sur notre Poppy (même si c’est pas la peine de faire attention dans le simulator V-rep), j’ai choisi le **polynôme de degré 5** :

$$q_f = a_0 + a_1 t_f + a_2 t_f^2 + a_3 t_f^3 + a_4 t_f^4 + a_5 t_f^5$$

***Eq. 3** : Polynôme de degré 5*

d’où:
q_f est le geste “Hello” et q_0 et le geste initial

La formule s'écrit en **temps continu**, pour la **discrétisation** :

```
a3 = 10*(posHello-posInit)/(simulationTime**3)
a4 = -15*(posHello-posInit)/(simulationTime**4)
a5 = 6*(posHello-posInit)/(simulationTime**5)
newPos = posInit + a3*deltaT**3 + a4*deltaT**4 + a5*deltaT**5 # be added for each setp
```

***Fig. 4** : Polynôme de degré 5 discret*

Et la boucle de simulation est :

```
posPresent = np.mat('[0; 0; 0.0; 0; 0.0]')
posHello = np.mat('[-1.5; 1.5; 0.0; -1.5; 0.0]')
simulationTime = 20
deltaT= 0.5
timeseries = np.linspace(0, simulationTime, num = int(simulationTime//deltaT))

for t in timeseries:
    newPos = leftArm.gen_traj(posHello, simulationTime, t)
    poppy.l_shoulder_y.goto_position(newPos[0,0]*180/PI, 0.05, wait=False)
    poppy.l_shoulder_x.goto_position(newPos[1,0]*180/PI, 0.05, wait=False)
    poppy.l_arm_z.goto_position(newPos[2,0]*180/PI, 0.05, wait=False)
    poppy.l_elbow_y.goto_position(newPos[3,0]*180/PI, 0.05, wait=True)
    posPresent = newPos
    print(newPos)
poppy.stop_simulation()
```

***Fig. 5** : Boucle de simulation*

Poppy atteint la position désirée (geste “Hello”), et nous pouvons constatons les résultats de la génération de trajectoire de la position initiale convergent à la position désirée comme suivant :

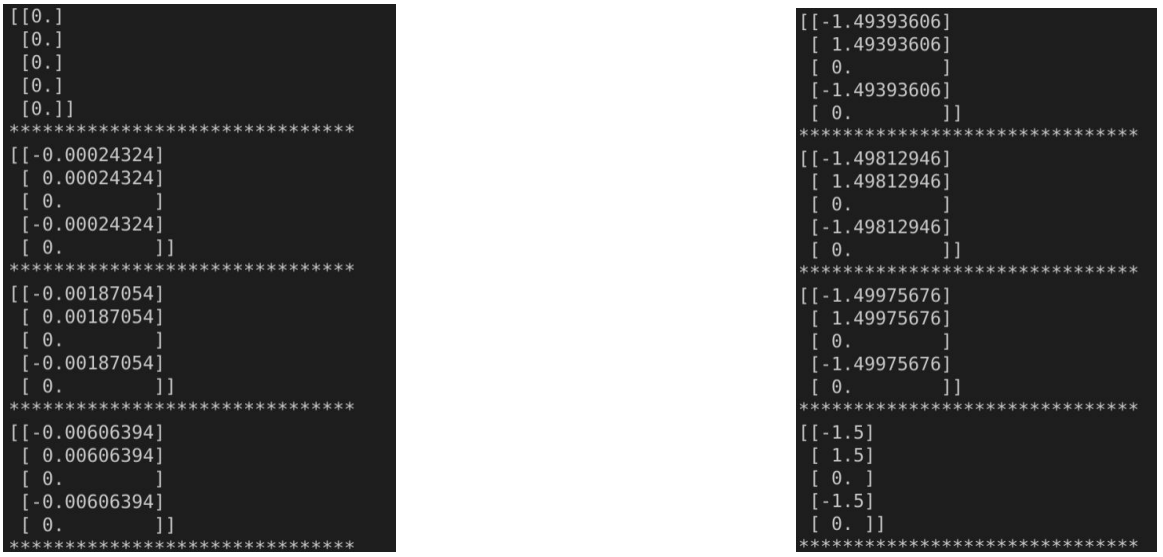

***Fig. 6** : Résultat de la génération de trajectoire*

J’ai aussi testé le **polynôme de degré 3**, nous pouvons constater que le mouvement généré par **le polynôme de degré 5** est plus lissé et continuel (voir la vidéo ci-jointe).

Conclusion

Malheureusement, à cause de la faute du tableau DH au début, je n’ai pas pu vérifier la partie de modélisation géométrique directe et la partie de commande cinématique, même si j’ai déjà réalisé les fonctions. Mais ce projet m’a beaucoup bénéficié:

- Il m'aide à réviser le cours et approfondir la compréhension en réalisant les théories dans livre.
- Il renforce ma compétence de programmation en Python.
- Suprisement, il m’a beaucoup inspiré sur l’autre projet (modélisation et commande de UR10 avec Ros).

D'ailleurs, la compétence de débugger est aussi importante.

Référence

Pypot :
https://github.com/poppy-project/pypot

https://poppy-project.github.io/pypot/index.html

Numpy :
https://numpy.org/

Poppy Humanoid :
https://docs.poppy-project.org/fr/getting-started/

https://github.com/poppy-project/poppy-humanoid

https://github.com/Phylliade/ikpy

https://github.com/poppy-project/community-notebooks/blob/master/tutorials-education/poppy-humanoid_poppy-torso__vrep_installation%20et%20prise%20en%20main/poppy%20simul%C3%A9/premier%20pas%20avec%20poppy%20humanoid%20en%20python%20-%2010%20choses%20%C3%A0%20savoir.ipynb