TD : contrôle d'une patte à 4 degrés de liberté

Introduction

Ce TD contient deux fichiers : `simulation.py` qui est une interface permettant d'exécuter différentes parties du code facilement et `control.py` dans lequel vous allez écrire la majeure partie de votre code.

Dans ce TD, l'objectif est de contrôler la patte d'un robot à 4 degrés de libertés en s'assurant que l'extrémité de la patte reste parallèle au sol. Pour ne pas avoir à gérer les contacts avec le sol, la patte est fixée au sommet du cylindre noir.

Vous pouvez observer la structure du robot et changer la position des différents degrés de liberté en lançant la commande suivante :

`./simulation.py --joint-space`

Comme dans le TD précédent, vous pouvez choisir comment est obtenue la consigne dans l'espace articulaire en vous basant sur les modes suivants :

- `--joint-space` : La cible est donnée dans l'espace articulaire du robot
- `--jacobian-inverse` : La cible est donnée dans l'espace opérationnel, la méthode utilisant l'inverse de la jacobienne est utilisée pour déterminer quelles positions utiliser.
- `--jacobian-transposed` : La cible est donnée dans l'espace opérationnel, la méthode utilisant la transposée de la jacobienne est utilisée pour déterminer quelles positions utiliser.
- `--analytical-mgi` : La cible est donnée dans l'espace opérationnel, les positions angulaires à utiliser sont calculées à partir d'un modèle analytique.

Comme ce robot possède 4 degrés de libertés, il est possible d'ajouter une contrainte supplémentaire. Dans notre cas, l'objectif est de garder le plan au bout de la patte aligné avec le sol. Pour ce faire, on va imposer une contrainte sur $r_{3,2}$ l'un des éléments de la matrice ${}^0T_E$:

$$ {}^0T_E = \begin{pmatrix} r_{1,1} & r_{1,2} & r_{1,3} & X \\ r_{2,1} & r_{2,2} & r_{2,3} & Y \\ r_{3,1} & r_{3,2} & r_{3,3} & Z \\ 0 & 0 & 0 & 1 \end{pmatrix} $$

Effectivement, lorsque $r_{3,2} = -1$, cela signifie que la plan est orienté comme souhaité.

L'avantage de cette approche est qu'elle permet de définir facilement la position du robot dans l'espace opérationel o en se basant sur ${}^0T_E$: $o = (X, Y, Z, r_{3,2})$. Comme il est facile de dériver la matrice ${}^0T_E$ par rapport à chaque degré de liberté, cela facilite l'obtention de la jacobienne.

Note : Dans ce TD, plusieurs questions sont posées, assurez-vous d'y répondre dans un fichier `README` que vous ajouterez à l'archive envoyée.

Implémentation du modèle

Les différents éléments de l'introduction et le fichier `resources/leg.urdf` vous permettent d'implémenter les différentes méthodes de la classe `LegModel` :

— `computeMGD` qui pourra être testé en vérifiant que la position de l'effecteur est bien au centre du plan à l'extrémité de la patte.
— `computeJacobian` en dérivant la matrice de transformation et en extrayant les éléments pertinents.
— `computeAnalyticalMGI` en approfondissant ce que vous avez fait jusqu'à présent. Attention, cette méthode n'est pas bloquante pour avancer dans le TP, n'hésitez donc pas à y revenir plus tard si vous bloquer. Comme l'objet a contrôler est une patte, on demandera dans ce cas de choisir parmi les solutions possibles sur la base de la valeur de q_1 :
  1. Si possible, q_1 doit être positif.
  2. Si plusieurs solutions respectent le premier critère, il faut que $\|q_1\| < \frac{\pi}{2}$ si possible.
  3. Si ces deux critères ne suffisent pas à retenir une unique solutions, indiquez dans quels cas elles ne suffisent pas et choisissez librement parmi celles qui restent.

En important les méthodes `searchJacInv` et `searchJacTransposed` depuis le TP précédent, vous pourrez tester qu'elles s'adaptent bien à un autre robot.

Vous pouvez tester toutes les différentes méthodes d'obtenir le MGI en modifiant la cible dans l'espace opérationnel manuellement.

Suivi de trajectoire

Le module `control.py` permet de lire des splines linéaires et cubiques correspondant à des consignes dans l'espace opérationnel.

Pour les utiliser, il faut ajouter l'option suivante : `--trajectory-type XXX`, où `XXX` est : `linear_spline` ou `cubic_spline`. Il est également nécessaire de spécifier le fichier à utiliser avec l'option `--trajectory-path PATH`. Finalement, l'option `--cyclic` permet de répéter la spline à l'infini.

Deux exemples de trajectoires sont fournis avec le TD : `splines/linear_step.json` et `splines/cubic_step.json`. Essayez les avec les différents modes de MGI que vous avez implémentés et affichez les trajectoires obtenues à l'aide du script `plot_trajectories.r`.

Quelle trajectoire vous semble la plus adaptée pour faire marcher un robot quadrupède et pourquoi ?

Pour aller plus loin

Pour effectuez les tâches suivantes, il sera potentiellement nécessaire de modifier le fichier `simulation.py`. Dupliquez donc votre dossier afin de conserver la version originale avant d'aller plus loin.

Les tâches à accomplir sont les suivantes (il n'est pas nécessaire de toutes les faire) :
— Ajoutez de nouvelles trajectoires, soit sous forme de spline, soit en parcourant des formes géométriques pures avec l'extrémité de la patte.
— Lorsque votre cible n'est pas atteignable et que vous utilisez la méthode de la jacobienne transposée, vous cherchez à minimiser une fonction de coût. Un des problèmes est que votre fonction de coût mélange des erreurs en position et des erreurs d'une autre nature $r_{3,2}$. Implémentez deux modes, l'un qui privilégie le fait de respecter les contraintes sur la position de l'outil et l'autre qui privilégie le fait de respecter la contrainte sur $r_{3,2}$.
— En vous basant sur le fait que $J(q)\dot{q} = \dot{o}$ et que les vitesses dans l'espace opérationnel sont fournies par les classes `LinearSplinePose` et `CubicSplinePose`, calculez $\dot{q}$. Ajoutez $\dot{o}$ et $\dot{q}$ aux valeurs écrites dans le fichier `data.csv`. Créer une trajectoire qui commencent par une position atteignable et s'éloigne hors de celui-ci. Observez les vitesses demandées au niveau des articulations et commentez.

Évaluation

Les documents seront à déposer sur Moodle dans une archive nommée : `nom_prenom.tar.gz` L'exécution de `tar -xzf nom_prenom.tar.gz` ne devra pas afficher d'erreur.

L'archive devra contenir un dossier `nom_prenom` dans lequel vous mettrez :

— Votre fichier `control.py`

— D'autres fichiers python si votre `control.py` si besoin

— Un fichier `README` récapitulant le travail effectué et les différentes réponses aux questions posées.

— Si vous avez avancé dans la partie 'pour aller plus loin', ajoutez aussi un dossier nommé `extension` contenant le code modifié.