TD: Modèle géométrique inverse

1 Introduction

Ce TD prolonge le précédent sur le MGD, il va permettre d'explorer différentes méthodes pour obtenir le modèle géométrique inverse d'un robot. Peu importe votre niveau d'avancement, il est requis que vous commenciez votre travail à partir de l'archive disponible sur la page du cours.

Quelques fonctionnalités supplémentaires ont été ajoutées au `motor_controller.py` sous forme de variables d'environnement:

- `CONTROL_MODE` permet de choisir le mode de contrôle utilisé parmi les suivants

  `computeMGD`: Le mode utilisé lors du TP précédent, la cible est précisée dans l'espace articulaire du robot.

  `analyticalMGI`: La cible est précisée dans l'espace opérationnel du robot. Le MGI est obtenu à l'aide d'une résolution analytique.

  `jacobianInverse`: La cible est précisée dans l'espace opérationnel du robot. Le MGI est obtenu à l'aide de l'inverse de la jacobienne.

  `jacobianTransposed`: La cible est précisée dans l'espace opérationnel du robot. Le MGI est obtenu à l'aide de la jacobienne transposée.

Comme les méthodes de MGI ne sont pas implémentées dans le code qui vous est fourni, vous pouvez lancer le code avec les différents `modes` mentionnés ci-dessus et changer la cible, mais le robot ne bougera pas.

Finalement, un robot de plus a été ajouté, celui-ci permettra d'approfondir les spécificités possibles de l'espace opérationnel et de mettre en pratique l'ensemble des connaissances acquises depuis le début du TP.

Pour tous vos tests de MGI, pensez bien à vérifier non seulement les cas simples, mais aussi ceux qui risquent d'être problématiques:

- Des cibles éloignées de la position initiale.
- Des cibles en dehors de l'espace atteignable.
- Des cibles qui peuvent être atteintes avec plusieurs configurations.

Le rendu devra également inclure un compte-rendu qui devra au moins inclure les réponses à toutes les questions explicitement mentionnée dans cette feuille de TD.

2 Préparation des outils nécessaires

Plutôt que de vous lancer directement dans l'implémentation des différentes méthodes permettant d'obtenir un MGI, il est important de préparer quelques outils pour pouvoir déboguer plus rapidement votre code.

2.1 Obtention de la Jacobienne

Implémentez la méthode `computeJacobian` pour les robots `RobotRT` et `RobotRRR`, pensez à tester votre code dans des configurations simples. Une implémentation correcte de cette méthode sera nécessaire pour les deux méthodes itératives.

2.2 Affiner les limites de l'espace opérationnel

Les méthodes `getOperationalDimensionLimits` des robots `RobotRT` et `RobotRRR` contiennent pour l'instant une implémentation basique qui ne tient pas compte des capacités réelle du robot. Modifiez ces valeurs afin d'obtenir l'hyper-rectangle le plus petit possible qui n'exclut aucune position atteignable.

2.3 Enrichir votre fichier de logs

Afin de pouvoir visualiser plus facilement comparer la position obtenue par votre robot et sa cible dans l'espace opérationnel, ajoutez aux données enregistrées dans le fichier de log la cible donnée à votre robot dans l'espace opérationnel ainsi que la cible dans l'espace articulaire (celle calculée par le MGI).

Comme le temps de calcul est une variable critique pour comparer les différentes méthodes, ajoutez aux éléments qui sont exportés le temps nécessaire pour mettre à jour le modèle physique du robot.

3 Méthodes itératives

3.1 Inverse de la jacobienne

En vous basant sur les éléments présentés en cours, implémentez la méthode `solveJacInverse` de la classe `RobotModel`.

3.2 Jacobienne transposée

En vous basant sur les éléments présentés en cours, implémentez la méthode `solveJacTransposed` de la classe `RobotModel`. Pour cette méthode, vous pourrez utiliser la fonction `scipy.optimize.minimize`. Pensez bien à comparer les résultats obtenus en spécifiant la jacobienne ou sans la spécifier.

4 MGI analytique

Implémentez les méthodes `analyticalMGI` pour les robots `RT` et `RRR`, ces méthodes doivent permettre non seulement d'identifier une solution, mais aussi de fournir le nombre de solutions existantes.

5 Un peu de recul

Vous avez à présent implémenté 3 méthodes différentes sur 2 robots relativement simples. À travers les différents tests que vous avez pu mener durant l'implémentation, vous avez sûrement fait face à différents problèmes ou simplement aux limitations des méthodes mentionnées.

Prenez un peu de temps pour documenter ces éléments et restituez les à travers des explications et des graphiques mettant en avant les éléments que vous avez pour répondre aux questions suivantes:

- Est-ce que les limitations correspondent à celles mentionnées en cours?
- Est-ce qu'une des méthodes semble supérieure à toutes les autres et pourquoi?
- Quel est votre avis personnel sur ces 3 méthodes?

Rassemblez vos réponses à ces questions dans votre compte-rendu.

6 MGD et MGI d'une patte à 4 degrés de liberté

Le robot `LegRobot` représente une patte de robot à 4 degrés de liberté, un des avantages de cette structure est qu'elle permet d'assurer de non seulement contrôler la position de l'extremité de la patte, mais aussi d'assurer que celle-ci soit toujours orientée en direction du sol.

Pour obtenir ce comportement, on va considérer que l'espace opérationnel est composé non seulement de la position (x, y, z) de l'outil, mais aussi de $r_{3,2}$ un des éléments de la matrice de transformation ${}^0T_E$, voir (1). On définit donc $\mathcal{G}_{\text{Leg}}$ le MGD de la jambe avec (2). Attention, ce choix ne permet pas de contrôler l'orientation de la patte de manière globale, mais plutôt la composante en z dans le repère 0 du vecteur unitaire $\vec{y}$ dans le repère E, voir (3).

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

$$ \mathcal{G}_{\text{Leg}}({}^0T_E) = \begin{pmatrix} X \\ Y \\ Z \\ r_{3,2} \end{pmatrix} \tag{2} $$

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

Implémentez toutes les méthodes de cette classe et testez différentes cibles. Commentez les résultats obtenus dans votre compte-rendu.

7 Consignes de rendu

1. Éditez le contenu du fichier `to_pack.txt` pour qu'il contienne tous les fichiers demandés
2. Créer une archive en utilisant la commande suivante:

   ```
   python3 -m seauval.manager pack --group <group_id> .
   ```

3. Vérifiez le contenu de l'archive `tar tzf XXX.tar.gz`
4. Envoyer votre archive par mail avec le sujet suivant: "(4TSD904U) Rendu TD MGI". N'oubliez pas de mettre en copie tous les membres du groupe.