université de BORDEAUX

Robotique, Mécatronique, Infotronique

TD: Modèle géométrique inverse - Rapport

Titien Cubilier
Florian Gardes
Abdelamine Mehdaoui
Luka Moraiz

23 Octobre 2020

Table des matières

1 Préparation des outils nécessaires : 3

2 Méthodes itératives : 4
2.1 Jacobienne transposée : 4

3 MGI analytique : 5
3.1 RT : 5
3.2 RRR : 5

4 Un peu de recul : 6
4.1 Est-ce que les limitations correspondent à celles mentionnées en cours ? 6
4.2 Est-ce qu'une des méthodes semble supérieure à toutes les autres et pourquoi ? 6
4.3 Quel est votre avis personnel sur ces 3 méthodes ? 6

5 MGD et MGI d'une patte à 4 degrés de liberté : 7

1 Préparation des outils nécessaires :

L'objectif derrière la première partie du log est de pouvoir visualiser le niveau de précision de nos méthodes pour le calcule de MGI(comme c'était déjà fait pour le MGD).
En ce qui concerne la 2ème partie, on renvoie juste le temps de calcul de chaque méthode pour pouvoir les comparer à la fin.

2 Méthodes itératives :

2.1 Jacobienne transposée :

Fonction scipy.optimize.minimze : le premier argument est la distance, ensuite joints, puis (self, target), puis on prends la formule du pdf concernant la transposé pour le dernier argument "jac".

Pour la Transposée de RT et seulement pour des valeur négatif que x nous avons un problème, une erreur de 0,07 et nous avons aucune idée pourquoi.

Malheureusement nous avons beaucoup perdu de temps sur ça ce qui fait que notre parti analytique a été commencé trop tard.

3 MGI analytique :

3.1 RT :

Nous devons déterminer deux joints, le premier appelons le teta1 se calcul par le biais d'un "artan(target)" Tandis que le second joint nous le déterminons en calculant la distance à partir du points d'origine en se basant sur la target et les longueurs de bras.

3.2 RRR :

Malheureusement il ne fonctionne pas. Cependant l'idée est d'avoir : .
phi = teta1 + teta2 + teta3
phi étant l'angle du vecteur L3 par rapport à l'axe x.
Commencons par teta2, le calcul étant arccos((Dist_joint[2]^2 - L1^2 - L2^2) / 2*L1*L2)
Pour teta 1, nous devons déterminer alpha étant Atan(Dist_joint[2])
Puis beta = arccos((Distance^2 +L1^2 -L2^2)/(2*L1*racine(Distance^2)))
teta1 sera la somme de alpha +/- beta suivant la position du joint[1]
Enfin pour teta3, c'est simplement la soustraction de phi par rapport à teta1 et 2.

4 Un peu de recul :

4.1 Est-ce que les limitations correspondent à celles mentionnées en cours ?

Oui, les limitations correspondent à celles mentionnées en cours.

notamment sur les méthodes analytiques dont on trouve pas une méthode générale pour tous les robots contrairement au méthodes algébriques, d'où la difficulté.

4.2 Est-ce qu'une des méthodes semble supérieure à toutes les autres et pourquoi ?

Selon nous la méthode Analytique devrait être la meilleure cependant elle est plus dur a mettre en place. Nous avons juste réussi à mettre en place cette méthode pour le robot RT donc on n'a pas pu beaucoup la testé. On a juste remarqué que pour RT analytique est plus rapide que les méthodes itératives.

La méthode inverse est plus lente mais plus simple à mettre en œuvre. Malheureusement elle a quelques limites que l'on a pu voir dans le robot leg

Pour finir la transposée est elle aussi simple à mettre en œuvre mais a les mêmes défauts que la méthode inverse. Nous avons eu des difficultés à utilise la fonction "minimize".

4.3 Quel est votre avis personnel sur ces 3 méthodes ?

La solution analytique a pour avantage de diminuer considérablement le nombre d'opération, mais on doit traiter tous les cas singulier, elle est plutôt efficace

Les deux méthodes numériques sont plus générales, la plus répandue est la Jacobienne Inverse, les algorithmes traitent de façon unifiée les cas réguliers, singuliers et redondants, elles nécessitent un temps de calcul relativement important

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

Pour le robot « leg » nous avons rapidement fait l'expérience des limites de la méthode itérative.

Par exemple, pour la fonction jacobienneInverse, la jacobienne est composée de 4 dérivées (de taille 3*4) nous ne pouvions donc plus inverser la matrice.

Pour régler ce problème nous avons ajouté une ligne à la matrice composée de R3,2 et ajouté un « 0 » au vecteur de distance afin qu'il soit lui aussi de dimension 4 grâce au MGD.