CENTRALE NANTES

Project 2:
Design of a parallel Schönflies motion generator

Abdellah KOUTIT
Arthur ASTIER

Mars 2023

Présentation du Problème:

Le deuxième sujet nous a beaucoup attiré car les robots de pick and place font partie intégrante du paysage industriel mondial depuis les années 1930, avec l'invention par Bill Taylor de la première grue robotisée.
S'en sont suivis des modèles de plus en plus développés, avec une architecture sérielle comme le robot SCARA (Selective Compliance Assembly Robot Arm) qui est capable d'exercer un mouvement de Schönflies. Ce mouvement réside dans la capacité du robot à se déplacer linéairement dans l'espace et avec une seule rotation possible autour d'un axe fixe au cours du temps.
Il est donc idéal pour une opération de pick and place qui fait transiter un objet entre deux plans parallèles avec possiblement une orientation différente entre le premier et le second plan.
Cependant, dans certains cas la robotique sérielle n'est pas la plus performante. En effet, pour des applications rapides avec des objets légers on va préférer une architecture parallèle, qui permet d'avoir un déplacement des masses mieux réparti. Dans notre cas, le mouvement de Schönflies nous impose un mécanisme à 4 degrés de liberté PPPR. Ainsi, en regardant le livre de Kong Gosselin aux pages 155-156 on peut voir que des mécanismes parallèles avec quatre jambes identiques nous sont proposés. Nous avons choisi de partir sur un modèle composé uniquement de liaisons rotoïdes, dont les axes de rotation ne sont pas inclinés.
Le modèle choisi est le modèle (b) $\underline{R}''R''R''R'R'$.

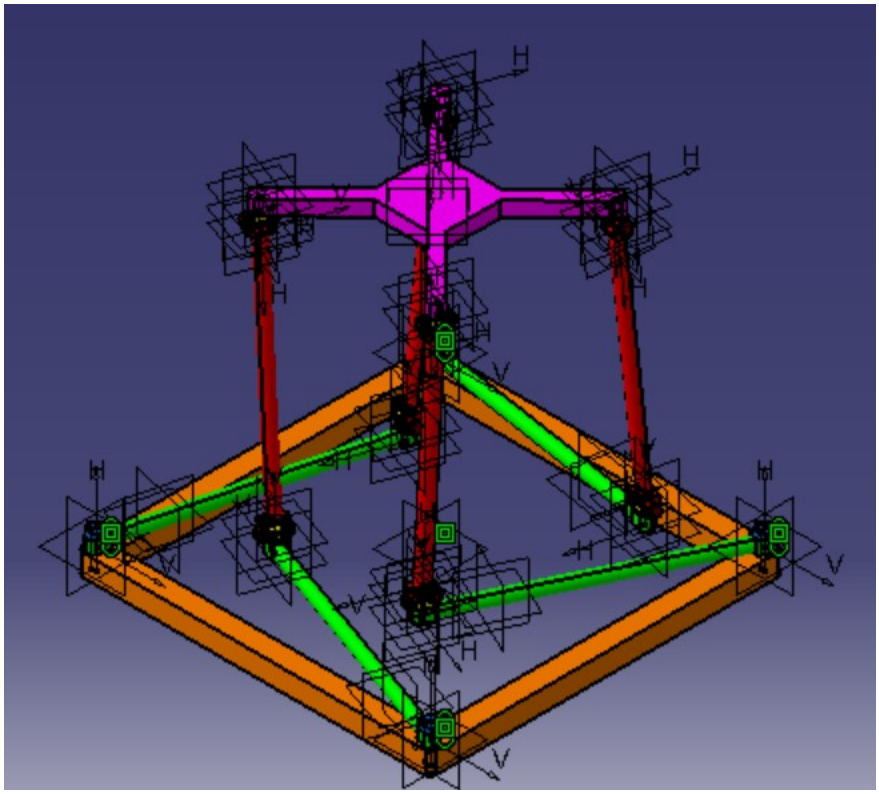

Figure 1: $\underline{R''}R''R''R'R'$.

1 Problème de conception :

1.1 Hypothèses:

En première approche, on souhaite simplifier un peu le problème. Nous allons donc faire un certain nombre d'hypothèses, réduisant ainsi la quantité de paramètres.

- Nous faisons l'hypothèse que les deux bras de chaque jambe seront de même section et de même matériau (d le diamètre des bras) .
- La base et la plate-forme mobile sont construites par des tubes à section carré d'épaisseur $e = sqrt(pi).d/2$: ce qui va nous permettre de les considérer comme des tubes à section circulaire de diamètre d.
- les paramètres (ρ et d) du matériau ne seront pas à optimiser ce qui est normal (les contraintes d'optimisation n'en dépendent pas)
- la position et la direction de la trajectoire ne sera pas à optimiser et puisque le système représente une symétrie par rapport à l'axe z donc on a choisi que la trajectoire sera dans le plan (x = 0)

1.2 Variables d'optimisation

x=[x(1) x(2) x(3) x(4)]

- x(1) = l_1 (longueur du premier bras)
- x(2) = l_2 (longueur du deuxième bras)
- x(3) = a (coté du carré (plateforme mobile))
- x(4) = b (coté du carré (Base))

1.3 Fonction objectif obj.m:

On souhaite minimiser la masse du robot. Ainsi, en considérant les bras d'une seule jambe plus un tube sur les quatre de la plateforme mobile et de la base, on obtient donc la fonction objectif suivante:

$$obj(x) = \rho \times pi \times d^2 \times (l1 + l2 + a + b)$$

1.4 Contraintes nonlcon.m:

- $C_{eq}(l) = err$ représentent les contraintes géométriques et traduisent l'existence du modèle géométrique inverse le long de la trajectoire.
- $C(2 * l) = -1/cond(B) + 0.1$ représentent les contraintes de dextérité sérielles le long de la trajectoire.
- $C(2 * l - 1) = -abs(det(A)) + \epsilon$ représentent les contraintes de non-singularité de la jacobienne parallèle.

Nous avons essayé de faire l'optimisation dans un premier temps avec la contrainte $cond(A) < max_{cond}$, mais le programme d'optimisation n'arrivait pas à converger même si nous augmentions la valeur de max_{cond}, jusqu'à atteindre la valeur élevée de 10^{12}. Nous avons ainsi essayé d'optimiser en évitant les singularités de A mais sans tenir compte de la contrainte de dextérité imposée par le sujet, car impossible à respecter dans notre cas.

2 Modèle géométrique inverse:

Model.m (click here to see code)

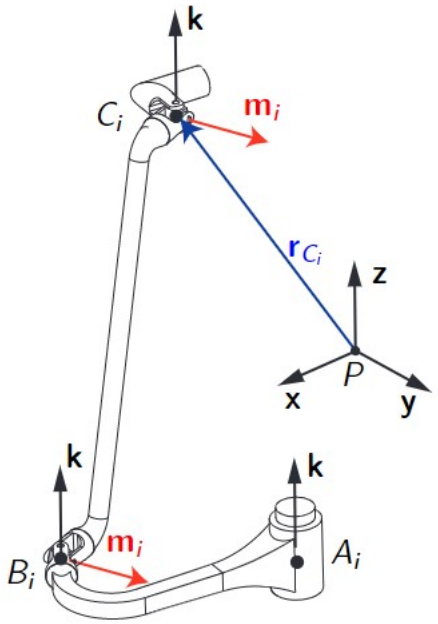

Figure 2: RUU limb

Pour déterminer le modèle géométrique inverse du mécanisme étudier (trouver le vecteur $[\phi_1\phi_2\phi_3\phi_4]$) il suffit de déterminer les coordonnées du point B_i pour chaque jambe.

Formulation mathématique:
Soit $i = 1, 2, 3, 4$,
Le point B_i peut s'exprimer comme l'intersection d'une sphère et d'un cercle :

$$B_i \in \mathcal{C}(A_i, l_1) \cap \mathcal{S}(C_i, l_2)$$

Solution numérique:
Cette intersection nous amène à la résolution d'un système à 2 équations 2 inconnues mais non linéaire
Ainsi, par une approche symbolique sur Matlab nous avons essayé de trouver un moyen pour inverser ce système. (Solu.m)

```
syms x_b y_b x_a x_c y_a y_c z_c l1 l2;

Cercle = (x_b-x_a)^2+(y_b-y_a)^2 == l1^2;
Sphere = (x_b-x_c)^2+(y_b-y_c)^2+(z_c)^2 == l2^2;

eqn = [Cercle , Sphere];
S= solve(eqn,[x_b,y_b]);

Sx=simplify(S.x_b(1));
Sy=simplify(S.y_b(1));
```

Nous avons ensuite implémenté le résultat trouvé dans la fonction [B,err] = trouver$_B(i, P, a, b, l1, l2)$ qui donne les coordonnés du point B si il existe une intersection, sinon elle retourne une err = 1 .

Soit x4 le vecteur d'optimisation de la contrainte géométrique seulement.

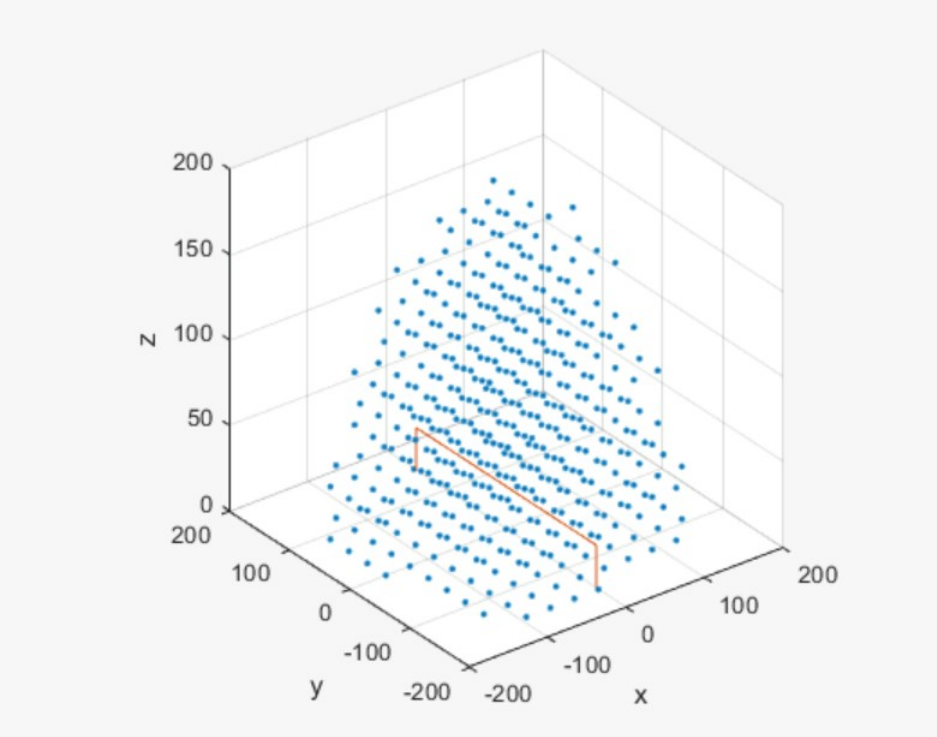

Figure 3: Espace atteignable par le robot correspondant a x4

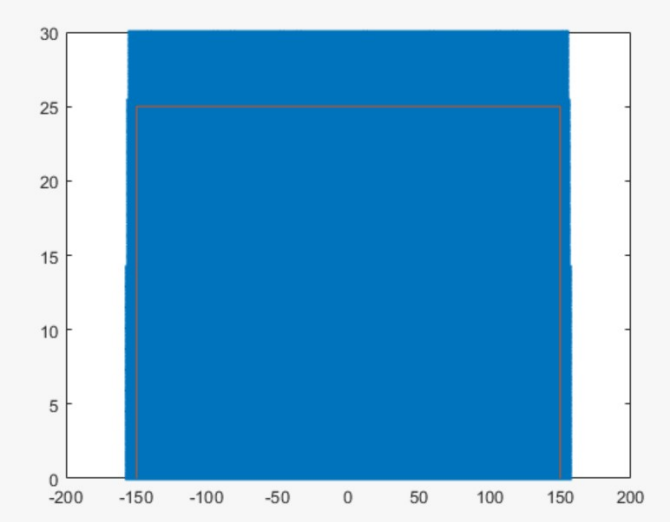

Figure 4: Espace atteignable suivant le plan de la trajectoire par le robot

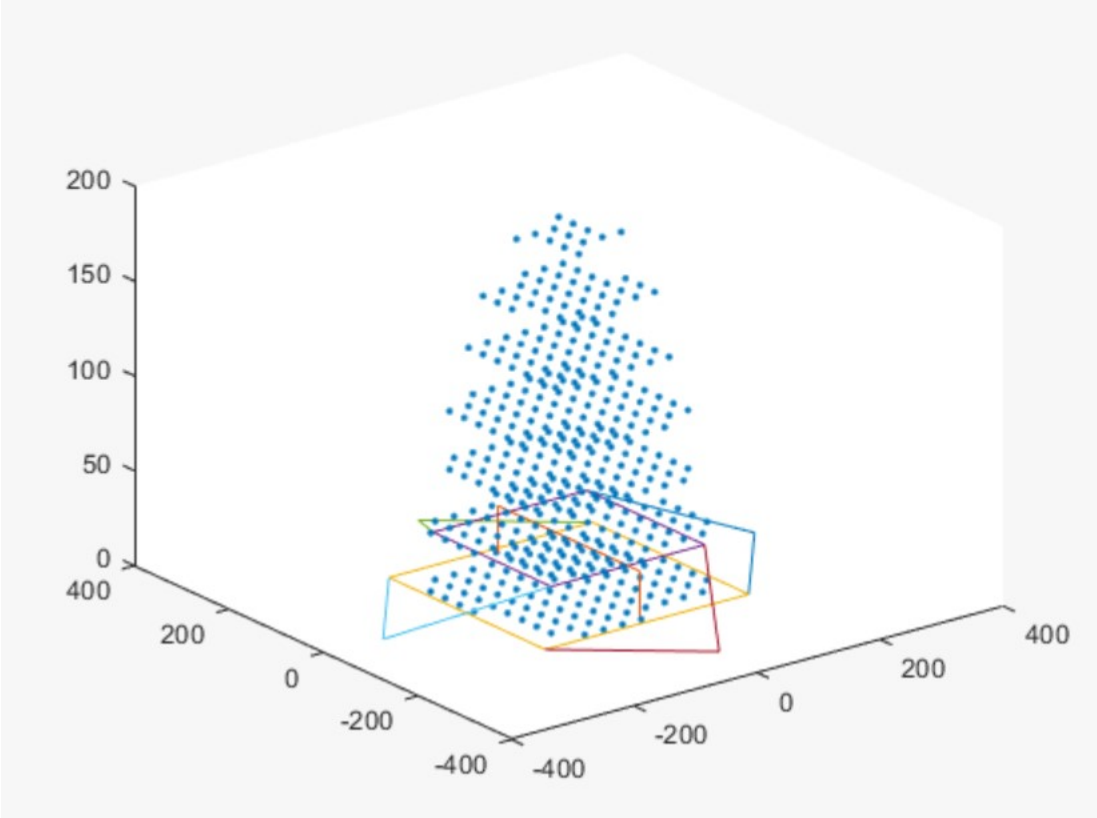

Figure 5: Espace atteignable par le robot correspondant a x4

Remarque: Dans un premier temps nous avons intégré le code (Solu.m) dans la fonction trouver$_B$.m. Cela fonctionnait, mais pour la partie optimisation il fallait attendre la résolution symbolique pour chaque itération et c'est pour cela qu'on a fait la résolution ailleurs et on a pris que les solutions.

En donnant les coordonnées du point P et l'orientation de l'effecteur, on définit les positions des points C_i cela est codé au niveau de la fonction Ci = trouver$_{Ci}(P, i, a)$, les points A_i sont fixes et générés par la fonction Ai = trouver$_A i(i, b)$

Validation du MGI:

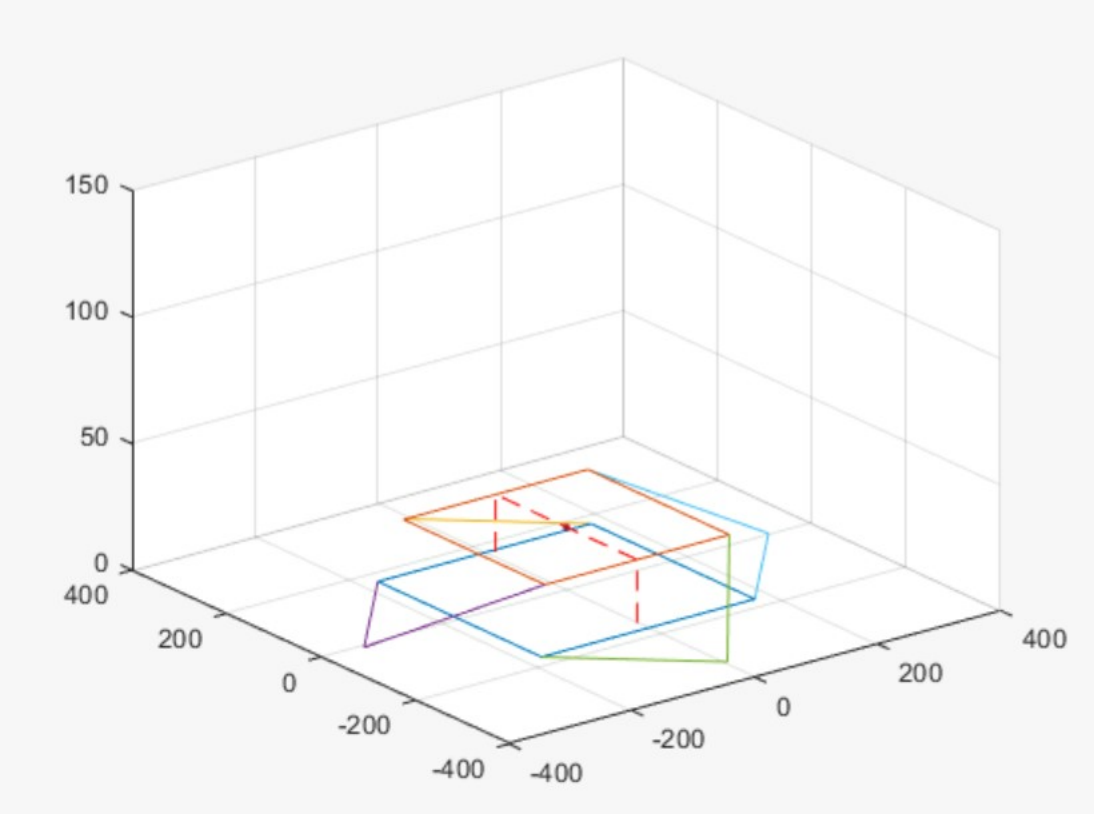

Figure 6: MGI pour P = [0 0 0 pi/2]

Notre MGI.m donne qu'une seule solution parmi les 4^2 solutions. Ce qui est important de vérifier dans notre fonction de contraintes d'optimisation, c'est justement l'existence de ce MGI.

3 Matrices jacobienne:

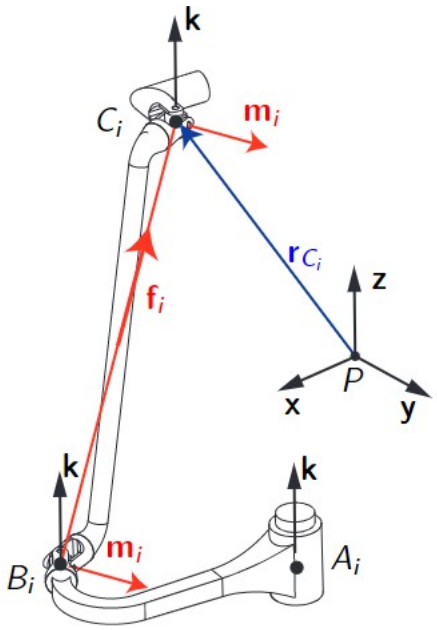

Figure 7: RUU limb

Les deux fonctions $\text{Jacob}_A(P, b, a, l1, l2)$ et $\text{Jacob}_B(P, b, a, l1, l2)$ calculent les matrices jacobiennes réduites parallèle et sérielle pour une configuration du robot donnée, selon leurs expressions analytiques suivantes.

$$A = \begin{bmatrix} (r_{C1} \times f_1)^T & f_1^T \\ (r_{C2} \times f_2)^T & f_2^T \\ (r_{C3} \times f_3)^T & f_3^T \\ (r_{C4} \times f_4)^T & f_4^T \\ (k \times m_1)^T & O_3^T \\ (k \times m_2)^T & O_3^T \end{bmatrix}$$

$$B = \begin{bmatrix} (\overrightarrow{A_1C_1} \times f_1)^T k & 0 & 0 & 0 & 0 & 0 \\ 0 & (\overrightarrow{A_2C_2} \times f_2)^T k & 0 & 0 & 0 & 0 \\ 0 & 0 & (\overrightarrow{A_3C_3} \times f_3)^T k & 0 & 0 & 0 \\ 0 & 0 & 0 & (\overrightarrow{A_4C_4} \times f_4)^T k & 0 & 0 \\ 0 & 0 & 0 & 0 & (\overrightarrow{A_5C_5} \times f_1)^T k & 0 \\ 0 & 0 & 0 & 0 & 0 & (\overrightarrow{A_6C_6} \times f_1)^T k \end{bmatrix}$$

4 Recherche du point initiale pour fmincon:

4.1 Problème:

Comme nous le savons, la fonction d'optimisation fmincon de Matlab est une fonction de recherche de minimum **local** sous certaines contraintes, d'où la nécessité de trouver le bon point initial x0 pour que le problème converge vers une solution pertinente.

Si on donne un x0 situé dans un voisinage où les contraintes ne sont pas satisfaites la fonction fmincon va retourner un 'infeasible point' vu qu'après plusieurs itérations elle a pas pu trouver un point vérifiant les contraintes.

```
    % Converged to an infeasible point.

fmincon stopped because the size of the current step is less than
the value of the step size tolerance but constraints are not
satisfied to within the value of the constraint tolerance.
```

4.2 Méthode utilisée:

Nous avons donc fait des tests différents et à chaque fois nous changions x_0 par $x_{0_{new}}$ donné par le test suivant:

1. On supprime quelques contraintes du problème (on a choisi de laisser que les contraintes d'existence du MGI le long de la trajectoire)
2. On donne à fmincon une fonction objectif nulle pour qu'elle essaye de seulement satisfaire les contraintes géométriques sans minimisation.
3. Retour aux contraintes initiales (dextérité pour la matrice B puis les non-singularités de A).

Remarque: On a considéré que la contrainte d'existence du MGI le long de la trajectoire est la plus forte: ce qui est normal puisqu'on peut pas calculer les matrices jacobienne sans inversion du MGD, du coup dans le ficher nonlcon.m on ajoute les contraintes cinématiques que si la contrainte géométrique est bien vérifiée.

5 Résultats:

Pour l'aluminium (ρ=2800 kg/m^3 , d = 20 mm) on a trouvé:

5.1 Optimisation en appliquant les contraintes géométriques:

```
%Aluminium
    x4 =

    224.9162    218.9266    249.9524    331.1366
%     l1          l2          a           b

>> m=obj(x4)
    m =

            3.6063
```

run code animation.m to see animation.mp4.

5.2 Optimisation en appliquant les contraintes géométriques sans considérer les singularités de la matrice A

```
%Aluminium
    x7 =

    330.4928   331.0376    473.7821    472.4131
%     l1          l2          a           b

>> m=obj(x7)
    m =

            5.6569
```

5.3 Optimisation en appliquant toutes les contraintes géométriques et cinématiques

```
%Aluminium
    x =

    363.5871   349.0358   562.9278    472.4136
%     l1          l2          a           b

>> m=obj(x)
    m =

              6.1504
```