Modélisation d’un robot à câbles et estimation des paramètres

—

Projet Mécatronique

Gwezheneg RIVIERE

Ngatam THIEBAUT

Arts et Métiers Sciences et Technologies

Table des matières

Introduction et contexte :

Contexte

Nous sommes 2 élèves ingénieurs en fin de parcours scolaire. Tous les 2 dans l'expertise mécatronique, une expertise qui allie la mécanique, l'informatique, l'électronique et l'automatique afin de faire des systèmes complexes automatisé. Notre projet de fin d'étude est porté sur un robot à câble (cf. figure ci-dessous). Ce robot se trouve actuellement dans le hall 3 des ateliers du campus des Arts et Métiers de Paris. Nous avons 5 mois (de février 2025 à juin 2025) pour faire la Modélisation d'un robot à câbles et estimation de paramètres. Vous trouverez une capture de la fiche de PJE en annexes, cette fiche nous a été fourni en début de PJE pour nous indiquer le travail attendu sur cet aspect du robot, en plus de ces tâches, il est important de noter que les modèles du robot que nous allions faire (modèle géométrique, cinématique et dynamique), allaient être utilisé par nos pairs dans leurs travaux.

Figure 1: Robot à câble du Hall 3 des Ateliers du campus des Arts et Métiers de Paris (vu de profil)

Figure 2: Robot à câble du Hall 3 des Ateliers du campus des Arts et Métiers de Paris (vu de face)

Robot à câbles

Les robots à câble sont des types de robot avec une faible inertie dû au fait qu'il ne s'agit que d'un effecteur (une plaque en aluminium laminée et découpé en carré de côté 230mm et d'épaisseur 2mm). Ces robots à câbles représentent une solution innovante dans le domaine de la robotique pour la manipulation d'objets à grande échelle, la capture de mouvements rapides ou encore l'exploration d'environnements complexes. Contrairement aux robots articulés traditionnels, ces systèmes utilisent des câbles tendus entre des treuils fixés à une structure et une plate-forme mobile, appelée effecteur. La position et l'orientation de cette plate-forme sont contrôlées en ajustant précisément la longueur des câbles.

Cette architecture présente plusieurs avantages notables : une grande mobilité, une structure légère, des vitesses élevées, et un coût réduit pour les charges utiles importantes. On retrouve les robots à câbles dans des applications variées, allant des simulateurs de mouvement (comme dans les simulateurs de vol ou de sports extrêmes), aux plateformes de manutention industrielle, en passant par la capture de mouvements dans le domaine du cinéma ou de la recherche biomécanique.

Ce projet vise à modéliser, simuler et analyser le comportement cinématique d'un robot à câbles, en mettant l'accent sur la relation entre les longueurs de câbles et la position de l'effecteur dans un plan 2D ou 3D. Il s'agit également d'explorer les aspects liés au modèle direct, au modèle inverse, à la commande et à la visualisation du système à travers des animations et des représentations graphiques.

Objectifs

La partie de ce projet que nous allons traiter étant sa modélisation et l'estimations de ses paramètres, nous avons commencé par créer un modèle géométrique simple du robot, à la suite de tests et en comparant les résultats obtenus avec la simulation et les résultats réelles, nous allons complexifier notre modèle afin de le rendre le plus proche possible du modèle réel. Nous passerons ensuite à un modèle cinématique, puis nous referons des tests avec le modèle réelle et notre simulation avec de mettre à l'épreuve ce nouveau modèle avant de passer au modèle dynamique.

Durant ce projet, nous avons, en plus des objectifs cités ci-dessus, trouvé d'autres objectifs à atteindre, ces objectifs sont recensés ci-dessous :

Objectif	Avancement
Modèle géométrique direct du robot	✓
Modèle géométrique inverse du robot	✓
Implantation des modèles géométriques sur Python	✓
Tests sur les modèles géométrique	✓
Complexification des modèles géométriques	✓
Modèle cinématique direct du robot	✓
Modèle cinématique inverse du robot	✓
Implantation des modèles cinématiques sur Python	✓
Tests sur les modèles cinématiques	🧱
Complexification des modèles cinématiques	🧱
Modèle dynamique	✓
Explication des codes et mise en ligne	✓
Test avec les tensions de câbles	🧱

🧱 Commencé mais pas fini

✓ Fait

Remerciements

Ce PJE n'aurait jamais autant avancée sans l'aide de certaines personnes qui nous ont aidé ; nous souhaitions donc rendre à César ce qui lui est dû en dédicaçant ce paragraphe à ces personnes. Tout d'abord, nous remercions les personnes qui nous ont proposé et qui supervisé ce PJE M. M. Rebillat et M. P. Margerit, qui nous a aussi poussé à utiliser le site GitHub afin de rendre notre travail (notamment le code de la simulation), à jour en temps réel. Nous remercions aussi, M. M. Guskov qui nous a aidé et débloqué de certaines situations lors de nos entretiens.

Enfin, nous remercions tous nos pairs, qui nous ont vraiment bien aidé et accompagné tout au long de ce PJE.

Merci à vous tous.

Schémas :

Schéma des nominations :

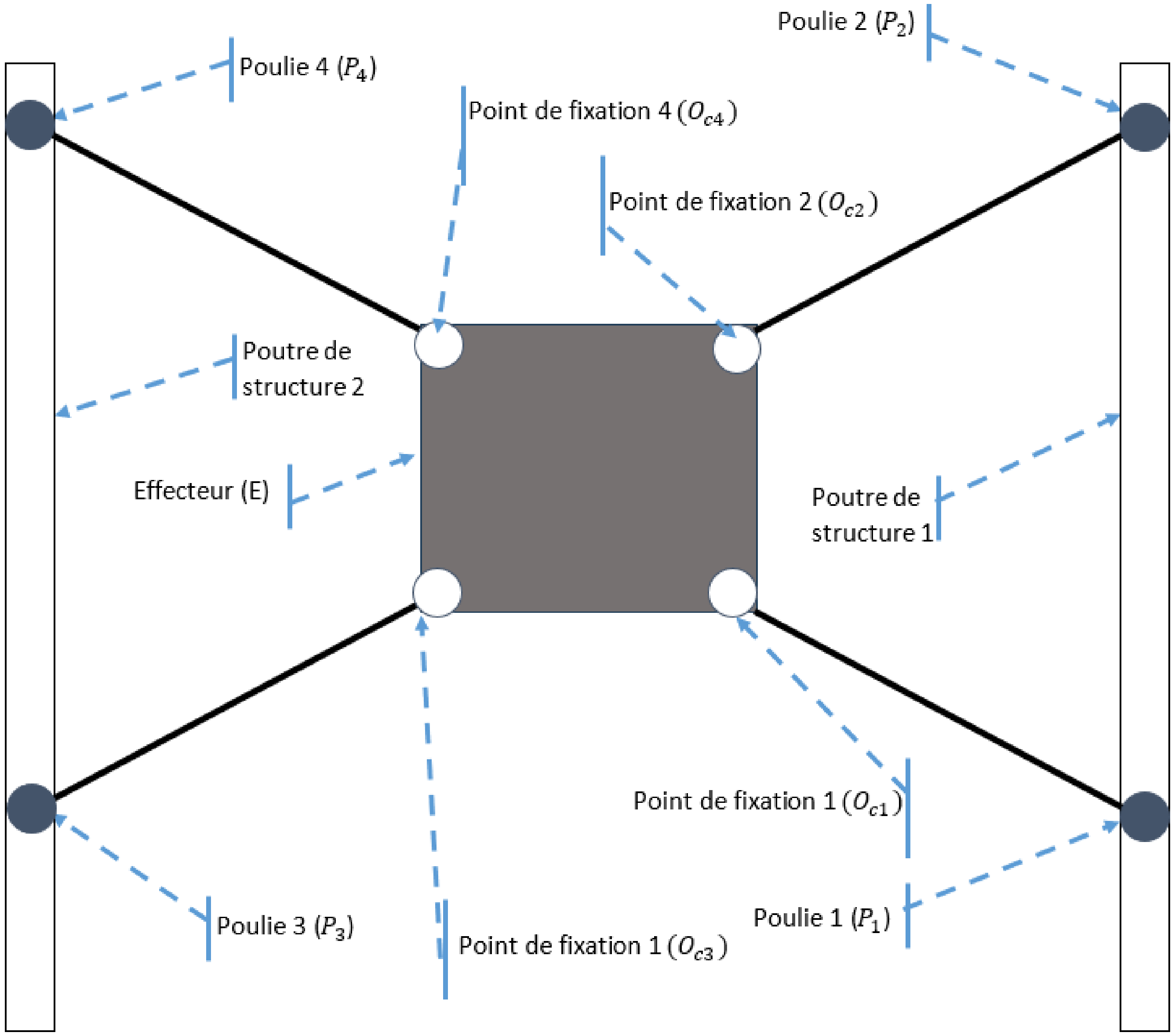


Figure 1: Schéma des nominations

Schéma des dimensions :

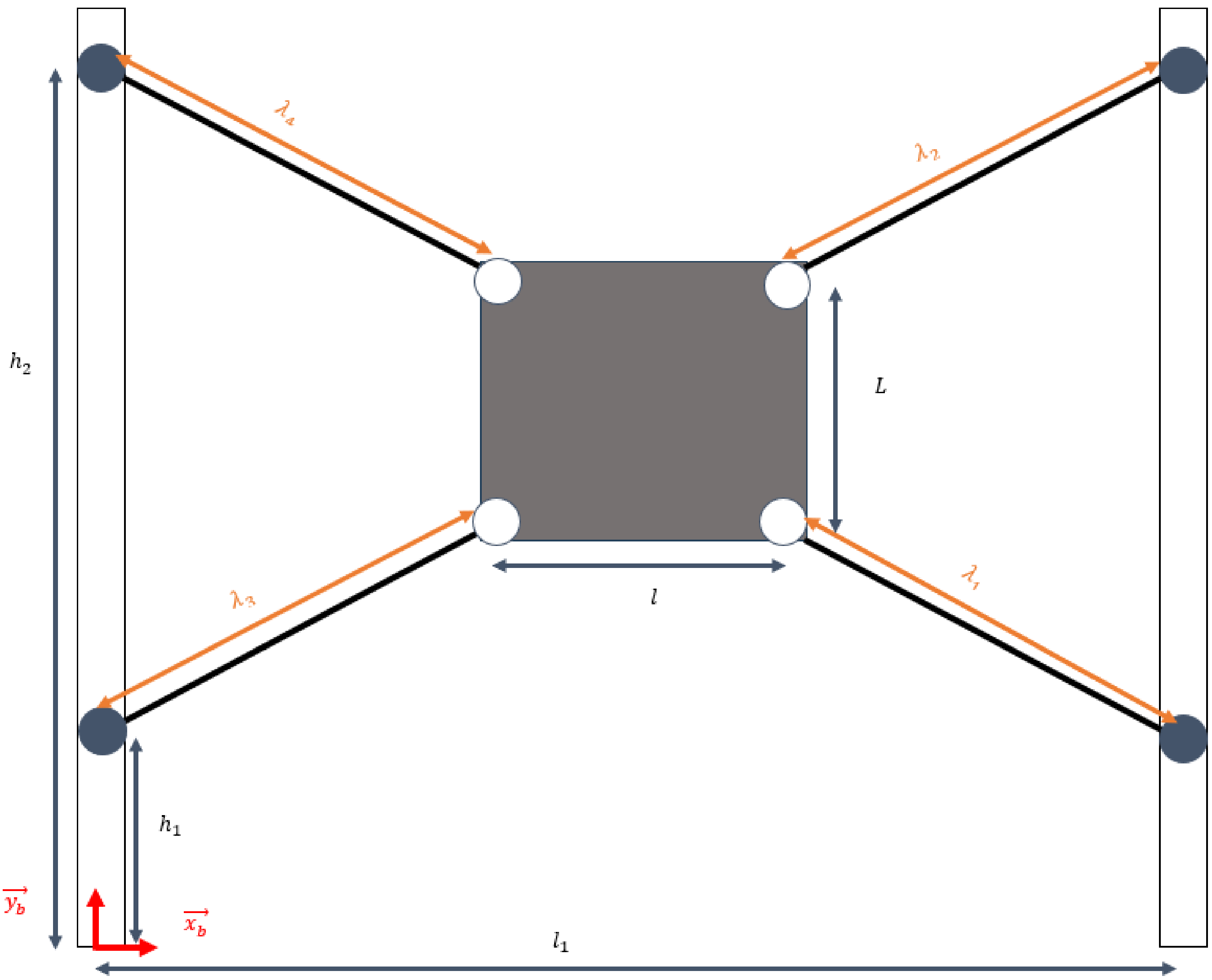


Figure 2: Schéma des dimensions

Dimension	Explication	Dimensions variables ?
h_1	Hauteur 1, hauteur des poulies 1 et 3 par rapport au sol. Dimension	Non
h_2	Hauteur 2, hauteur des poulies 2 et 4 par rapport au sol.	Non
l_1	Longueur 1, longueur entre les pieds de la structure du robot à câbles.	Non
L	Longueur de la plaque de l'effecteur.	Non
l	Largeur de la plaque de l'effecteur.	Non
λ_i	Longueur du câbles i.	Oui

Tableau 1: Tableau des dimensions du robot

Schéma des bases et repères :

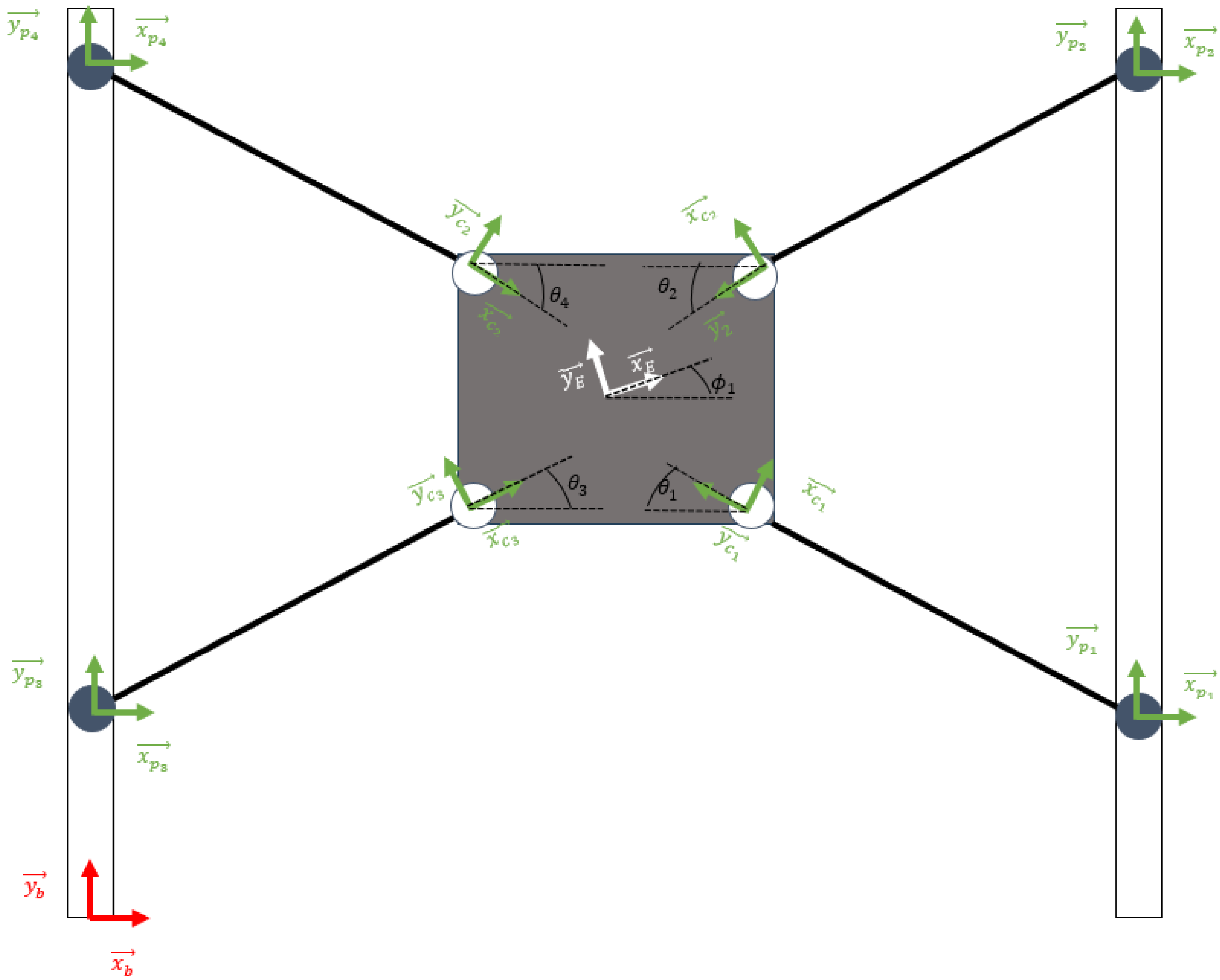


Figure 3: Schéma des bases et répères

Repère	Objet associé
$(O_b, \overrightarrow{x_b}, \overrightarrow{y_b}, \overrightarrow{Z_b})$	Structure fixe du robot à câbles. Repère parallèle au sol.
$(O_{p_i}, \overrightarrow{x_{p_i}}, \overrightarrow{y_{p_i}}, \overrightarrow{Z_{p_i}})$	Poulie i du robot à câbles. Repère parallèle au sol.
$(O_{c_i}, \overrightarrow{x_{c_i}}, \overrightarrow{y_{c_i}}, \overrightarrow{Z_{c_i}})$ Pour i = 3,4	Point d'accroche i sur l'effecteur du robot à câbles. Repère avec un angle $\theta_i = (\overrightarrow{x_b}, \overrightarrow{x_{c_i}})$.
$(O_{c_i}, \overrightarrow{x_{c_i}}, \overrightarrow{y_{c_i}}, \overrightarrow{Z_{c_i}})$ Pour i = 1, 2	Point d'accroche i sur l'effecteur du robot à câbles . Repère avec un angle $\theta_i = (-\overrightarrow{x_{p_b}}, \overrightarrow{y_{c_i}})$
$(\phi_1, \overrightarrow{x_E}, \overrightarrow{y_E}, \overrightarrow{Z_E})$	Effecteur du robot à câbles. Repère avec un angle $\phi_1 = (\overrightarrow{x_b}, \overrightarrow{x_E})$.

Tableau 2: Tableau des bases et repères du robot

Modèle géométrique :

Fermetures géométriques

Fermeture géométrique par la poulie 1, $\phi_1 \neq 0$: $\overrightarrow{O_bO_{p_1}} + \overrightarrow{O_{p_1}O_{c_1}} + \overrightarrow{O_{c_1}O_e} + \overrightarrow{O_eO_b} = \vec{O}$

- ⇨ $\overrightarrow{\mathrm{x_b}}$: $X_e = -\lambda_1 \sin(\theta_1) - \frac{1}{2}[lcos(\phi_1) + Lsin(\phi_1)] + l_1$
- ⇨ $\overrightarrow{y_e}$: $Y_e = \lambda_1 \cos(\theta_1) + \frac{1}{2}[-lsin(\phi_1) + Lcos(\phi_1)] + h_1$

Fermeture géométrique par la poulie 2, $\phi_1 \neq 0$: $\overrightarrow{O_bO_{p_2}} + \overrightarrow{O_{p_2}O_{c_2}} + \overrightarrow{O_{c_2}O_e} + \overrightarrow{O_eO_b} = \vec{O}$

- ⇨ $\overrightarrow{\mathrm{x_b}}$: $X_e = -\lambda_2 \cos(\theta_2) - \frac{1}{2}[lcos(\phi_1) - Lsin(\phi_1)] + l_1$
- ⇨ $\overrightarrow{y_e}$: $Y_e = -\lambda_2 \sin(\theta_2) - \frac{1}{2}[lsin(\phi_1) + Lcos(\phi_1)] + h_2$

Fermeture géométrique par la poulie 3, $\phi_1 \neq 0$: $\overrightarrow{O_bO_{p_3}} + \overrightarrow{O_{p_3}O_{c_3}} + \overrightarrow{O_{c_3}O_e} + \overrightarrow{O_eO_b} = \vec{O}$

- ⇨ $\overrightarrow{\mathrm{x_b}}$: $X_e = \lambda_3 \cos(\theta_3) + \frac{1}{2}[lcos(\phi_1) - Lsin(\phi_1)]$
- ⇨ $\overrightarrow{y_e}$: $Y_e = \lambda_3 \sin(\theta_3) + \frac{1}{2}[lsin(\phi_1) + Lcos(\phi_1)] + h_1$

Fermeture géométrique par la poulie 4, $\phi_1 \neq 0$: $\overrightarrow{O_bO_{p_4}} + \overrightarrow{O_{p_4}O_{c_4}} + \overrightarrow{O_{c_4}O_e} + \overrightarrow{O_eO_b} = \vec{O}$

- ⇨ $\overrightarrow{\mathrm{x_b}}$: $X_e = \lambda_4 \cos(\theta_4) + \frac{1}{2}[lcos(\phi_1) + Lsin(\phi_1)]$
- ⇨ $\overrightarrow{y_e}$: $Y_e = -\lambda_4 \sin(\theta_4) + \frac{1}{2}[lsin(\phi_1) - Lcos(\phi_1)] + h_2$

Modèle géométrique direct

On a donc :

(1) : $X_e = -\lambda_1 \sin(\theta_1) - \frac{1}{2}[lcos(\phi_1) + Lsin(\phi_1)] + l_1$

(2) : $Y_e = \lambda_1 \cos(\theta_1) + \frac{1}{2}[-lsin(\phi_1) + Lcos(\phi_1)] + h_1$

(3) : $X_e = -\lambda_2 \cos(\theta_2) - \frac{1}{2}[lcos(\phi_1) - Lsin(\phi_1)] + l_1$

(4) $Y_e = -\lambda_2 \sin(\theta_2) - \frac{1}{2}[lsin(\phi_1) + Lcos(\phi_1)] + h_2$

(5) : $X_e = \lambda_3 \cos(\theta_3) + \frac{1}{2}[lcos(\phi_1) - Lsin(\phi_1)]$

(6) : $Y_e = \lambda_3 \sin(\theta_3) + \frac{1}{2}[lsin(\phi_1) + Lcos(\phi_1)] + h_1$

(7) : $X_e = \lambda_4 \cos(\theta_4) + \frac{1}{2}[lcos(\phi_1) + Lsin(\phi_1)]$

(8) : $Y_e = -\lambda_4 \sin(\theta_4) + \frac{1}{2}[lsin(\phi_1) - Lcos(\phi_1)] + h_2$

Modèle géométrique inverse

Ces équations nous permettent d'avoir les expressions les longueurs des câbles λ_i:

(9) : $\lambda_1 = \sqrt{(-\frac{1}{2}(lcos(\phi_1) + Lsin(\phi_1) - X_e + l_1)^2 + (+\frac{1}{2}(lsin(\phi_1) - Lcos(\phi_1) + Y_e \; - h_1)^2}$

(10) : $\lambda_2 = \sqrt{(-\frac{1}{2}(lcos(\phi_1) - Lsin(\phi_1) - X_e + l_1)^2 + (-\frac{1}{2}(lsin(\phi_1) + Lcos(\phi_1) - Y_e + h_2)^2}$

(11) : $\lambda_3 = \sqrt{(-\frac{1}{2}(lcos(\phi_1) - Lsin(\phi_1) + X_e)^2 + (-\frac{1}{2}(lsin(\phi_1) + Lcos(\phi_1) + Y_e - h_1)^2}$

(12) : $\lambda_4 = \sqrt{(-\frac{1}{2}(lcos(\phi_1) - Lsin(\phi_1) + \; X_e)^2 + (\frac{1}{2}(lsin(\phi_1) - Lcos(\phi_1) - Y_e + h_2)^2}$

On notera que les longueurs des câbles (λ_i) sont donné par la relation :

$$\lambda_i = r_i * q_i \,,$$

$\underline{Avec}$:

$r_i : le\ cofficient\ d'enroulement\ de\ l'enrouleur\ i$

$q_i : angle\ de\ rotation\ de\ l'enrouleur\ i$

Dans notre étude, on supposera que les enrouleurs sont les mêmes on a donc :

$$r_i = r = \kappa * \sqrt{e^2 + \frac{\rho^2}{2\pi}}$$

$\underline{Avec}$:

$\kappa : rapport\ de\ réduction\ des\ enrouleurs$

$e : rayon\ des\ enrouleurs$

$\rho : pas\ des\ enrouleurs$

Ces relations nous permettent d'avoir un lien directe entre les longues des câbles i et les angles de rotation des moteurs i. Ces angles de rotation étant les paramètres commandables, ils seront les valeurs en entrée de notre système.

(13) : $q_1 = \frac{1}{r}\sqrt{(-\frac{1}{2}(lcos(\phi_1) + Lsin(\phi_1) - X_e + l_1)^2 + (+\frac{1}{2}(lsin(\phi_1) - Lcos(\phi_1) + Y_e \; - h_1)^2}$

(14) : $q_2 = \frac{1}{r}\sqrt{(-\frac{1}{2}(lcos(\phi_1) - Lsin(\phi_1) - X_e + l_1)^2 + (-\frac{1}{2}(lsin(\phi_1) + Lcos(\phi_1) - Y_e + h_2)^2}$

(15) : $q_3 = \frac{1}{r}\sqrt{(-\frac{1}{2}(lcos(\phi_1) - Lsin(\phi_1) + X_e)^2 + (-\frac{1}{2}(lsin(\phi_1) + Lcos(\phi_1) + Y_e - h_1)^2}$

(16) : $q_4 = \frac{1}{r}\sqrt{(-\frac{1}{2}(lcos(\phi_1) - Lsin(\phi_1) + \; X_e)^2 + (\frac{1}{2}(lsin(\phi_1) - Lcos(\phi_1) - Y_e + h_2)^2}$

Pour le modèle cinématique, nous devons d'abord définir la position initiale de l'effecteur dans le repère de la base. On nomme cette position $X_{initial} = (X_0\ Y_0\ \phi_0)^\tau$, on calculera les longueurs de câbles liées à cette position plus tard, pour l'instant on note ces longueurs $\Lambda_0 = [\lambda_{1_0}\ \lambda_{2_0}\ \lambda_{3_0}\ \lambda_{4_0}]^\tau$, et on note le vecteur $Q_0 = [q_{1_0}\ q_{2_0}\ q_{3_0}\ q_{4_0}]^\tau$, le vecteur des position angulaire des moteur associé à cette position initiale.

On a donc pour un déplacement à partir de cette position de base, à un point de coordonnées $X_{final} = [X_1\ Y_1\ \phi_{1_1}]$, dont les longueurs de câbles associé sont $\Lambda_1 = [\lambda_{1_1}\ \lambda_{2_1}\ \lambda_{3_1}\ \lambda_{4_1}]^\tau$et les position angulaires des moteurs sont $Q_1 = [q_{1_1}\ q_{2_1}\ q_{3_1}\ q_{4_1}]^\tau$

La variation des positions angulaires qui est donnée par

$$\boxed{\textbf{(17)} \qquad : Q_1 - Q_0 = R^{-1} * (\Lambda_1 - \Lambda_0)}$$

Avec $R = diag(r, r, \ldots, r)$, de taille 4x4

Nous allons donc implémenter ces équations sur Python afin de créer notre 1er modèle, ce modèle est basé sur la fermeture géométrique de notre système.

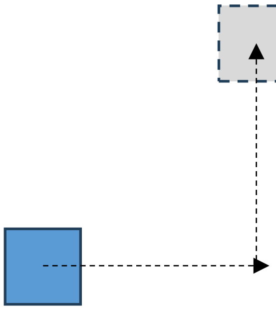
Modèle 1– fermeture géométrique

Vérification du modèle géométrique

Test, objectif et attentes

L'objectif de ce test est de valider le modèle géométrique et le modèle cinématique que nous vous avons présenté ci-dessus, et que nous avons implanté dans un programme Python.

Le test que l'on va faire est le suivant, une translation de 1m (+ 1000mm sur l'axe $\overrightarrow{x_b}$) et translation de 1m vers le haut (+ 500 mm sur l'axe $\vec{y}_b$). Et l'on souhaite connaitre la longueur des câbles au cours de ce déplacement.



Pour cela, on a besoin définir une position initiale et de connaitre la longueur des câbles à cette position. On définit notre position initial comme étant le centre de notre robot à câbles et l'on en déduit les coordonnées finales avec le déplacement voulu.

(Notons pour ce premier test que l'on va considérer une rotation suivant l'axe z nulle, nous regarderons cela plus tard).

Coordonnées initiales	Coordonnés finales
$X_{initial} = \frac{l_1}{2} = 1075\ mm$	$X_{final} = X_{initial} + 500 = 1575\ mm$
$Y_{initial} = \frac{h_2}{2} = 1090\ mm$	$Y_{final} = Y_{initial} + 500 = 1590\ mm$
$\phi_{1_{initial}} = 0\ rad$	$\phi_{1_{final}} = \phi_{1_{initial}} + 0 \ = 0\ rad$

Tableau 3: Tableau des coordonnées pour le test

L'on va donc effectuer ce déplacement sur notre modèle numérique et voir comment varient les longueurs de câbles pendant le test et comparer ces résultats avec ceux que nous avons en refaisant le même déplacement avec le robot à câble physique.

Si les résultats sont similaires, cela nous permet de valider nos modèles numériques et cela nous permet de passer au modèle dynamique.

Résultats du programme Python (Modèle 1)

On implémente le modèle du robot à câble décrit par les équations définies ci-dessus sur Python (le code du modèle 1 est disponible et expliqué dans les annexes). En tapant la commande :

```
inverse_plot(X_0+500,  Y_0+500, 0.0, 100, 10)
```

Dans le terminal de commande, on obtient les résultats suivants :

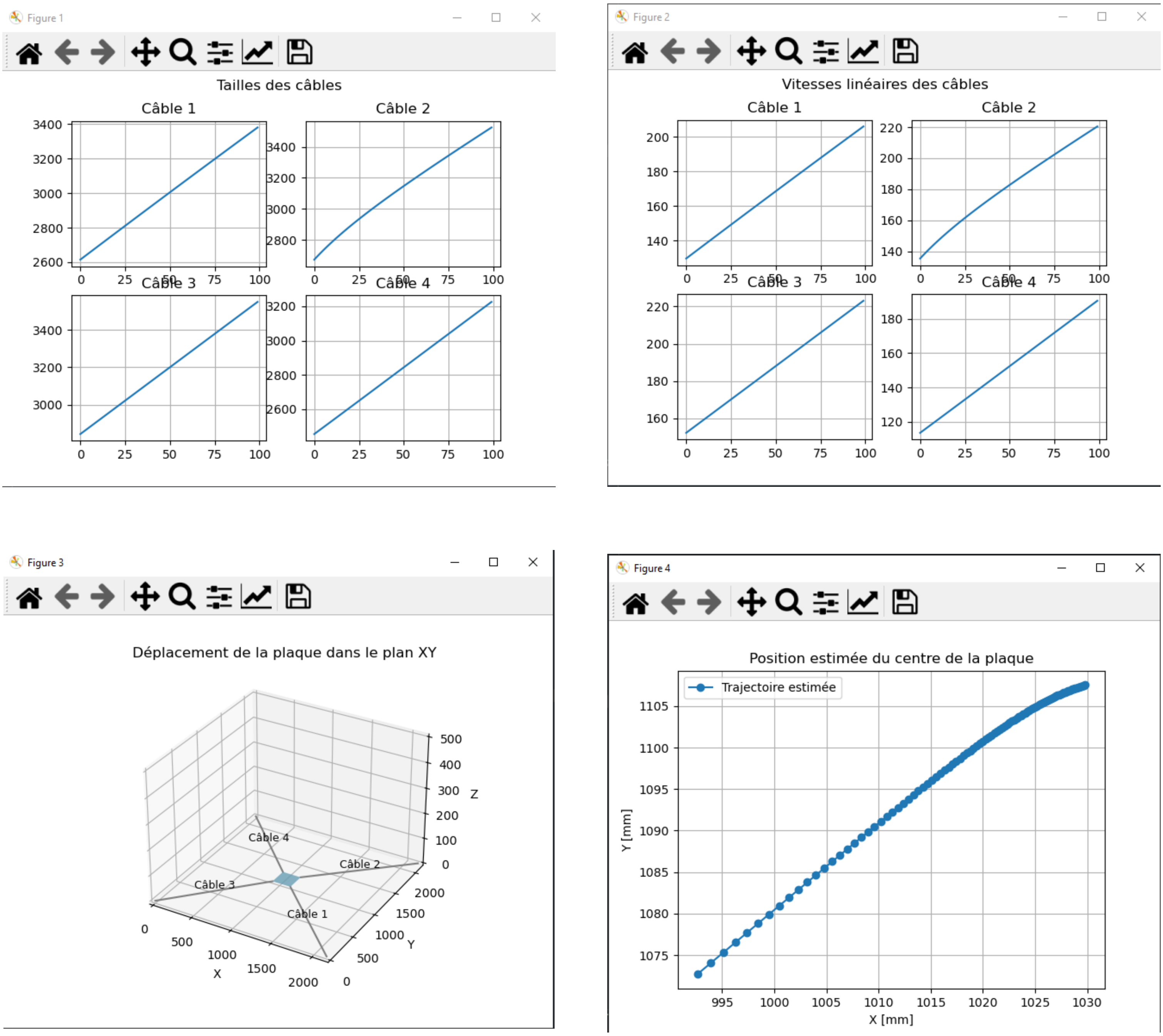


Figure 4: Résultats de simulation avec le modèle 1

Ces résultats de simulation sont intéressants car ils nous montrent que notre 1er modèle ne marche pas comme on le souhaite. En effet, on remarque que :

- L'effecteur n'atteint pas la position finale demandé : Après plusieurs simulations avec différents temps de simulation et plus d'étapes de simulation, l'effecteur n'arrive pas à faire ce déplacement de +500/+500.

Il nous faut donc **changer notre modèle quand bien même nos équations sont correctes afin d'avoir un simulation géométrique correcte pour passer sur le modèle cinématique.**

Après réflexion et des discussions avec nos pairs, nous avons opté pour un nouveau modèle, le suivi de point d'accroche sur l'effecteur.

Modèle 2 – Suivi des points d'accroche de l'effecteur

Ce modèle est basé sur 2 hypothèses :

- **Les câbles sont accrochés à d'une extrémité au centre des poulies et de l'autre, au point d'attache de l'effecteur.**
- **Les câbles sont toujours tendus**

À partir de ces hypothèses, on transforme chaque câble est un ligne droite parfait entre le centre des poulies (dont les coordonnées sont connues et invariante dans le temps), et les points d'accroche de l'effecteur (dont les coordonnées par rapport au centre de l'effecteur sont aussi connues et invariante dans le temps). Il nous suffit alors de trouver la distance minimale entre ces 2 points pour avoir les longueurs des câbles (notée λ sur le schéma ci-dessous).

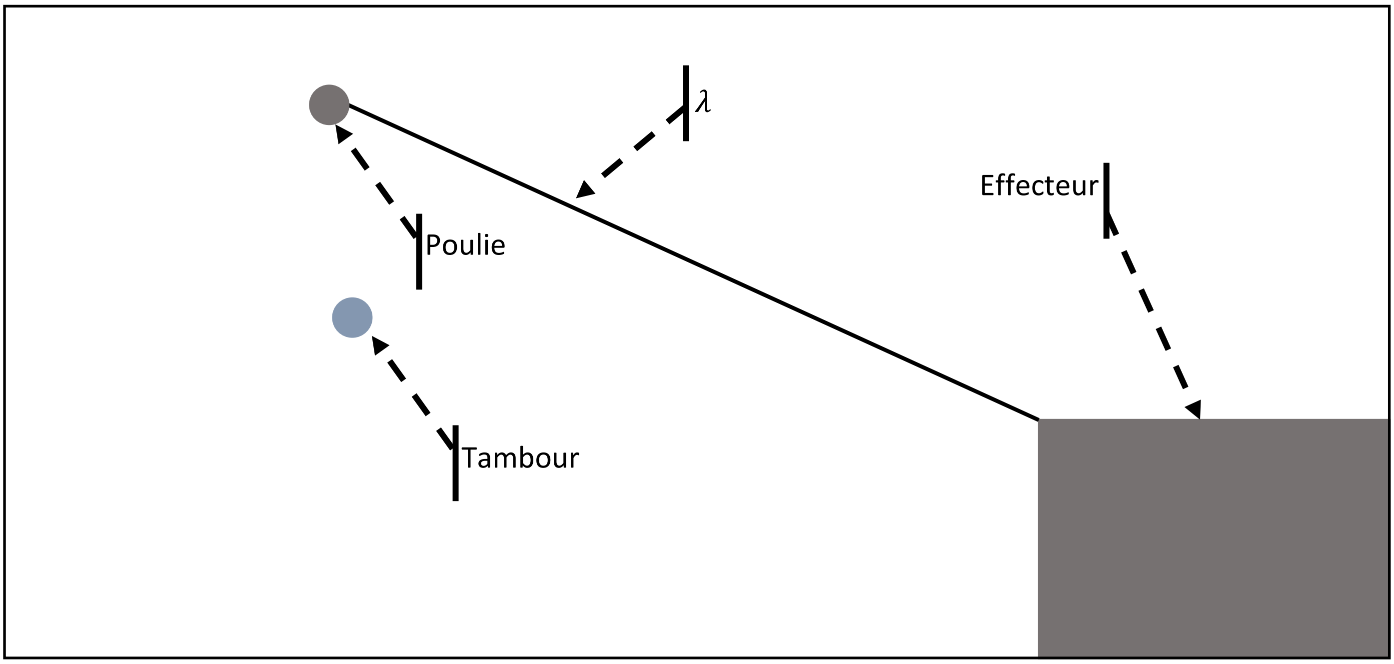


Figure 5: Schéma du modèle 2

Le code du modèle (qui sera appelé modèle 2 dans la suite) est disponible et expliqué dans les annexes.

En lance ensuite le code de ce nouveau modèle, pour faire le test avec notre modèle géométrique. Ce code crée un document Excel dans lequel sont regroupé les données des simulations :

Rappelons les hypothèses prises avec ce 2ème modèle :

- **Les câbles sont accrochés à d'une extrémité au centre des poulies et de l'autre, au point d'attache de l'effecteur.**
- **Les câbles sont toujours tendus**

La commande écrite pour le test est la suivante :

```
animation_2(X_0, Y_0, 0, X_0 + 500, Y_0 + 500, 0, 1, 0.001, 1000, 0)
```

1	Longueurs câble 1	Longueurs câble 2	Longueurs câble 3	Longueurs câble 4	Vitesses câble 1	Vitesses câble 2	Vitesses câble 3	Vitesses câble 4	Coordonées du centre	Coordonée
2	956,1511	1257,269	1055,758	1334,588	-1584,9	-9926,57	9745,278	-499,603	1010	1075
3	955,9926	1256,276	1056,733	1334,538	-1574,7	-9926,45	9745,753	-492,128	1010,707	1075,707
4	955,8352	1255,284	1057,707	1334,489	-1564,5	-9926,33	9746,227	-484,652	1011,414	1076,414
5	955,6787	1254,291	1058,682	1334,44	-1554,29	-9926,21	9746,7	-477,176	1012,121	1077,121
6	955,5233	1253,298	1059,656	1334,393	-1544,07	-9926,1	9747,172	-469,699	1012,828	1077,828
7	955,3689	1252,306	1060,631	1334,346	-1533,85	-9925,98	9747,642	-462,221	1013,536	1078,536
8	955,2155	1251,313	1061,606	1334,299	-1523,63	-9925,86	9748,111	-454,742	1014,243	1079,243
9	955,0631	1250,321	1062,581	1334,254	-1513,4	-9925,74	9748,579	-447,262	1014,95	1079,95
10	954,9118	1249,328	1063,556	1334,209	-1503,17	-9925,62	9749,045	-439,782	1015,657	1080,657
11	954,7615	1248,335	1064,53	1334,165	-1492,93	-9925,51	9749,511	-432,301	1016,364	1081,364
12	954,6122	1247,343	1065,505	1334,122	-1482,69	-9925,39	9749,974	-424,819	1017,071	1082,071
13	954,4639	1246,35	1066,48	1334,08	-1472,44	-9925,27	9750,437	-417,337	1017,778	1082,778
14	954,3167	1245,358	1067,455	1334,038	-1462,18	-9925,15	9750,898	-409,853	1018,485	1083,485
15	954,1705	1244,365	1068,431	1333,997	-1451,93	-9925,03	9751,358	-402,369	1019,192	1084,192
16	954,0253	1243,373	1069,406	1333,957	-1441,66	-9924,91	9751,817	-394,885	1019,899	1084,899
17	953,8811	1242,38	1070,381	1333,917	-1431,4	-9924,79	9752,275	-387,4	1020,607	1085,607
18	953,738	1241,388	1071,356	1333,878	-1421,12	-9924,67	9752,731	-379,914	1021,314	1086,314
19	953,5959	1240,395	1072,331	1333,84	-1410,85	-9924,55	9753,186	-372,427	1022,021	1087,021
20	953,4548	1239,403	1073,307	1333,803	-1400,57	-9924,42	9753,64	-364,94	1022,728	1087,728
21	953,3147	1238,41	1074,282	1333,767	-1390,28	-9924,3	9754,093	-357,452	1023,435	1088,435
22	953,1757	1237,418	1075,257	1333,731	-1379,99	-9924,18	9754,544	-349,964	1024,142	1089,142

...

589	1039,381	678,3001	1633,111	1430,675	4188,88	-9745,22	9894,309	3636,457	1425,072	1490,072
590	1039,8	677,3255	1634,101	1431,038	4196,807	-9744,48	9894,438	3642,519	1425,779	1490,779
591	1040,22	676,3511	1635,09	1431,403	4204,724	-9743,73	9894,566	3648,577	1426,486	1491,486
592	1040,64	675,3767	1636,08	1431,768	4212,631	-9742,98	9894,695	3654,63	1427,193	1492,193
593	1041,061	674,4024	1637,069	1432,133	4220,529	-9742,23	9894,822	3660,678	1427,9	1492,9
594	1041,484	673,4282	1638,059	1432,499	4228,417	-9741,47	9894,95	3666,722	1428,607	1493,607
595	1041,906	672,454	1639,048	1432,866	4236,295	-9740,71	9895,078	3672,761	1429,314	1494,314
596	1042,33	671,48	1640,038	1433,233	4244,164	-9739,94	9895,205	3678,796	1430,021	1495,021
597	1042,754	670,506	1641,027	1433,601	4252,024	-9739,18	9895,332	3684,826	1430,729	1495,729
598	1043,18	669,5321	1642,017	1433,969	4259,873	-9738,41	9895,458	3690,851	1431,436	1496,436
599	1043,606	668,5582	1643,006	1434,338	4267,714	-9737,63	9895,585	3696,872	1432,143	1497,143
600	1044,032	667,5845	1643,996	1434,708	4275,544	-9736,86	9895,711	3702,888	1432,85	1497,85
601	1044,46	666,6108	1644,985	1435,078	4283,365	-9736,08	9895,837	3708,899	1433,557	1498,557
602	1044,888	665,6372	1645,975	1435,449	4291,176	-9735,29	9895,963	3714,905	1434,264	1499,264
603	1045,317	664,6636	1646,965	1435,821					1434,971	1499,971
604										

Résultats du programme Python (Modèle 2)

		Câble 1	Câble 2	Câble 3	Câble 4	X	Y
Mesurée	Initiale	1010	1320	1010	1320	1010	1075
	Finale	1095	650	1690	1450	1510	1575
Simulée	Initiale	956,1511	1257,269	1055,758	1334,588	1010	1075
	Finale	1045,317	664,6636	1646,965	1435,821	1434,971	1499,971
Ecarts (%)	Initiale	5,631836	4,989476	4,334139	1,093068	0	0
	Finale	4,752876	2,206174	2,613018	0,98753	5,228595	5,002018
Ecart (mm)	Initiale	53,84886	62,73113	45,75802	14,58795	0	0
	Finale	49,68263	14,66364	43,03548	14,17916	75,02882	75,02882

Tableau 4: Résultats des tests sur le robot

On remarque qu'on a des écarts compris entre 1% et 10% entre les longueurs de câbles mesurées pour ces 2 positions et les longueurs de câbles obtenus lors de la simulation. Ces écarts sont compris entre 10mm et 70mm, on peut supposer que cela vient de nos hypothèses de départs, notamment sur notre représentation de des poulies. **Notre modèle est validé par l'expérience avec le système physique.**

Nous allons donc complexifier notre modèle afin de le rendre plus proche du système réel.

Complexification du modèle et prise en compte de l'enroulement autour des poulies

Changement d'hypothèse :

Dans cette partie, nous allons revenir sur l'hypothèse :

- **Les câbles sont accrochés à d'une extrémité au centre des poulies et de l'autre, au point d'attache de l'effecteur.**

Et, nous allons voir comment nous rapprocher plus de notre modèle.

Rappelons à quoi ressemble notre modèle (en pointillé noir) :

Figure 6: Complexification de notre modèle de base

Dans ce cas-là, notre modèle calcule la longueur de câble comme étant la distance entre le point d'accroche de l'effecteur (cercle vert) et le centre de la poulie (cercle noir), mais lorsqu'on regarde au niveau de la poulie, on comprend que le câble ne va pas à jusqu'à ce point-là, le câble va sur un point tangent à la poulie (cercle bleu sur la figure ci-dessous).

Figure 7: Zoom au niveau de la poulie du robot

L'on choisit alors de complexifier notre modèle afin de prendre en compte cette donnés. L'on va aussi considérer la longueur de câble entre la poulie et le tambour où elle est reliée. Pour prendre en compte cette complexification, nous devons considérer 3 nouvelles longueurs de câbles :

- λ' : la longueur de câble entre le point d'attache de l'effecteur et le point d'arrivée de la poulie (dans le cercle bleu ci-contre et dans le schéma ci-dessous).
- λ'' : la longueur de câble directement sur la gouttière de la poulie (en orange dans le schéma ci-dessous).
- λ''' : la longueur entre la poulie et le tambour (en jaune dans le schéma ci-dessous).

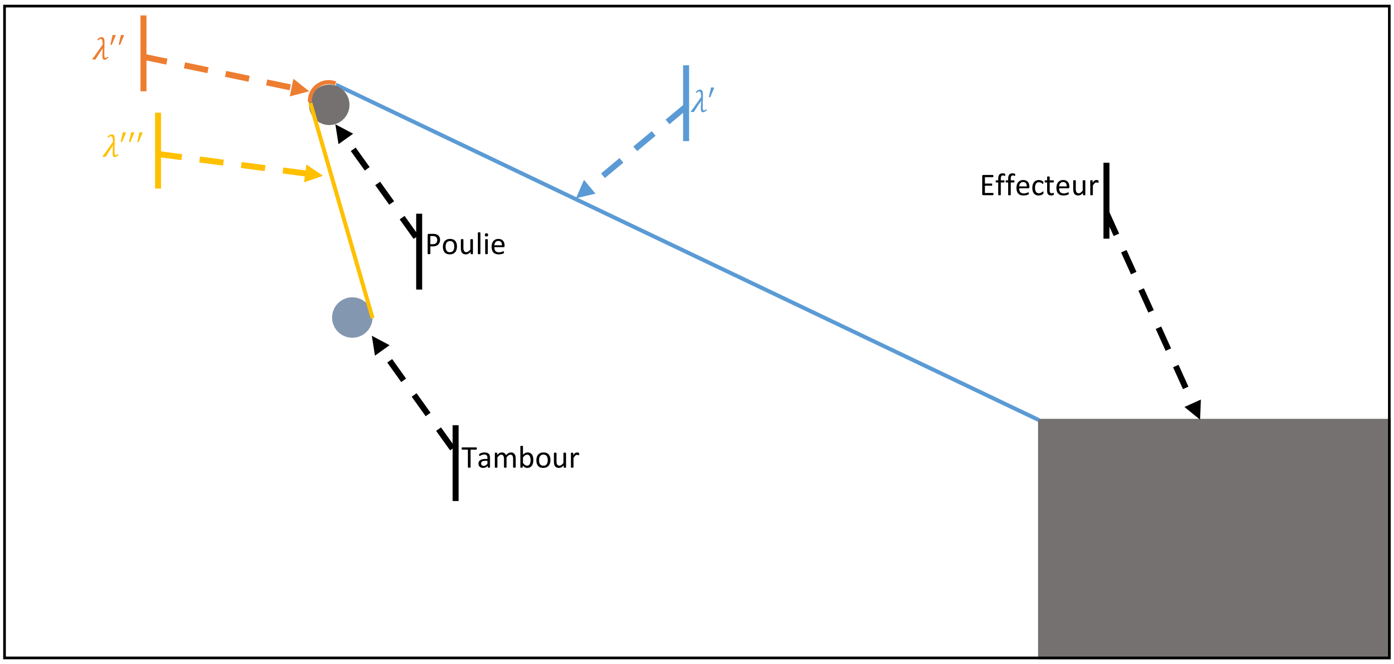


Figure 8: Schéma de la complexification

Mise en équations :

En schématisant notre problème et en distinguant les différents λ, on a le schéma suivant :

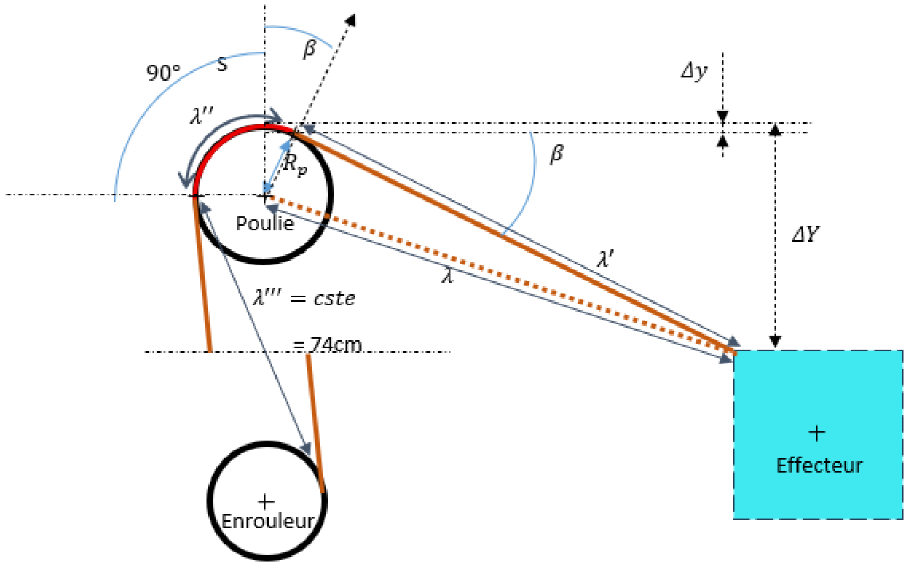


Figure 11: Schéma prise en compte de la longueur totale de câble

On connait déjà λ (calculé dans les parties ci-dessus), on peut en déduire λ'. On peut alors déterminer l'expression de β dans le but d'avoir ensuite la longueur de câble enroulée autour de la poulie (notée λ''). De plus on considère que la longueur de câble entre l'enrouleur et la poulie est constante (à tension constante) et notée λ'''. Alors la longueur de câble totale est la somme λ'+ λ''+ λ'''.

$$\textbf{(18)}: \lambda' = \sqrt{\lambda^2 + {R_p}^2}$$

$$\textbf{(19)}: \beta_{2,4} = \sin^{-1}\left(\frac{h_2 + R_p - Y_e - L/2}{\lambda'}\right) \text{ ou } \beta_{1,3} = \sin^{-1}\left(\frac{R_p - h_1 + Y_e - L/2}{\lambda'}\right)$$

$$\textbf{(20)}: \lambda'' = \left(\frac{\pi}{2} + \beta\right) * R_p$$

$$\textbf{(21)}: \lambda''' = \text{cste} = 74 \text{ cm}$$

$$\textbf{(22)}: \lambda_{tot} = \lambda' + \lambda'' + \lambda'''$$

L'expression de **(19) viens d'approximations des petits angles et de longueurs négligeables (dans le cadre du calcule de cet angle). On estime l'erreur d'angle a quelques °.**

En conclusion, l'hypothèse :

- **Les câbles sont accrochés à d'une extrémité au centre des poulies et de l'autre, au point d'attache de l'effecteur.**

Devient :

- **La longueur sur l'axe y entre un point d'accroche de l'effecteur et le centre de la poulie et plus grande que le rayon de la poulie.**
  $\Delta y \ll \Delta Y$
  $\beta * R_p \ll \lambda'$

Cette hypothèse en plus d'être plus proche de notre modèle réel est vrai dans la plupart des cas.

Remarque :

Ces approximations marchent pour des positions où l'effecteur n'est pas trop proche des poulies (notamment selon l'axe $\vec{X}$).

Implémentation dans le code de la simulation :

On rajoute ces équations dans notre code de simulation :

```
def rectification_lambda(lambda_1, lambda_2, lambda_3, lambda_4, Y_e):
    """

    Parameters
    ----------
    lambda_1 : {float} longueur de câble 1 sans complexification du modèle. Distance entre le centre de la poulie 1 et le
pooint d'attache 1 sur l'effecteur.
    lambda_2 : {float} longueur de câble 2 sans complexification du modèle. Distance entre le centre de la poulie 2 et le
pooint d'attache 2 sur l'effecteur.
    lambda_3 : {float} longueur de câble 3 sans complexification du modèle. Distance entre le centre de la poulie 3 et le
pooint d'attache 3 sur l'effecteur.
    lambda_4 : {float} longueur de câble 4 sans complexification du modèle. Distance entre le centre de la poulie 4 et le
pooint d'attache 4 sur l'effecteur.

    Returns
    -------
    lambda_tot_1 : {float} longueur de câble total depuis le tambour 1 jusqu'au point d'attache 1 sur l'effecteur
    lambda_tot_2 : {float} longueur de câble total depuis le tambour 2 jusqu'au point d'attache 2 sur l'effecteur
    lambda_tot_3 : {float} longueur de câble total depuis le tambour 3 jusqu'au point d'attache 3 sur l'effecteur
    lambda_tot_4 : {float} longueur de câble total depuis le tambour 4 jusqu'au point d'attache 4 sur l'effecteur

    """
    # lambda_prime_i : correction sur la longueur de câble entre la poulie i et le point d'accroche i sur l'effecteur
    lambda_prime_1 = sqrt(float(lambda_1)**2 + r_p**2)
    lambda_prime_2 = sqrt(float(lambda_2)**2 + r_p**2)
    lambda_prime_3 = sqrt(float(lambda_3)**2 + r_p**2)
    lambda_prime_4 = sqrt(float(lambda_4)**2 + r_p**2)

    # Clamp pour éviter les erreurs de domaine de asin
    def safe_asin(x):
        return math.asin(min(1.0, max(-1.0, x)))

    # beta_i : angle entre le sommet de la poulie i et le point tangent entre la poulie i et le câble i
    arg_beta_1 = (Y_e - h_1 - L/2 + r_p) / lambda_prime_1
    arg_beta_2 = (h_2 - Y_e - L/2 + r_p) / lambda_prime_2
    arg_beta_3 = (Y_e - h_1 - L/2 + r_p) / lambda_prime_3
    arg_beta_4 = (h_2 - Y_e - L/2 + r_p) / lambda_prime_4

    beta_1 = safe_asin(arg_beta_1)
    beta_2 = safe_asin(arg_beta_2)
    beta_3 = safe_asin(arg_beta_3)
    beta_4 = safe_asin(arg_beta_4)

    # lambda_sec_i : longueur de câble directement enroulé sur la poulie i
    lambda_sec_1 = (beta_1 + math.pi / 2) * r_p
    lambda_sec_2 = (beta_2 + math.pi / 2) * r_p
    lambda_sec_3 = (beta_3 + math.pi / 2) * r_p
    lambda_sec_4 = (beta_4 + math.pi / 2) * r_p

    # lambda_tot_i : longueur de câble total depuis le tambour jusqu'au point d'attache sur l'effecteur
    lambda_tot_1 = lambda_prime_1 + lambda_sec_1 + lambda_troisieme
    lambda_tot_2 = lambda_prime_2 + lambda_sec_2 + lambda_troisieme
    lambda_tot_3 = lambda_prime_3 + lambda_sec_3 + lambda_troisieme
    lambda_tot_4 = lambda_prime_4 + lambda_sec_4 + lambda_troisieme

    return lambda_tot_1, lambda_tot_2, lambda_tot_3, lambda_tot_4
```

Puis un plot pour l'affichages des longueurs totales des câbles :

```
    # Tracé des longueurs totales de câble
    fig_var, axs_var = plt.subplots(nrows=2, ncols=2)
    fig_var.suptitle("Longueurs totales des câbles")
    l_traj_vec = np.array(l_traj)
    D1, D2, D3, D4 = l_traj_vec[:,0], l_traj_vec[:,1], l_traj_vec[:,2], l_traj_vec[:,3]

    lambda_tot_1, lambda_tot_2, lambda_tot_3, lambda_tot_4  = [], [], [], []
    for l_tot_1, l_tot_2, l_tot_3, l_tot_4, Ye in zip(D1, D2, D3, D4, Y_traj):
        l1, l2, l3, l4 = rectification_lambda(l_tot_1, l_tot_2, l_tot_3, l_tot_4, Ye)
        lambda_tot_1.append(l1)
        lambda_tot_2.append(l2)
        lambda_tot_3.append(l3)
        lambda_tot_4.append(l4)

    for ax, D, title, color in zip(axs_var.flat, [lambda_tot_4, lambda_tot_2, lambda_tot_3, lambda_tot_1], ["Câble 4",
"Câble 2", "Câble 3", "Câble 1"], ["red", "green", "blue", "purple"]):
        Etape = np.linspace(0, nb_points, np.shape(D1)[0])
        ax.plot(Etape, D, marker='o', color=color)
        ax.set_title(title)
        ax.set_xlabel("Itération")
        ax.set_ylabel("Longueur totale de câble [mm]")
        ax.grid()
```

Modèle cinématique

Calcul Jacobienne et modèle cinématique inverse

On pose notre vecteur des sorties ($\vec{X}$) et notre vecteur des entrées ($\vec{L}$), on a :

- $\vec{X} = (X_e \; Y_e \; \phi_1)^T$
- $\vec{Q} = (q_1 \; q_2 \; q_3 \; q_4)^T$

Le modèle géométrique inverse de notre système est donné par :

$\vec{Q} = f(\vec{X})$

On notera q_i , $i \in [1, 4]$ les composantes du vecteur $\vec{Q}$ et x_e , $e \in [1, 2]$ les composantes du vecteur $\vec{X}$.

La fonction f, permet de calculer les entrées de notre système $(q_1 \; q_2 \; q_3 \; q_4)$, à partir des variables de sorties de notre système $(X_e \; Y_e \; \phi_1)$ Cette fonction vectorielle découle des équations (13), (14), (15) et (16) présentés précédemment.

L'équation **(17)**, nous donne :

$\dot{Q}_1 = \dot{R}^{-1} * (\Lambda_1 - \Lambda_0) + R^{-1} * \dot{\Lambda}_1,$

Or les r_i sont des valeurs constantes, on a donc :

(23)	$: \dot{Q}_1 = R^{-1} * \dot{\Lambda}_1$

Or, la relation entre $\dot{\Lambda}_1$ *et* $\dot{X}_1$, par définition, s'écrit :

(24)	$: \dot{\Lambda}_1 = J(x) * \dot{X}_1$

On en déduit :

(25)	$: \dot{Q}_1 = R^{-1} * J(x) * \dot{X}_1$

On rappelle que la Jacobienne du modèle cinématique inverse est donc donnée par :

(26)	$: \dot{q}_\iota = \frac{\partial q_i}{\partial x_e} * \dot{x}_{e_1} = J_i * \dot{x}_{e_1}$

Avec les $\dot{q}_\iota$, les composantes du vecteur $\dot{Q}$ et les x_{e_1}, les composantes du vecteur $\dot{X_{e_1}}$

Par dérivation, on en déduit :

$$
j_\iota = \begin{bmatrix} \frac{2X_e - 2X_i + 2a_i\cos(\phi_1) - 2b_i\sin(\phi_1)}{2l_i} \\ \frac{2Y_e - 2Y_i + 2b_i\cos(\phi_1) + 2a_i\sin(\phi_1)}{2l_i} \\ \frac{-2(b_i\cos(\phi_1) + a_i\sin(\phi_1))(X_e - X_i + a_i\cos(\phi_1) - b_i\sin(\phi_1))) - 2(a_i\cos(\phi_1) - b_i\sin(\phi_1))(Y_e - Y_i + b_i\cos(\phi_1) + a_i\sin(\phi_1))}{2l_i} \end{bmatrix}
$$

Les X_i et Y_i sont les coordonnées du point de la poulie i (les points O_{p_i}). Et les a_i et b_i sont les coordonnées du point d'accroche i sur l'effecteur par rapport au centre de l'effecteur (les points O_{c_i}). On a donc :

i =	$\boldsymbol{X_i}$	$\boldsymbol{Y_i}$	$\boldsymbol{a_i}$	$\boldsymbol{b_i}$
1	l_1	h_1	$\frac{l}{2}$	$-\frac{L}{2}$
2	l_1	h_2	$\frac{l}{2}$	$\frac{L}{2}$
3	0	h_1	$-\frac{l}{2}$	$-\frac{L}{2}$
4	0	h_2	$-\frac{l}{2}$	$\frac{L}{2}$

Tableau 5 : Tableau d'assignation des paramètres X_i, Y_i, a_i et b_i

Modèle cinématique directe

Pour le modèle cinématique directe, on va utiliser la pseudo inverse de la Jacobienne précédemment présenté et calculé, car la Jacobienne étant de taille 3x4, nous ne pouvons pas calculer son inverse.

Rappelons le calcul d'une pseudo inverse noté J^+, pour une matrice Jacobienne noté J.

On a : $J^+ = (J^\tau * J)^{-1} * J^\tau$

À partir de l'équation du modèle inverse on a alors :

$$\dot{Q}_1 = R^{-1} * J(x) * \dot{X}_1$$

⇨ $\dot{X}_1 = J^+(x) * R * \dot{Q}_1$

Modèle dynamique

Formalisme des écritures :

Cette partie se fonde grandement sur la thèse de M. Johann Lamaury en s'adaptant aux conditions et hypothèses de notre montage.

À noter que l'on se place dans un premier temps dans le cas où les centres géométriques et massiques sont confondus (plaque à vide).

On écrit alors notre relation dynamique de la manière suivante (Newton-Euler) :

$$\begin{pmatrix} \sum \boldsymbol{f} \\ \sum \boldsymbol{\tau} \end{pmatrix} = \boldsymbol{M}(\dot{\boldsymbol{x}})\ddot{\boldsymbol{x}} + \boldsymbol{C}(\boldsymbol{x},\dot{\boldsymbol{x}})\dot{\boldsymbol{x}}$$

- **f** est le vecteur des forces extérieurs sur la plaque (avec des coordonnées en $\overrightarrow{x_b}$ et $\overrightarrow{y_b}$)
- **τ** est un vecteur 1 x 1 contenant le moment autour de l'axe $\overrightarrow{z_b}$ auquel la plaque est soumise.
- **M** est la matrice « d'inertie » ou « de masse » de taille n x n (avec n le nombre de DDL = 3)
- **C'**est la matrice n x n des efforts de Coriolis et centrifuges du corps

Expressions :

Expressions de **M** et **C**:

- $\boldsymbol{M}(\boldsymbol{x}) = \begin{bmatrix} \boldsymbol{m_{tot}} & \boldsymbol{O_{p*q}} \\ \boldsymbol{0_{q*p}} & \boldsymbol{K} \end{bmatrix}$

Avec **p** le nombre de DDL en translation (2 DDL), **q** le nombre de DDL en rotation, **mtot** la masse de l'effecteur + charge et $\boldsymbol{K} = \boldsymbol{Q} * \boldsymbol{I} * \boldsymbol{Q}^{\boldsymbol{T}}$ ou **Q** est la matrice de rotation du repère effecteur au repère global et **I** la matrice d'inertie.

N.B. : Dans le cas où la rotation de l'effecteur est nulle, **Q** est l'identité 3 x 3.

- $\boldsymbol{C}(\boldsymbol{x},\dot{\boldsymbol{x}})\dot{\boldsymbol{x}} = \begin{bmatrix} \boldsymbol{0_{nx1}} \\ \boldsymbol{\omega}.\boldsymbol{K}.\boldsymbol{\omega} \end{bmatrix}$

Où **ω** est le vecteur vitesse angulaire.

N.B. : dans notre cas **C est nulle** car **ω** est nulle.

On a finalement :

$$\boldsymbol{M}(\boldsymbol{x}).\ddot{\boldsymbol{x}} + \boldsymbol{C}(\boldsymbol{x},\dot{\boldsymbol{x}}).\dot{\boldsymbol{x}} = -\boldsymbol{J}^{\boldsymbol{T}}.\boldsymbol{t} + \boldsymbol{g}(\boldsymbol{x})$$

Ou dans notre cas :

$$\boxed{\begin{bmatrix} m_{tot} & 0 & 0 \\ 0 & m_{tot} & 0 \\ 0 & 0 & I_{zz} \end{bmatrix} .\ddot{x} = -\boldsymbol{J^T}.\boldsymbol{t} + \begin{bmatrix} 0 \\ -m_{tot} * g \\ 0 \end{bmatrix} \quad \textbf{(27)}}$$

On prend ici **t** le vecteur 4 x 1 des tensions de câbles. Ainsi en le multipliant par la transposée de la jacobienne on obtient le vecteur des efforts entre l'effecteur et les câbles.

Dans un second temps on considère (conformément à la thèse de M. Johann Lamaury) que la dynamique des blocs moteurs +enrouleurs s'écrit de la manière suivante :

$$\boxed{\boldsymbol{\tau_m} = \boldsymbol{I_m}.\ddot{\boldsymbol{q}} + \boldsymbol{F_f}(\dot{\boldsymbol{q}}) + \boldsymbol{R}.\boldsymbol{t} \quad \textbf{(28)}}$$

Avec $\boldsymbol{\tau_m}$ le vecteur des couples moteurs, $\boldsymbol{I_m}$ la matrice d'inertie des moteurs (inconnue), $\dot{\boldsymbol{q}}$ et $\ddot{\boldsymbol{q}}$ respectivement les vitesses et accélération articulaires des moteurs, $\boldsymbol{F_f}$ les couples de frottement des câbles et $\boldsymbol{R}$ la matrice diagonale des coefficients notés ri dans la thèse. Ce coefficient fait office de rapport de réduction entre le moteur et l'enrouleur (prise en compte du pas de l'enrouleur et de son rayon).

Cette équation **(28)** peut être utilisée en négligeant les frottements que nous n'avons pas pu acquérir en prenant comme inertie des moteurs $I = 1{,}5.10^{-2}\ kg.m^2$ (recherche internet).

Alors en substituant **t** dans **(27)** par son expression selon **(28)** :

$$\boxed{\boldsymbol{M}(\boldsymbol{x}).\ddot{\boldsymbol{x}} + \boldsymbol{C}(\boldsymbol{x},\dot{\boldsymbol{x}}).\dot{\boldsymbol{x}} = -\boldsymbol{J^T}\boldsymbol{R}^{-1}(\boldsymbol{\tau_m} - \boldsymbol{I_m}.\ddot{\boldsymbol{q}}) + \boldsymbol{g}(\boldsymbol{x}) \quad \textbf{(29)}}$$

Cette équation peut être exprimée selon **x** (position de l'effecteur) ou **q** (positions des moteurs) selon le modèle que l'on veut (inverse ou directe) en utilisant les expressions liant $\dot{\boldsymbol{x}}$ et $\dot{\boldsymbol{q}}$ ainsi que leurs dérivées :

$$\ddot{\mathbf{q}} = \mathbf{R}^{-1}\left(\mathbf{J}\ddot{\mathbf{x}} + \dot{\mathbf{J}}\dot{\mathbf{x}}\right)$$
$$\ddot{\mathbf{x}} = \mathbf{J}^{+}\mathbf{R}\ddot{\mathbf{q}} + \dot{\mathbf{J}}^{+}\mathbf{R}\dot{\mathbf{q}}$$

Conclusion & améliorations possibles

Ce projet a surement été l'un des projets les plus formateurs que nous ayons pu faire durant notre scolarité. Ce projet a été formateur car bien que supervisé par M. M. Rebillat et M. P. Margerit, nous étions en grande partie en autonomie. Et les rares entretiens que nous avions avec M. M. Guskov, ne nous ont pas forcément aidé (plutôt le contraire, ce qui explique la rareté de ces entretiens…).

Nous étions donc armés de notre savoir, nos expérience, un sujet de thèse qui traite des robot à câbles et de Spyder. **Nous avions dû mettre des objectifs atteignable sur les tâches qui nous ont été demandé et des deadline sur ces objectif.** L'une des choses que nous avions sous-estimés était la charge de travail sur ce projet, nous pensions naïvement qu'avec le sujet de thèse et en implémentant les équation de fermetures géométriques dans Python, nous aurions un modèle qui marche. Comme vous avez pu le voir durant la lecture de ce rapport, ça n'a pas été si simple. Et nous avions dû nous redoubler d'effort et nous creuser la tête encore plus pour imaginer un nouveau modèle qui serait plus fiable et qui marcherait, le modèle 2.

Cette expérience qui nous a découragé dans un premier temps et qui nous a demandé de redoubler d'effort pour trouver une nouvelle approche a aussi été, très formatrice. **Nous avons ensuite mis en place une démarche scientifique (choix du modèle, implantation, test du modèle et comparaison avec le système réel, conclusion et complexification du modèle).**

L'un des choses que l'on n'a pas développé dans ce rapport et le choix du langage de programmation, pourquoi avoir choisi Python ? La réponse vient aussi de notre expérience durant ce projet, il se trouve qu'un groupe d' élèves des années d'avant, avait déjà fait un modèle géométrique de ce robot. Mais ce modèle avait été implémenté sous MatLab et ne marchait pas très bien. Nous avions alors choisi de faire ce modèle sous Python car les personnes qui poursuivront notre études seront surement dans le même cas que nous, c'est-à-dire beaucoup plus à l'aise avec Python qu'avec MatLab.

Cette expérience explique aussi pourquoi nos codes sous Python sont ultra commenté et (avec des retours utilisateurs, des notes sur le code et des docstring), nous avons aussi essayé de rendre notre code le plus lisible possible. **Vous noterez aussi que ce document fait office de README.txt, en effet vous y trouverez tout notre raisonnement, notre travail, nos hypothèses, résultats, ainsi que comment utiliser les codes de simulation.**

Malheureusement, nous sommes forcés d'admettre que bien que fier de notre travail, il n'est pas fini et des pistes d'améliorations sont possibles, les voici :

<u>Modèle en 3D :</u> Vous l'avez remarqué le modèle 2 que nous vous avons proposé est un modèle où tout se passe dans un plan XY, alors que le robot est censé évolué dans un environnement en 3D. Voici la première piste d'amélioration, augmenter le modèle 2D du modèle 2 pour le passer en modèle 3D. Et, étant donné que le modèle 2 est basé sur du suivi de points, le passage en 3D ne devrait pas être très compliqué. Nous ne l'avions pas fait par manque de temps.

<u>Consigne de vitesse et test modèle cinématique :</u> Pour le modèle cinématique, nous avion implanté une vitesse V_min, cette vitesse nous permet de choisir une vitesse minimale pour l'effecteur. Dans notre cas, cette vitesse est constante, mais l'on peut imaginer que le robot à câble suit une loi de vitesse (loi trapézoïdale ou une commande en vitesse), que l'on pourrait faire varier en fonction des différentes phases de vitesse de notre système. C’est une piste d’amélioration possible qu’il faudrait surement creuser avec l’équipe qui s’occupera de la partie sur l’automatisation.

Dynamique du robot : Les équations du modèle dynamiques vous ont été présenté dans ce rapport, nous n’avions pas pu les mettre en pratique par manque de temps et de ressources (à ce jour, nous n’avons pas d’ordre de grandeur réelle dans la tension des câbles). Lorsque les capteurs de tension seront installés et fonctionnel, il serait surement intéressant de prendre des mesures avec pour faire le modèle dynamique du robot.

Merci,

Gwezheneg RIVIERE & Ngatam Thiébaut

Annexes :

Fiche de PJE

Projet 2024-2025

Modélisation d'un robot parallèle à cables et estimation de paramètres

Réf.: PA-F25045 Centre: Paris Directeur: Mikhail GUSKOV

Le projet se déroulera en: INCONNU

Etudiants demandés: 2 Type: PJE9

Expertise imposée: Mécatronique

Projet CAMIPable: NON

Thème: Génie mécanique

Sciences et techniques: Modélisation, Simulation,

Secteurs industriels, Services:

Contexte:	Les robots parallèles à câbles sont des robots avec peu d'inertie et permettant des mouvements rapides et précis. L'objectif est de finir la conception et la mise en route d'un robot parallèle à câbles sur le Campus de Paris.
Description:	Les robots parallèles à câbles sont des robots avec peu d'inertie et permettant des mouvements rapides et précis. L'objectif est de finir la conception et la mise en route d'un robot parallèle à câbles sur le Campus de Paris.
Prérequis:	- Intérêt pour la modélisation mécanique en mécatronique - Intérêt pour les procédure d'estimation de paramètres
Acquis:	- Modélisation d'un robot parallèle à câbles - Estimation de paramètres
Tâches:	- Mesures & prises de côtes sur le robot existant - Définition d'un modèle et implémentation - Comparaison essais/calculs

Explication du code du modèle 1 :

Importation des différents modules

```
################################## Import ########################################

import sympy
import scipy

import numpy as np
import matplotlib.pyplot as plt

from numpy import linalg
from sympy import sin, cos, sqrt, Matrix
from mpl_toolkits.mplot3d import Axes3D
import matplotlib.animation as animation
from sympy.abc import alpha, beta, phi, delta, theta, psi, omega
```

Paramètres du systèmes

```
############################## Paramètres du système ###########################

## Paramètres du système
h_1 = 400 # (en mm) hauteur poulie 1
h_2 = 2180 # (en mm) hauteur poulie 2
l = 230 # (en mm) largeur de la plaque de l'effecteur
L = 230 # (en mm) longueur de la plaque de l'effecteur
l_1 = 1900 # (en mm) distance entre 2 poulie de la structure
K = 0.5 # rapport de transmission de l'enrouleur
e = 30 # (en mm) rayon de l'enrouleur
rho = 5 # (en mm) pas de l'enrouleur
pas_mot = 1.8 # (en °) pas du moteur => 200 pas pour 1 tour

# Positions des poulies (points fixes) dans le plan
poulies = np.array([
    [l_1, h_1],  # P1 (en bas à droite)
    [l_1, h_2],  # P2 (en haut à droite)
    [0, h_1],    # P3 (en bas à gauche)
    [0, h_2]     # P4 (en haut à gauche)
])

# Coordonnées des points d'attache de la plaque (dans son repère local)
v_attache = np.array([
    [ l/2, -L/2],  # Coin bas droite
    [ l/2,  L/2],  # Coin haut droite
    [-l/2, -L/2],  # Coin bas gauche
    [-l/2,  L/2]   # Coin haut gauche
])

# Position intiale de l'effecteur - au centre du repère
X_0 = 1075  # Position initiale du centre de la plaque en X
Y_0 = 1090  # Position initiale du centre de la plaque en Y

## Modèle inverse - variables - position de l'effecteur
X_e = sympy.symbols("X_e") # position de l'effecteur sur l'axe X de la structure
Y_e = sympy.symbols("Y_e") # position de l'effecteur sur l'axe Y de la structure
phi_1 = sympy.symbols("phi_1") # position angulaire de l'effecteur par rapport au repère de la base
```

Équations du système

```
############################## Equations du système ############################

## Coefficients pour le calcul
r = K * (e**2 + (rho**2)/2*np.pi)**1/2 # coefficient d'enroulement des enrouleurs
X = [l_1, l_1, 0, 0] # position sur l'axe x_base des poulies
Y = [h_1, h_2, h_1, h_2] # position sur l'axe y_base des poulies
a = [l/2, l/2, -l/2, -l/2] # position sur l'axe x_effecteur des points d'accroche
b = [-L/2, L/2, -L/2, L/2] # position sur l'axe y_effecteur des points d'accroche

## Modèle inverse - équations modèle analytique
lambda_1 =  sqrt((X_0 + X_e - X[0] + a[0]*cos(phi_1) - b[0]*sin(phi_1))**2 + (Y_0 + Y_e - Y[0] + a[0]*sin(phi_1) +
b[0]*cos(phi_1))**2) # longueur du câble de la poulie 1 (en mm)
lambda_2 =  sqrt((X_0 + X_e - X[1] + a[1]*cos(phi_1) - b[1]*sin(phi_1))**2 + (Y_0 + Y_e - Y[1] + a[1]*sin(phi_1) +
b[1]*cos(phi_1))**2) # longueur du câble de la poulie 2 (en mm)
lambda_3 =  sqrt((X_0 + X_e - X[2] + a[2]*cos(phi_1) - b[2]*sin(phi_1))**2 + (Y_0 + Y_e - Y[2] + a[2]*sin(phi_1) +
b[2]*cos(phi_1))**2) # longueur du câble de la poulie 3 (en mm)
lambda_4 =  sqrt((X_0 + X_e - X[3] + a[3]*cos(phi_1) - b[3]*sin(phi_1))**2 + (Y_0 + Y_e - Y[3] + a[3]*sin(phi_1) +
b[3]*cos(phi_1))**2) # longueur du câble de la poulie 4 (en mm)
# ---
q_1 = lambda_1 / r # angle de rotation du moteur 1 (en rad)
q_2 = lambda_2 / r # angle de rotation du moteur 2 (en rad)
q_3 = lambda_3 / r # angle de rotation du moteur 3 (en rad)
q_4 = lambda_4 / r # angle de rotation du moteur 4 (en rad)
# ---
p_1 = (200*q_1) / 2*np.pi # nombre de pas sur le moteur 1
p_2 = (200*q_2) / 2*np.pi # nombre de pas sur le moteur 2
p_3 = (200*q_3) / 2*np.pi # nombre de pas sur le moteur 3
p_4 = (200*q_4) / 2*np.pi # nombre de pas sur le moteur 4
```

Ces équations ont été trouvé en faisant les fermetures géométriques de notre système décrites au début de ce rapport.

Fonctions subsidiaires

Calcul les vitesses et longueurs de câble pour un déplacement de l'effecteur en X_e, Y_e et ϕ_1 sur une durée "temps"

```
def inverse_test(X_e_exp, Y_e_exp, phi_1_exp, temps):
    """

    Parameters
    ----------
    X_e_exp : {float} Coordonnée finale de l'effecteur sur l'axe x
    Y_e_exp : {float} Coordonnée finale de l'effecteur sur l'axe y
    phi_1_exp : {float} Coordonnée finale la rotation de l'effecteur autour de l'axe z
    temps : {int} temps de la simulation

    Returns
    -------
    V_lambda_exp : {array} Vecteur des vitesses linéaires des câbles
    D_lambda_exp : {array} Vecteur des longueurs de câbles

    """
    X_vars = Matrix([X_e, Y_e, phi_1])
    Y_out = Matrix([lambda_1, lambda_2, lambda_3, lambda_4])
    J = Y_out.jacobian(X_vars)
    J = J.subs({X_e: X_e_exp, Y_e: Y_e_exp, phi_1: phi_1_exp})

    V_X = X_vars.subs({X_e: X_e_exp/temps, Y_e: Y_e_exp/temps, phi_1: phi_1_exp/temps})
    V_lambda_exp = np.dot(J, V_X)
    D_lambda_exp = V_lambda_exp * temps

    longueur_initial = 1320
    for i in range(4):
        D_lambda_exp[i][0] += longueur_initial

    return V_lambda_exp, D_lambda_ex
```

Crée une animation 3D du mouvement de la plaque à partir des longueurs de câbles

```
def animation_test(longueur_1, longueur_2, longueur_3, longueur_4):
    """

    Parameters
    ----------
    longueur_1 : {float} Longueur courante du câble 1
    longueur_2 : {float} Longueur courante du câble 2
    longueur_3 : {float} Longueur courante du câble 3
    longueur_4 : {float} Longueur courante du câble 4

    Returns
    -------
    ani : animation à l'instant courant

    """
    cable_lengths = {
        "Câble 1": longueur_1,
        "Câble 2": longueur_2,
        "Câble 3": longueur_3,
        "Câble 4": longueur_4,
    }
```

```
num_frames = len(longueur_1)
plaque_size = 230

fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
ax.set_xlim(0, 2150)
ax.set_ylim(0, 2150)
ax.set_zlim(0, 500)
ax.set_title("Déplacement de la plaque dans le plan XY")
ax.set_xlabel("X")
ax.set_ylabel("Y")
ax.set_zlabel("Z")

anchor_points = {
    "Câble 1": [2150, 0, 0],
    "Câble 2": [2150, 2150, 0],
    "Câble 3": [0, 0, 0],
    "Câble 4": [0, 2150, 0],
}

lines = []
texts = []
plaque = None

def update(frame):
    nonlocal lines, texts, plaque
    for l in lines: l.remove()
    for t in texts: t.remove()
    if plaque: plaque.remove()

    lines.clear()
    texts.clear()

    l1 = cable_lengths["Câble 1"][frame]
    l2 = cable_lengths["Câble 2"][frame]
    l3 = cable_lengths["Câble 3"][frame]
    l4 = cable_lengths["Câble 4"][frame]

    dx = (l2 + l4 - l1 - l3) * 0.25
    dy = (l1 + l2 - l3 - l4) * 0.25
    cx = X_0 + dx
    cy = Y_0 + dy
    cz = 0

    half = plaque_size / 2
    p1 = [cx + half, cy - half, cz]
    p2 = [cx + half, cy + half, cz]
    p3 = [cx - half, cy - half, cz]
    p4 = [cx - half, cy + half, cz]

    for name, anchor, corner in zip(
        ["Câble 1", "Câble 2", "Câble 3", "Câble 4"],
        [anchor_points["Câble 1"], anchor_points["Câble 2"], anchor_points["Câble 3"], anchor_points["Câble 4"]],
        [p1, p2, p3, p4],
    ):
        line = ax.plot([anchor[0], corner[0]], [anchor[1], corner[1]], [anchor[2], corner[2]], 'gray')[0]
        lines.append(line)

        mid = [(anchor[i] + corner[i]) / 2 for i in range(3)]
        text = ax.text(mid[0], mid[1], mid[2] + 10, name, color='black', fontsize=9, ha='center')
        texts.append(text)

    X = np.array([[p1[0], p2[0]], [p3[0], p4[0]]], dtype=float)
```

```
        Y = np.array([[p1[1], p2[1]], [p3[1], p4[1]]], dtype=float)
        Z = np.array([[cz, cz], [cz, cz]], dtype=float)

        plaque = ax.plot_surface(X, Y, Z, color='skyblue', alpha=0.8)
        return lines + [plaque] + texts

    ani = animation.FuncAnimation(fig, update, frames=num_frames, interval=300, blit=False)
    print("\n Animation lancée !")
    plt.show()
    return ani
```

Fonction principale

```
# Simule le déplacement de l’effecteur avec un mouvement linéaire + rotation
# Affiche les courbes des longueurs et vitesses de câbles, animation 3D
# et une estimation de trajectoire du centre de la plaque
def inverse_plot(X_e_final, Y_e_final, phi_1_final, nombre_etape, temps):
    """

    Parameters
    ----------
    X_e_final : {float} Coordonnée finale de l'effecteur sur l'axe x
    Y_e_final : {float} Coordonnée finale de l'effecteur sur l'axe y
    phi_1_final : {float} Coordonnée finale la rotation de l'effecteur autour de l'axe z
    nombre_etape : {int} Nombre d'étape de la simulation
    temps : {int} Temps de la simulation

    Returns
    -------
    None.

    """
    X1 = np.linspace(X_0, X_e_final, nombre_etape)
    X2 = np.linspace(Y_0, Y_e_final, nombre_etape)
    PHI = np.linspace(0, phi_1_final, nombre_etape)
    Etape = np.arange(nombre_etape)

    D1, D2, D3, D4 = [], [], [], []
    V1, V2, V3, V4 = [], [], [], []

    for i, j, k in zip(X1, X2, PHI):
        vitesse, deplacement = inverse_test(i, j, k, temps)
        D1.append(deplacement[0][0])
        D2.append(deplacement[1][0])
        D3.append(deplacement[2][0])
        D4.append(deplacement[3][0])

        V1.append(vitesse[0][0])
        V2.append(vitesse[1][0])
        V3.append(vitesse[2][0])
        V4.append(vitesse[3][0])

    # Plot des longueurs
    fig_1, axs_1 = plt.subplots(2, 2)
    fig_1.suptitle("Tailles des câbles")
    for ax, D, title in zip(axs_1.flat, [D1, D2, D3, D4], ["Câble 1", "Câble 2", "Câble 3", "Câble 4"]):
        ax.plot(Etape, D)
        ax.set_title(title)
        ax.grid()

    # Plot des vitesses
    fig_2, axs_2 = plt.subplots(2, 2)
    fig_2.suptitle("Vitesses linéaires des câbles")
    for ax, V, title in zip(axs_2.flat, [V1, V2, V3, V4], ["Câble 1", "Câble 2", "Câble 3", "Câble 4"]):
        ax.plot(Etape, V)
        ax.set_title(title)
        ax.grid()

    global ani
    ani = animation_test(D1, D2, D3, D4)

    # Estimation de position
    CX, CY = [], []
    for l1, l2, l3, l4 in zip(D1, D2, D3, D4):
```

```
    dx = (l2 + l4 - l1 - l3) * 0.25
    dy = (l1 + l2 - l3 - l4) * 0.25
    cx = 1075 + dx
    cy = 1075 + dy
    CX.append(cx)
    CY.append(cy)

fig_pos, ax_pos = plt.subplots()
ax_pos.plot(CX, CY, label="Trajectoire estimée", marker='o')
ax_pos.set_title("Position estimée du centre de la plaque")
ax_pos.set_xlabel("X [mm]")
ax_pos.set_ylabel("Y [mm]")
ax_pos.grid()
ax_pos.legend()

plt.show()
```

Cette fonction va créer des vecteurs X1 et X2 d'une taille etape, ces vecteurs vont contenir les coordonnée en x et y de l'effecteur à chaque étape. On va aussi créer des vecteur pour leurs longueurs à chaque étapes (vecteurs D) et des vecteurs pour leurs vitesses linéaires des câbles (vecteurs V).

```
    """
    X1 = np.linspace(X_0, X_e_final, nombre_etape)
    X2 = np.linspace(Y_0, Y_e_final, nombre_etape)
    PHI = np.linspace(0, phi_1_final, nombre_etape)
    Etape = np.arange(nombre_etape)

    D1, D2, D3, D4 = [], [], [], []
    V1, V2, V3, V4 = [], [], [], []
```

Une fois ces vecteurs créé, ils vont nous servir pour avoir les longueurs des câbles à chaque étape, grâce au modèle inverse implémenté dans la fonction inverse_test.

```
    for i, j, k in zip(X1, X2, PHI):
        vitesse, deplacement = inverse_test(i, j, k, temps)
        D1.append(deplacement[0][0])
        D2.append(deplacement[1][0])
        D3.append(deplacement[2][0])
        D4.append(deplacement[3][0])

        V1.append(vitesse[0][0])
        V2.append(vitesse[1][0])
        V3.append(vitesse[2][0])
        V4.append(vitesse[3][0])
```

Puis, on va faire les plot des longueurs de câbles, des vitesses linéaires, de la trajectoire et faire l'animation. Cette partie n'est pas détaillé ici car manque d'intérêt.

Explication du code du modèle 2 :

Importation des différents modules :

```
################################## Import #######################################

import sympy

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import matplotlib.animation as animation

from numpy.linalg import pinv
from sympy import sin, cos, sqrt, Matrix
from matplotlib.animation import FuncAnimation
from sympy.abc import alpha, beta, delta, theta, psi, omega
```

Paramètres du système :

```
############################## Paramètres du système ###########################

## Paramètres du système
h_1 = 400 # (en mm) hauteur poulie 1
h_2 = 2180 # (en mm) hauteur poulie 2
l = 230 # (en mm) largeur de la plaque de l'effecteur
L = 230 # (en mm) longueur de la plaque de l'effecteur
l_1 = 1900 # (en mm) distance entre 2 poulie de la structure
K = 0.5 # rapport de transmission de l'enrouleur
e = 30 # (en mm) rayon de l'enrouleur
rho = 5 # (en mm) pas de l'enrouleur
pas_mot = 1.8 # (en °) pas du moteur => 200 pas pour 1 tour
r_p = 40 #(en mm) rayon interne des poulies
lambda_troisieme = 740 # (en mm) longueur de cable supposée constante entre l'enrouleur et la poulie (pas de
centre a centre)

## Coefficients pour le calcul
r = K * (e**2 + (rho**2)/2*np.pi)**1/2 # coefficient d'enroulement des enrouleurs
X = [l_1, l_1, 0, 0] # position sur l'axe x_base des poulies
Y = [h_1, h_2, h_1, h_2] # position sur l'axe y_base des poulies
a = [l/2, l/2, -l/2, -l/2] # position sur l'axe x_effecteur des points d'accroche
b = [-L/2, L/2, -L/2, L/2] # position sur l'axe y_effecteur des points d'accroche

## Positions des poulies (points fixes) dans le plan
poulies = np.array([
    [l_1, h_1],  # P1 (en bas à droite)
    [l_1, h_2],  # P2 (en haut à droite)
    [0, h_1],    # P3 (en bas à gauche)
    [0, h_2]     # P4 (en haut à gauche)
])

## Coordonnées des points d'attache de la plaque (dans son repère local)
v_attache = np.array([
    [ l/2, -L/2],  # Coin bas droite
    [ l/2,  L/2],  # Coin haut droite
    [-l/2, -L/2],  # Coin bas gauche
    [-l/2,  L/2]   # Coin haut gauche
])

## Position intiale de l'effecteur - au centre du repère
X_0 = 1010  # Position initiale du centre de la plaque en X
Y_0 = 1075  # Position initiale du centre de la plaque en Y
```

Fonctions subsidiaires :

Matrice de Rotation 2D pour un angle ϕ_1 :

```
def rotation_matrix(phi):
    """

    Parameters
    ----------
    phi : {float} Angle de rotation autour de l'axe z en °

    Returns
    -------
    R : {array} Matrice de rotation en 2D autour de l'axe z

    """
    R = np.array([
        [np.cos(phi), -np.sin(phi)],
        [np.sin(phi),  np.cos(phi)]
    ])

    return R
```

On retrouve en effet la matrice de rotation autour d'un axe z dans un plan 2D :

$$M = \begin{bmatrix} \cos(\phi) & sin(\phi) \\ sin(\phi) & cos(\phi) \end{bmatrix}$$

Calcul des coordonnées globales des points d'attache sur l'effecteur :

```
def compute_attachment_points(X, Y, phi):
    """

    Parameters
    ----------
    X : {float} Coordonnée sur x du point d'attache sur l'effecteur
    Y : {float} Coordonnée sur y du point d'attache sur l'effecteur
    phi : {float} Rotation en ° autour de l'axe z

    Returns
    -------
    M : {array} Coordonées dans le repère global des points d'attache de l'effecteur

    """
    R = rotation_matrix(phi)
    M = np.array([[X, Y]]) + (v_attache @ R.T)
    return M
```

Les coordonnées des points d'attache sont données par :

$$[x \quad y] + \begin{bmatrix} \frac{l}{2} & -\frac{L}{2} \\ \frac{l}{2} & \frac{L}{2} \\ -\frac{l}{2} & -\frac{L}{2} \\ -\frac{l}{2} & \frac{L}{2} \end{bmatrix} * \begin{bmatrix} \cos(\phi) & \sin(\phi) \\ \sin(\phi) & \cos(\phi) \end{bmatrix}^t$$

Calcul des longueurs des câbles:

```
def cable_lengths(X, Y, phi):
    """

    Parameters
    ----------
    X : {float} Coordonnée sur x du point d'attache sur l'effecteur
    Y : {float} Coordonnée sur y du point d'attache sur l'effecteur
    phi : {float} Rotation en ° autour de l'axe z

    Returns
    -------
    L : {float} Distance entre le point d'attache i sur l'effecteur et le centre de la poulie.

    """
    A = compute_attachment_points(X, Y, phi) # Calcul les coordonées des points d'attache de la plaque
    L = np.linalg.norm(A - poulies, axis=1) # Renvoie la norme de la matrice qui est la différence des coordonées des points
d'attache de la plaque actuellement et les coordonnées des poulies initialement
    return L
```

Veuillez noter que pour le calcul des longueurs de câbles, on va calculer la distance entre le point d'attache i sur l'effecteur et le centre poulie i. Ce nous permet d'avoir une approche intéressante mais pose les hypothèses suivantes :

- **Les câbles sont accrochés à d'une extrémité au centre des poulies et de l'autre, au point d'attache de l'effecteur.**
- **Les câbles sont toujours tendus**

Ces hypothèses bien qu'intéressantes seront utile dans une première approche mais seront revues une fois notre premier modèle vérifié.

Calcul de la Jacobienne $J = \frac{\partial \lambda}{\partial X}, X = [x, y, \phi_1]$ *(variation des longueurs de câbles par rapport aux coordonnées)*

```
def jacobian(X, Y, phi):
    """

    Parameters
    ----------
    X : {float} Coordonnée sur x du point d'attache sur l'effecteur
    Y : {float} Coordonnée sur y du point d'attache sur l'effecteur
    phi : {float} Rotation en ° autour de l'axe z

    Returns
    -------
    J : {array} Jacobienne d_rond lambda/ d_rond X (variation des longueurs des câbles par rapport à X, Y, phi)

    """
    A = compute_attachment_points(X, Y, phi)  # Calcul les coordonées des points d'attache de la plaque
    R = rotation_matrix(phi) # Matrice de rotation autour de l'axe z
    J = np.zeros((4, 3)) # Initialisation de la Jacobienne
    for i in range(4):
        diff = A[i] - poulies[i]  # Vecteur du point poulie vers point d'attache i
        d = np.linalg.norm(diff)  # Longueur du câble i
        if d == 0:
            continue  # Evite division par zéro
        dX = diff[0] / d  # Dérivée partielle par rapport à X
        dY = diff[1] / d  # Dérivée partielle par rapport à Y
        dphi_vec = R @ np.array([-v_attache[i][1], v_attache[i][0]])  # Rotation du vecteur d'attache
        dphi = np.dot(diff, dphi_vec) / d  # Dérivée partielle par rapport à phi
        J[i, :] = [dX, dY, dphi] # Remplissage de la Jacobienne sur la ligne i
    return J
```

Fonction principale :

```
def animation_2(X_inital, Y_initial, phi_1_initial, X_final, Y_final, phi_1_final, V_min, step, nb_points, epsilon):
    """
    Parameters
    ----------
    X_intial : {float} Coordonnées sur l'axe x finale du centre de l'effecteur initialement
    Y_initial : {float} Coordonnées sur l'axe y finale du centre de l'effecteur initialement
    phi_1_initial : {float} Angle de rotation finale autour de l'axe z du centre l'effecteur initialement

    X_final : {float} Coordonnées sur l'axe x finale du centre de l'effecteur
    Y_final : {float} Coordonnées sur l'axe y finale du centre de l'effecteur
    phi_1_final : {float} Angle de rotation finale autour de l'axe z du centre l'effecteur

    V_min : {float} Vitesse minimale de l'effecteur
    step : {float} Taille des pas de la simulation
    nb_points : {int} Nombre de points pour effectuer la simulation (en seconde)
    epsilon : {int} Valeur en % de la bande d'arrêt

    Returns
    -------
    Cette fonction fait la simulation du déplacement de l'effecteur du robot parallèle à câble de sa postion initale (le
centre du repère)
    à une position final de coordonée X_final, Y_final et avec un angle phi_1_final.
    Elle affiche différent plot:
      - le 1er est l'animation liée à cette simulation
      - Le 2nd est constitué de 4 subplot qui chacun affichent la longueurs des 4 câbles en fonction des itérations
      - Le 3ème affiche la position angulaire des moteurs
      - le 4ème affiche les pas des moteurs
      - le 5ème affiche la position du centre de l'effecteur
      - le 6ème affiche la rotation du centre de l'effecteur
      - Le 7ème affiche les vitesses linéaires des câbles
      - Le 8ème affiche les vitesse de rotation des moteur
      - Le 9ème affiche le longueurs totales des câbles

    Cette fonction créé aussi un document xls avec les données de la simulation
      """
    # --------------- Initialisation des paramètres --------------------------#

    # Initialisation à la position initiale
    X0, Y0 , phi_1_0 = X_inital, Y_initial, phi_1_initial
    print("\nPosition initiale: X0 = ", X0, "Y0 = ", Y0, "phi_1_0 = ", phi_1_0)
    X_traj, Y_traj, phi_traj = [X0], [Y0], [phi_1_0]

    # Initialisation des longueurs de câbles
    l_0 = cable_lengths(X0, Y0, 0.0)
    l_traj = [l_0]
    print("Longueurs des câbles initiales: ",phi_1_0)

    X, Y, phi_1 = X0, Y0, phi_1_0

    # Initialisation des vitesses
    v_traj = []


    # ----------------- Boucle de simulation ---------------------------------#

    for i in range(nb_points):
      dx = X_final - X
      dy = Y_final - Y
      dphi = phi_1_final - phi_1
      error = np.array([dx, dy, dphi])
      error_norm = np.linalg.norm(error)
```

```
        # Pour éviter de diviser par 0
        if error_norm < 1e-4:
            break

        # Direction normalisée de l'erreur
        direction = error / error_norm

        # Vitesse constante (ou minimale)
        vitesse_constante = V_min  # mm/itération

        # Déplacement souhaité avec vitesse fixe
        target_move = direction * vitesse_constante

        # Calcul de la Jacobienne
        J = jacobian(X, Y, phi_1)  # Jacobienne (d_rond L/ d_rond X); X = [x, y, phi_1) à l'instant courant

        # Calcul de la pseudo inverse de la Jacobienne pour estimer variation de la position de la plaque
        J_pseudo_inv = pinv(J.T @ J) @ J.T  # Pseudo-inverse de la Jacobienne
        dl = J @ target_move  # Variation attendue des longueurs des câbles
        delta_q = J_pseudo_inv @ dl  # Variation de position à partir de la longueur des câbles attendues


        # Mise à jour de la position
        X += delta_q[0]
        Y += delta_q[1]
        phi_1 += delta_q[2]
        print("\nPosition : X = ", X, "Y = ", Y, "phi_1 = ", phi_1)

        # Stockage des nouvelles valeures
        X_traj.append(X)
        Y_traj.append(Y)
        phi_traj.append(phi_1)

        # Calcul de la nouvelle longueur et vitesse des câbles
        l_curr = cable_lengths(X, Y, phi_1)
        l_prev = l_traj[-1]
        v = (l_curr - l_prev) / step  # Vitesse estimée
        v_traj.append(v)
        l_traj.append(l_curr)
        print("Longueurs des câbles : ",l_curr)

        # Arrêt si on a atteint la position finale à avec une marge de [Valeur finale * epsilon/100; Valeur finale *
epsilon/100]
        tol = epsilon / 100
        abs_tol_x = tol * max(1.0, abs(X_final))
        abs_tol_y = tol * max(1.0, abs(Y_final))
        abs_tol_phi = tol * max(1.0, abs(phi_1_final))

        if abs(X - X_final) <= abs_tol_x and \
           abs(Y - Y_final) <= abs_tol_y and \
           abs(phi_1 - phi_1_final) <= abs_tol_phi:
            print("\nArrêt à l'itération:", i, "\n")
            break


    # ----------------- Animation graphique ----------------------------------#

    def anim():
        fig, ax = plt.subplots()
        ax.set_xlim(-200, l_1 + 200)
        ax.set_ylim(-200, h_2 + 200)
        ax.set_aspect('equal')
```

```
        plate, = ax.plot([], [], 'b-', lw=2)
        cables, = ax.plot([], [], 'k--', lw=1)
        center, = ax.plot([], [], 'ro')
        traj_line, = ax.plot([], [], 'g-', linewidth=3, label="Trajectoire")

        def update(frame):
            X, Y, phi_1 = X_traj[frame], Y_traj[frame], phi_traj[frame]
            A = compute_attachment_points(X, Y, phi_1)
            plate.set_data(A[:, 0].tolist() + [A[0, 0]], A[:, 1].tolist() + [A[0, 1]])
            cable_x, cable_y = [], []
            for i in [0, 1, 3, 2]:
                cable_x += [poulies[i, 0], A[i, 0], None]
                cable_y += [poulies[i, 1], A[i, 1], None]
            cables.set_data(cable_x, cable_y)
            center.set_data([X], [Y])
            traj_line.set_data(X_traj[:frame+1], Y_traj[:frame+1])

            return plate, cables, center, traj_line

        ani = FuncAnimation(fig, update, frames=len(X_traj), interval=50, blit=True)
        return ani

    global ani
    ani = anim()
    plt.title("Simulation du Robot Parallèle à Câbles")



    # ------------------------- Tracés ---------------------------------------#

    # Tracé des longueurs de câble
    fig_var, axs_var = plt.subplots(nrows=2, ncols=2)
    fig_var.suptitle("Longueurs des câbles")
    l_traj_vec = np.array(l_traj)
    D1, D2, D3, D4 = l_traj_vec[:,0], l_traj_vec[:,1], l_traj_vec[:,2], l_traj_vec[:,3]
    for ax, D, title, color in zip(axs_var.flat, [D4, D2, D3, D1], ["Câble 4", "Câble 2", "Câble 3", "Câble 1"], ["red", "green",
"blue", "purple"]):
        Etape = np.linspace(0, nb_points, np.shape(D1)[0])
        ax.plot(Etape, D, marker='o', color=color)
        ax.set_title(title)
        ax.set_xlabel("Itération")
        ax.set_ylabel("Longueur de câble [mm]")
        ax.grid()


    # Tracé des positions des moteurs
    fig_var, axs_var = plt.subplots(nrows=2, ncols=2)
    fig_var.suptitle("Positions angulaires des moteurs")
    q_traj_vec = np.array(l_traj)
    Q1, Q2, Q3, Q4 = q_traj_vec[:,0]/r, q_traj_vec[:,1]/r, q_traj_vec[:,2]/r, q_traj_vec[:,3]/r
    for ax, Q, title, color in zip(axs_var.flat, [Q4, Q2, Q3, Q1], ["Moteur 4", "Moteur 2", "Moteur 3", "Moteur 1"], ["red",
"green", "blue", "purple"]):
        Etape = np.linspace(0, nb_points, np.shape(D1)[0])
        ax.plot(Etape, Q, marker='o', color=color)
        ax.set_title(title)
        ax.set_xlabel("Itération")
        ax.set_ylabel("Position du moteur [radians]")
        ax.grid()


    # Tracé des pas
    fig_var, axs_var = plt.subplots(nrows=2, ncols=2)
    fig_var.suptitle("Pas des moteurs")
```

```
    l_traj_vec = np.array(l_traj)
    P1, P2, P3, P4 = pas_mot * l_traj_vec[:,0], pas_mot * l_traj_vec[:,1], pas_mot * l_traj_vec[:,2], pas_mot *
l_traj_vec[:,3]
    for ax, P, title, color in zip(axs_var.flat, [P4, P2, P3, P1], ["Moteur 4", "Moteur 2", "Moteur 3", "Moteur 1"], ["red",
"green", "blue", "purple"]):
        Etape = np.linspace(0, nb_points, np.shape(D1)[0])
        ax.plot(Etape, P, marker='o', color=color)
        ax.set_title(title)
        ax.set_xlabel("Itération")
        ax.set_ylabel("Pas du moteur")
        ax.grid()


    # Tracé de la position du centre de l'effecteur
    fig_pos, ax_pos = plt.subplots()
    fig_pos.suptitle("Position du centre de l'effecteur")
    ax_pos.plot(X_traj, Y_traj, marker='o', color='orangered')
    ax_pos.set_xlabel("X [mm]")
    ax_pos.set_ylabel("Y [mm]")
    ax_pos.grid()


    # Tracé de la rotation du centre de l'effecteur
    fig_rot, ax_rot = plt.subplots()
    fig_rot.suptitle("Rotation du centre de l'effecteur")
    ax_rot.plot(Etape, phi_traj, marker='o', color='grey')
    ax_rot.set_xlabel("Itération")
    ax_rot.set_ylabel("Angle [rad]")
    ax_rot.grid()


    # Tracé des vitesses linéaires des câbles
    fig_vit_lin, axs_vit_lin = plt.subplots(nrows=2, ncols=2)
    fig_vit_lin.suptitle("Vitesses linéaires des câbles")
    v_traj_vec = np.array(v_traj)
    V1, V2, V3, V4 = v_traj_vec[:,0], v_traj_vec[:,1], v_traj_vec[:,2], v_traj_vec[:,3]
    Etape_v = np.arange(len(V1))  # une étape de moins que les longueurs

    for ax, V, title, color in zip(axs_vit_lin.flat, [V4, V2, V3, V1], ["Câble 4", "Câble 2", "Câble 3", "Câble 1"], ["red", "green",
"blue", "purple"]):
        ax.plot(Etape_v, V, marker='o', color=color)
        ax.set_title(title)
        ax.set_xlabel("Itération")
        ax.set_ylabel("Vitesse [mm/itération]")
        ax.grid()


    # Tracé des vitesses angulaires des moteurs
    fig_vit_ang, axs_vit_ang = plt.subplots(nrows=2, ncols=2)
    fig_vit_ang.suptitle("Vitesses angulaires des moteurs")
    v_traj_vec = np.array(v_traj)
    Omega1, Omega2, Omega3, Omega4 = r*v_traj_vec[:,0], r*v_traj_vec[:,1], r*v_traj_vec[:,2], r*v_traj_vec[:,3]
    Etape_v = np.arange(len(V1))  # une étape de moins que les longueurs

    for ax, Omega, title, color in zip(axs_vit_ang.flat, [Omega4, Omega2, Omega3, Omega1], ["Moteur 4", "Moteur 2",
"Moteur 3", "Moteur 1"], ["red", "green", "blue", "purple"]):
        ax.plot(Etape_v, Omega, marker='o', color=color)
        ax.set_title(title)
        ax.set_xlabel("Itération")
        ax.set_ylabel("Vitesse [rad/itération]")
        ax.grid()
```

```
    # Tracé des longueurs totales de câble
    fig_var, axs_var = plt.subplots(nrows=2, ncols=2)
    fig_var.suptitle("Longueurs totales des câbles")
    l_traj_vec = np.array(l_traj)
    D1, D2, D3, D4 = l_traj_vec[:,0], l_traj_vec[:,1], l_traj_vec[:,2], l_traj_vec[:,3]

    lambda_tot_1, lambda_tot_2, lambda_tot_3, lambda_tot_4  = [], [], [], []
    for l_tot_1, l_tot_2, l_tot_3, l_tot_4, Ye in zip(D1, D2, D3, D4, Y_traj):
        l1, l2, l3, l4 = rectification_lambda(l_tot_1, l_tot_2, l_tot_3, l_tot_4, Ye)
        lambda_tot_1.append(l1)
        lambda_tot_2.append(l2)
        lambda_tot_3.append(l3)
        lambda_tot_4.append(l4)

    for ax, D, title, color in zip(axs_var.flat, [lambda_tot_4, lambda_tot_2, lambda_tot_3, lambda_tot_1], ["Câble 4",
"Câble 2", "Câble 3", "Câble 1"], ["red", "green", "blue", "purple"]):
        Etape = np.linspace(0, nb_points, np.shape(D1)[0])
        ax.plot(Etape, D, marker='o', color=color)
        ax.set_title(title)
        ax.set_xlabel("Itération")
        ax.set_ylabel("Longueur totale de câble [mm]")
        ax.grid()

    # Affichage de tous les graphes
    plt.show()



    #----------- Création du fichier xls avec les données des test -----------#

    # Listes avec les longueurs des câbles
    longueur_cable_1 = ["Longueurs câble 1"] + list(D1)
    longueur_cable_2 = ["Longueurs câble 2"] + list(D2)
    longueur_cable_3 = ["Longueurs câble 3"] + list(D3)
    longueur_cable_4 = ["Longueurs câble 4"] + list(D4)

    # Listes avec les longueurs totales des câbles
    longueur_totale_cable_1 = ["Longueurs totales câble 1"] + list(lambda_tot_1)
    longueur_totale_cable_2 = ["Longueurs totales câble 2"] + list(lambda_tot_1)
    longueur_totale_cable_3 = ["Longueurs totales câble 3"] + list(lambda_tot_1)
    longueur_totale_cable_4 = ["Longueurs totales câble 4"] + list(lambda_tot_1)

    # Listes avec les vitesses des câbles
    vitesse_cable_1 = ["Vitesses câble 1"] + list(V1)
    vitesse_cable_2 = ["Vitesses câble 2"] + list(V2)
    vitesse_cable_3 = ["Vitesses câble 3"] + list(V3)
    vitesse_cable_4 = ["Vitesses câble 4"] + list(V4)

    # Listes avec les coordonées du centre de l'effecteur
    trajectoire_x = ["Coordonées du centre de l'effecteur sur l'axe x"] + list(X_traj)
    trajectoire_y = ["Coordonées du centre de l'effecteur sur l'axe y"] + list(Y_traj)


    # Regroupe-les dans une liste (ordre d'écriture)
    arrays = [longueur_cable_1, longueur_cable_2, longueur_cable_3, longueur_cable_4,
              longueur_totale_cable_1, longueur_totale_cable_2, longueur_totale_cable_3, longueur_totale_cable_4,
              vitesse_cable_1, vitesse_cable_2, vitesse_cable_3, vitesse_cable_4,
              trajectoire_x, trajectoire_y]

    # Nom de la feuille et du fichier
    sheet_name = "Données de test numérique"  # Nom de la feuille
    filename = "Data_test.xlsx" # Nom du fichier xls
```

```
# Paramètre : écriture verticale ou horizontale
ecriture_verticale = False  # Mettre False pour les mettre côte à côte

# Création du writer
with pd.ExcelWriter(filename, engine='openpyxl') as writer:
    start_row, start_col = 0, 0

    for array in arrays:
        df = pd.DataFrame(array)
        # Écrit le DataFrame dans la feuille, à la position voulue
        df.to_excel(writer, sheet_name=sheet_name, startrow=start_row, startcol=start_col, index=False, header=False)

        # Mise à jour de la position de départ pour le prochain array
        if ecriture_verticale:
            start_row += df.shape[0] + 1  # Ajoute une ligne vide entre les blocs
        else:
            start_col += df.shape[1] + 1  # Ajoute une colonne vide entre les blocs

print(f"\n\nLes tableaux ont été écrits dans la feuille '{sheet_name}' du fichier '{filename}'.")
```

Explication détaillé de la fonction principale

Cette fonction est la fonction qui lance la simulation avec les paramètres de départ choisi. Pour lancer cette simulation il faut taper « animation_2($X_{initial}, Y_{initial}, \phi 1_{initial}, X_{final}, Y_{final}, \phi 1_{final}, step, nb_{points}, \epsilon$) » dans le terminale. Voici un tableau explicatif de l'utilité de chaque paramètres :

Nom du paramètre	Type	Description
$X_{initial}$	Float	Coordonnées sur l'axe x finale du centre de l'effecteur initialement
$Y_{initial}$	Float	Coordonnées sur l'axe y finale du centre de l'effecteur initialement
$\phi 1_{initial}$	Float	Angle de rotation finale autour de l'axe z du centre l'effecteur initialement
X_{final}	Float	Coordonnées sur l'axe x finale du centre de l'effecteur
Y_{final}	Float	Coordonnées sur l'axe y finale du centre de l'effecteur
$\phi 1_{final}$	Float	Angle de rotation finale autour de l'axe z du centre l'effecteur
V_min	Float	Vitesse minimale de l'effecteur
$step$	Float	Taille des pas de la simulation (en secondes)
nb_{points}	Int	Nombre de points pour effectuer la simulation
ϵ	Int	Valeur en % de la bande d'arrêt

Tableau 6: Description des paramètres en entrée de la fonction de la simulation

Ensuite, la fonction animation_2, va créer 3 vecteurs pour suivre l'évolution du déplacement de l'effecteur durant la simulation : $X_{traj}, Y_{traj}, \phi_{traj}$. Ces 3 vecteurs seront initialisés avec les coordonnées initiaux de l'effecteur $X_{initial}, Y_{initial}$ *et* $\phi 1_{initial}$.

```
# -------------- Initialisation des paramètres -------------------------#

# Initialisation à la position initiale
X0, Y0 , phi_1_0 = X_inital, Y_initial, phi_1_initial
print("\nPosition initiale: X0 = ", X0, "Y0 = ", Y0, "phi_1_0 = ", phi_1_0)
X_traj, Y_traj, phi_traj = [X0], [Y0], [phi_1_0]
```

L'on va ensuit calculer la longueurs des 4 câbles avec cette position initiale ; grâce à la fonction « cable_lengths » et afficher cette longueur sur la console.

```python
# Initialisation des longueurs de câbles
l_0 = cable_lengths(X0, Y0, 0.0)
l_traj = [l_0]
print("Longueurs des câbles initiales: ",phi_1_0)
```

Puis, l'on va renommer les paramètres initiaux comme étant $X, Y\ et\ \phi_1$. Et créer le vecteur des vitesses. Avant d'entrer dans la boucle de simulation.

```python
X, Y, phi_1 = X0, Y0, phi_1_0

# Initialisation des vitesses
v_traj = []
```

Pour la boucle de simulation, l'on va commencer en calculant plusieurs données :

- $dx = X_{final} - X_{actuel}$
- $dy = \ Y_{final} - Y_{actuel}$
- $d\phi = \ \phi_{final} - \phi_{actuel}$
- $erreur = [dx\ dy\ d\phi]$
- $direction = \frac{erreur}{||erreur||}$

De ces valeurs, on va calculer le déplacement souhaité grâce à la vitesse minimale que nous avons rentrée au début (V_min)

```python
# ---------------- Boucle de simulation -------------------------------#

for i in range(nb_points):
    dx = X_final - X
    dy = Y_final - Y
    dphi = phi_1_final - phi_1
    error = np.array([dx, dy, dphi])
    error_norm = np.linalg.norm(error)

    # Pour éviter de diviser par 0
    if error_norm < 1e-4:
        break

    # Direction normalisée de l'erreur
    direction = error / error_norm

    # Vitesse constante (ou minimale)
    vitesse_constante = V_min  # mm/itération

    # Déplacement souhaité avec vitesse fixe
    target_move = direction * vitesse_constante
```

L'on va ensuite utiliser la fonction jacobian précédemment définie pour calculer la Jacobienne à ces coordonnées ($J = \frac{\partial \lambda}{\partial X}, X = [x, y, \phi_1]$). Cette Jacobienne va nous servir pour calculer la longueur attendues des câbles.

```
# Calcul de la Jacobienne
J = jacobian(X, Y, phi_1)  # Jacobienne (d_rond L/ d_rond X); X = [x, y, phi_1) à l'instant courant
dl = J @ target_move  # Variation attendue des longueurs des câbles
```

Puis l'on va utiliser ces longueurs de câble attendues pour calculer la position de l'effecteur avec ces longueurs de câbles, en calculant la pseudo-inverse de la Jacobienne

Rappel : $J^{+} = (J * J^{t})^{-1} * J^{t}$

```
# Calcul de la pseudo inverse de la Jacobienne pour estimer variation de la position de la plaque
J_pseudo_inv = pinv(J.T @ J) @ J.T  # Pseudo-inverse de la
dl = J @ target_move  # Variation attendue des longueurs des câbles
delta_q = J_pseudo_inv @ dl  # Variation de position à partir de la longueur des câbles attendues
```

Puis, vient la phase de mise à jour des données et leurs stockage dans des listes pour l'affichage :

```
# Mise à jour de la position
X += delta_q[0]
Y += delta_q[1]
phi_1 += delta_q[2]
print("\nPosition : X = ", X, "Y = ", Y, "phi_1 = ", phi_1)

# Stockage des nouvelles valeures
X_traj.append(X)
Y_traj.append(Y)
phi_traj.append(phi_1)
```

Une fois que les positions ont été mises à jour, on va calculer la longueurs des câbles avec la fonction cable_lengths :

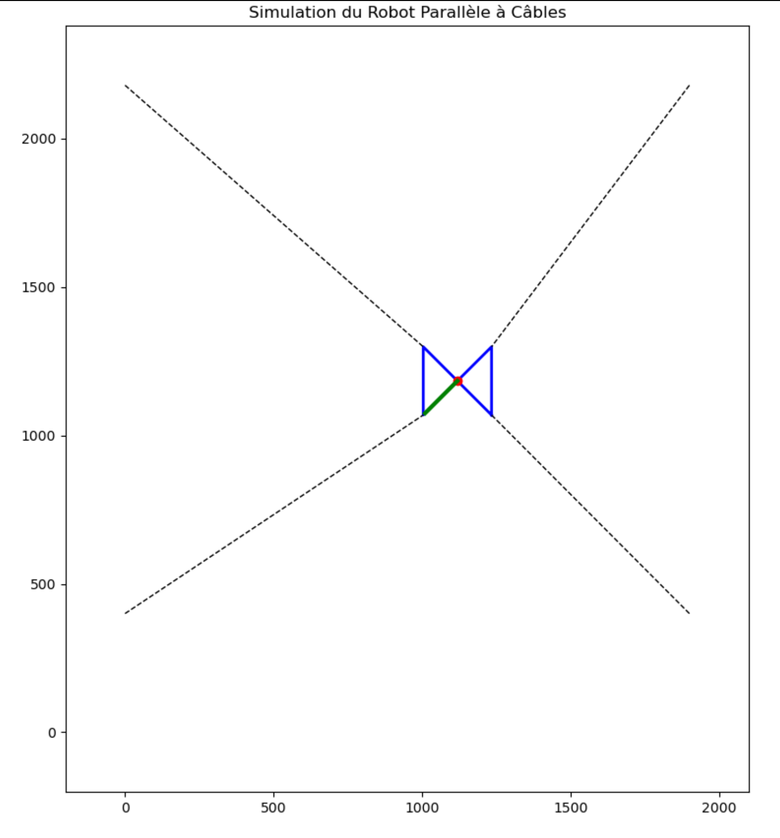
```
# Calcul de la nouvelle longueur et vitesse des câbles
l_curr = cable_lengths(X, Y, phi_1)
l_prev = l_traj[-1]
v = (l_curr - l_prev) / step  # Vitesse estimée
v_traj.append(v)
l_traj.append(l_curr)
print("Longueurs des câbles : ",l_curr)
```

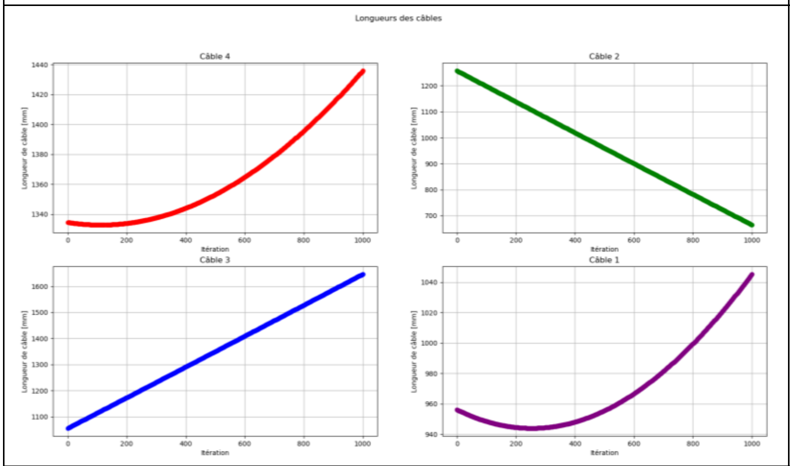
Pour finir on met une condition d'arrêt de la boucle, si les coordonnées (en $x, y\ et\ \phi_1$) de l'effecteur se trouvent dans un intervalle de $[coordonnée_{finale} - \frac{\epsilon}{100}; coordonnées_{finale} + \frac{\epsilon}{100}]$, la simulation s'arrête et on renvoi l'itération où la simulation s'est arrêtée.

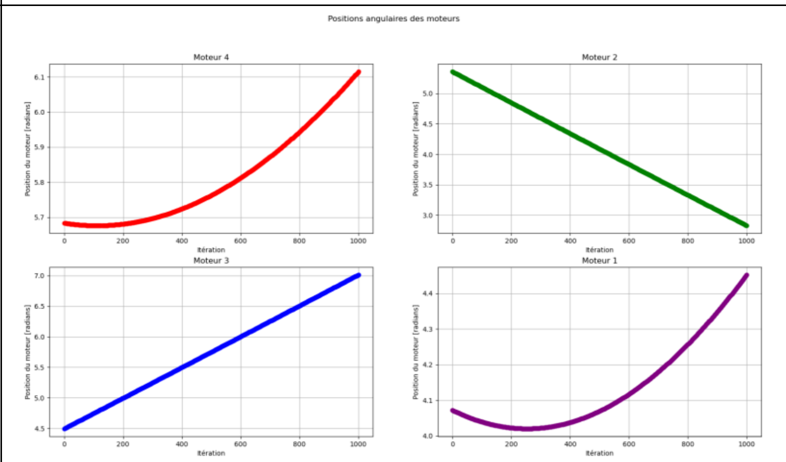
```
# Arrêt si on a atteint la position finale à avec une marge de [Valeur finale * epsilon/100; Valeur finale * epsilon/100]
tol = epsilon / 100
abs_tol_x = tol * max(1.0, abs(X_final))
abs_tol_y = tol * max(1.0, abs(Y_final))
abs_tol_phi = tol * max(1.0, abs(phi_1_final))

if abs(X - X_final) <= abs_tol_x and \
   abs(Y - Y_final) <= abs_tol_y and \
   abs(phi_1 - phi_1_final) <= abs_tol_phi:
    print("\nArrêt à l'itération:", i, "\n")
    break
```

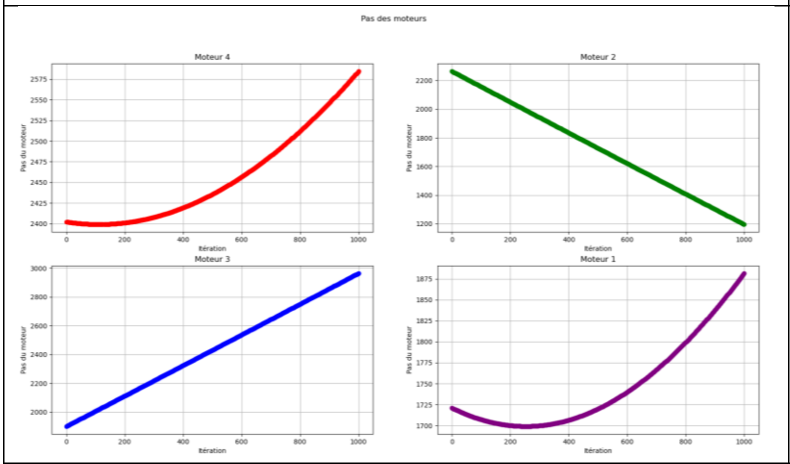
Nous ne commenterons pas la partie d'affichage de cette fonction car peu d'intérêt.

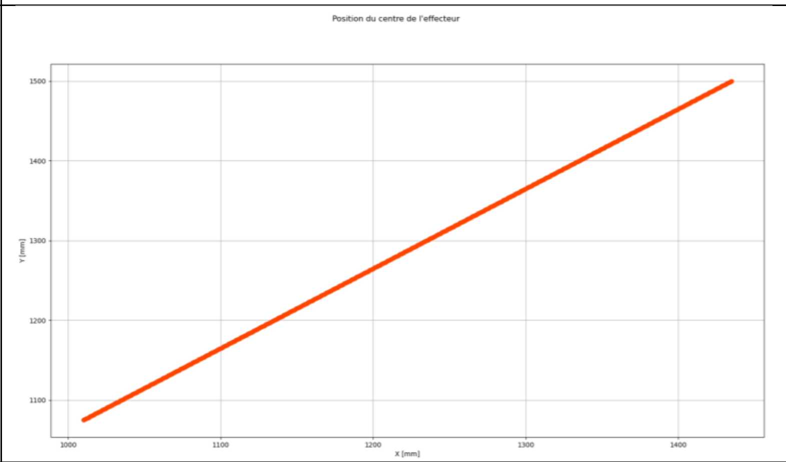
Résulats complets modèle 2

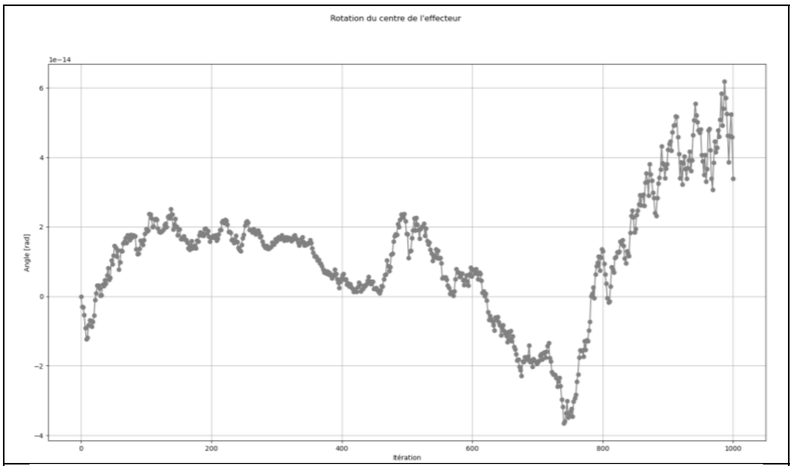


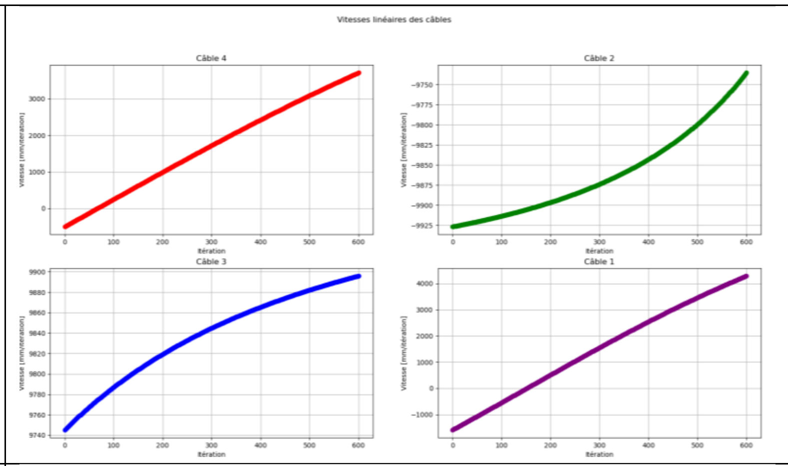


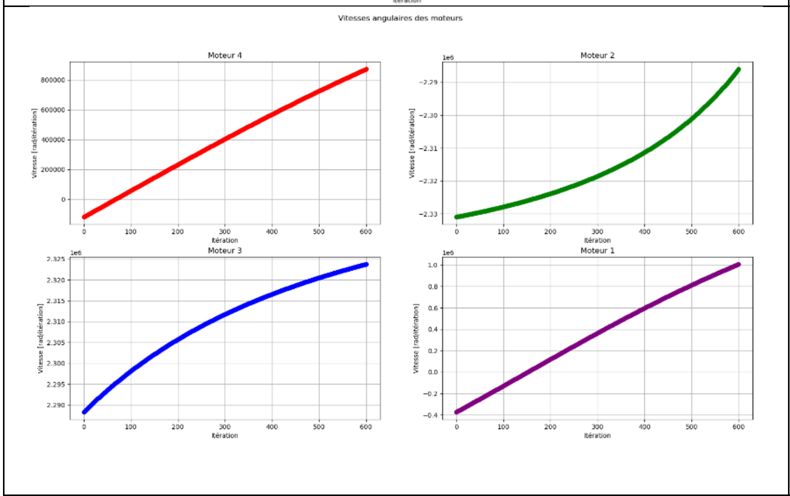


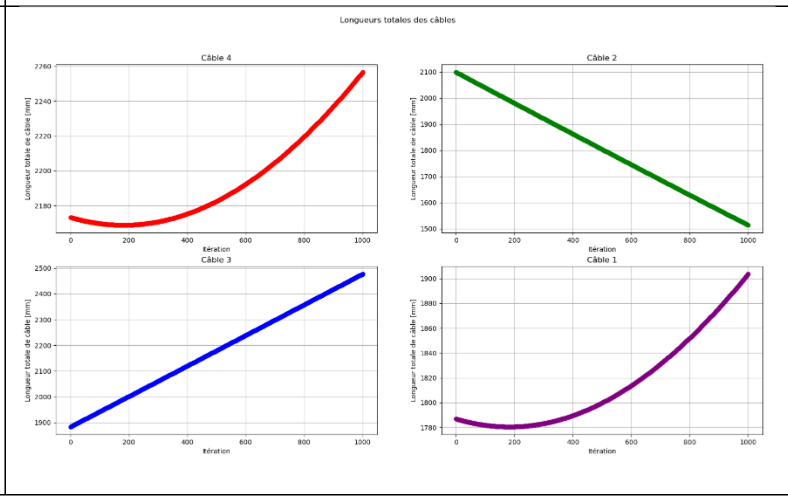





Rotation du centre de l'effecteur
Angle [rad]
Itération


Vitesses linéaires des câbles
Câble 4
Câble 2
Câble 3
Câble 1


Vitesses angulaires des moteurs
Moteur 4
Moteur 2
Moteur 3
Moteur 1


Longueurs totales des câbles
Câble 4
Câble 2
Câble 3
Câble 1