



**Modélisation d’un robot à câbles et estimation des paramètres**

**\_**

**Projet Mécatronique**

Gwezheneg RIVIERE

Ngatam THIEBAUT

Table des matières

[1](#_Toc200981036)

[Introduction et contexte : 3](#_Toc200981037)

[Contexte 3](#_Toc200981038)

[Robot à câbles 4](#_Toc200981039)

[Objectifs 5](#_Toc200981040)

[Remerciements 6](#_Toc200981041)

[Schémas : 7](#_Toc200981042)

[Schéma des nominations : 7](#_Toc200981043)

[Schéma des dimensions : 8](#_Toc200981044)

[Schéma des bases et repères : 9](#_Toc200981045)

[Modèle géométrique : 11](#_Toc200981046)

[Fermetures géométriques 11](#_Toc200981047)

[Modèle géométrique direct 12](#_Toc200981048)

[Modèle géométrique inverse 13](#_Toc200981049)

[Vérification du modèle géométrique 15](#_Toc200981050)

[Test, objectif et attentes 15](#_Toc200981051)

[Résultats du programme Python (modèle 1) 16](#_Toc200981052)

[Modèle 2 – Suivi des points d’accroche de l’effecteur 18](#_Toc200981053)

[Résultats du test sur le robot 21](#_Toc200981054)

[Modèle cinématique 22](#_Toc200981055)

[Calcul Jacobienne et modèle cinématique inverse 22](#_Toc200981056)

[Modèle cinématique directe 24](#_Toc200981057)

[Complexification du modèle et prise en compte de l’enroulement autour des poulies 25](#_Toc200981058)

[Modèle dynamique 29](#_Toc200981059)

[Formalisme des écritures : 29](#_Toc200981060)

[Améliorations possibles 30](#_Toc200981061)

[Annexes : 31](#_Toc200981062)

[Fiche de PJE 31](#_Toc200981063)

[Explication du code du modèle 1 : 32](#_Toc200981064)

[Importation des différents modules 32](#_Toc200981065)

[Paramètres du systèmes 32](#_Toc200981066)

[Équations du système 33](#_Toc200981067)

[Fonctions subsidiaires 34](#_Toc200981068)

[Fonction principale 37](#_Toc200981069)

[Explication du code du modèle 2 : 39](#_Toc200981070)

[Importation des différents modules : 39](#_Toc200981071)

[Paramètres du système : 39](#_Toc200981072)

[Fonctions subsidiaires : 40](#_Toc200981073)

[Fonction principale : 42](#_Toc200981074)

[Explication détaillé de la fonction principale 54](#_Toc200981075)

[To do list 58](#_Toc200981076)



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



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

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 objectif sont recensé 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 direct du robot |  |
| Modèle dynamique inverse du robot |  |
| Implantation des modèles dynamique sur Python |  |
| Tests sur les modèles dynamique |  |
| Explication des codes et mise en ligne |  |
| Test avec les tensions de câbles |  |

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 :

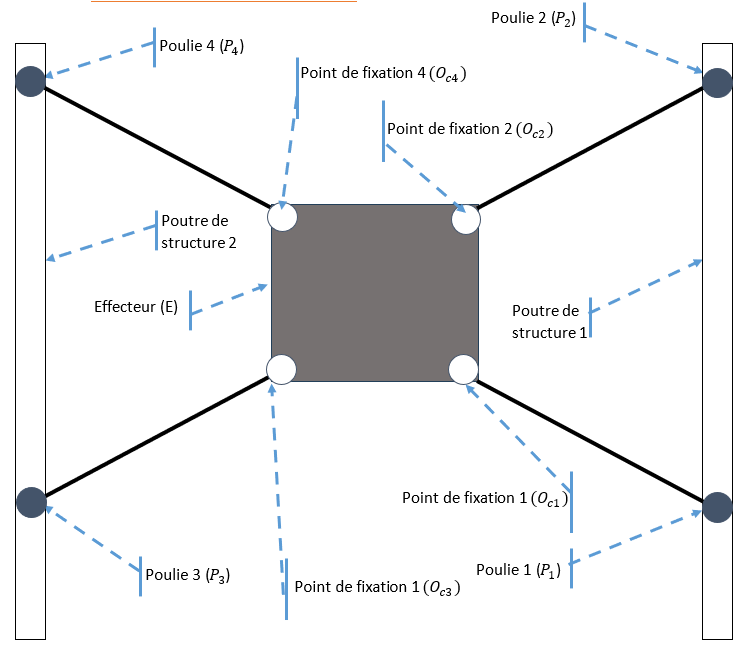


Figure : Schéma des nominations

Schéma des dimensions :

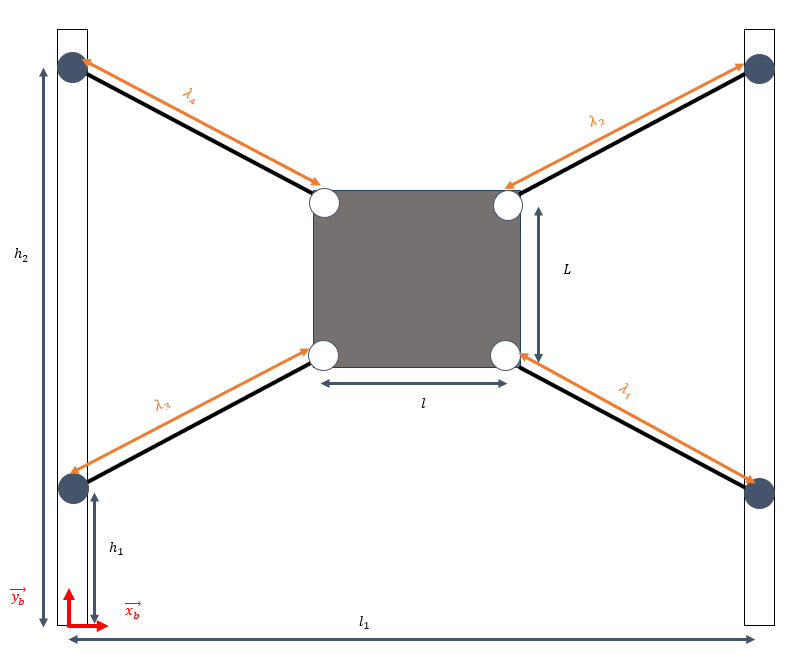


Figure : Schéma des dimensions

|  |  |  |
| --- | --- | --- |
| **Dimension** | **Explication** | **Dimensions variables ?** |
|  | Hauteur 1, hauteur des poulies 1 et 3 par rapport au sol. Dimension | Non |
|  | Hauteur 2, hauteur des poulies 2 et 4 par rapport au sol. | Non |
|  | Longueur 1, longueur entre les pieds de la structure du robot à câbles. | Non |
|  | Longueur de la plaque de l’effecteur. | Non |
|  | Largeur de la plaque de l’effecteur. | Non |
|  | Longueur du câbles i. | Oui |

|  |  |
| --- | --- |
| **Repère** | **Objet associé** |
|  | Structure fixe du robot à câbles. Repère parallèle au sol. |
|  | Poulie i du robot à câbles. Repère parallèle au sol. |
| Pour i = 3,4 | Point d’accroche i sur l’effecteur du robot à câbles. Repère avec un angle |
| Pour i = 1, 2 | Point d’accroche i sur l’effecteur du robot à câbles . Repère avec un angle |
|  | Effecteur du robot à câbles. Repère avec un angle . |

Schéma des bases et repères :

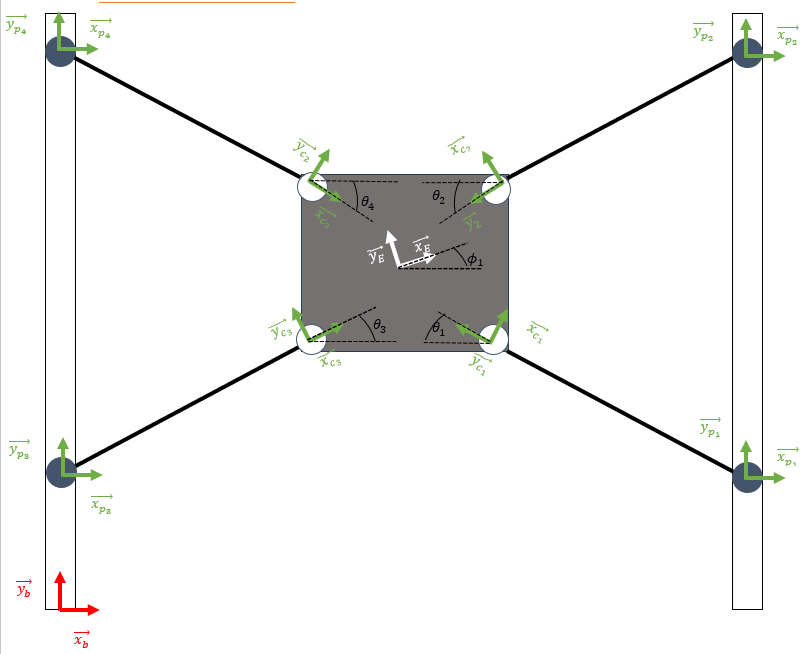


Figure : Schéma des bases et répères

Modèle géométrique :

Fermetures géométriques

Fermeture géométrique par la poulie 1,  :



Fermeture géométrique par la poulie 2,  :



Fermeture géométrique par la poulie 3,  :

Fermeture géométrique par la poulie 4,  :

Modèle géométrique direct

On a donc :

Modèle géométrique inverse

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

1. :
2. :
3. :

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

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

**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.**

1. :
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 , on calculera les longueurs de câbles liées à cette position plus tard, pour l’instant on note ces longueurs , et on note le vecteur , 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 , dont les longueurs de câbles associé sont et les position angulaires des moteurs sont

La variation des positions angulaires qui est donnée par

**(17)** :

Avec , 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 ) et translation de 1m vers le haut ( + 500 mm sur l’axe . 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).

|  |  |
| --- | --- |
| Coordonées initiales | Coordonnés finales |
|  |  |
|  |  |
|  |  |

Tableau : 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 :

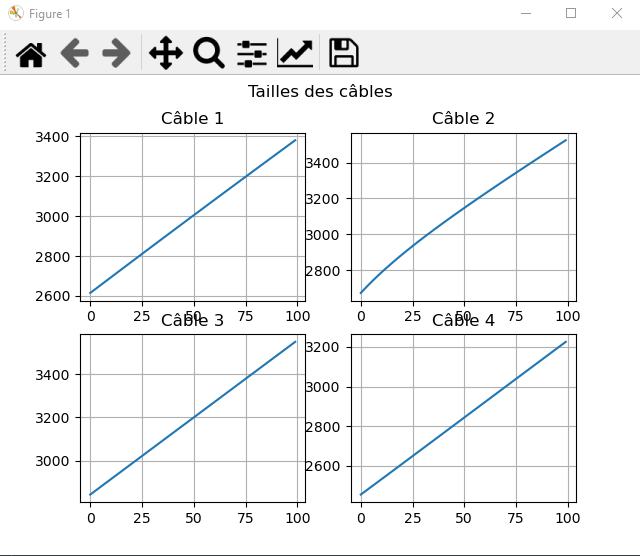
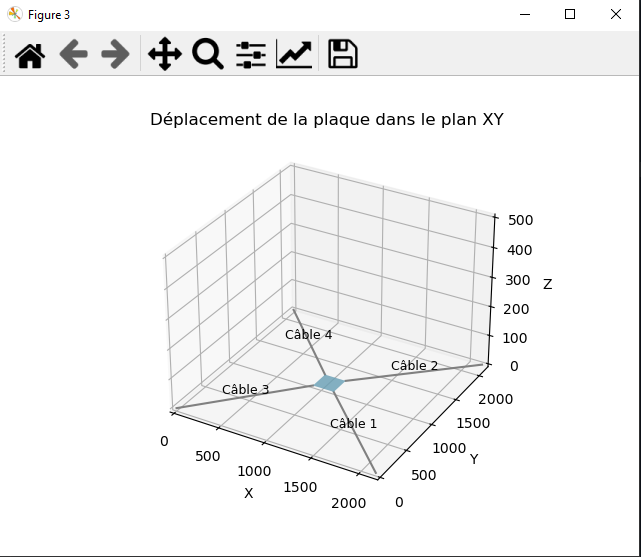
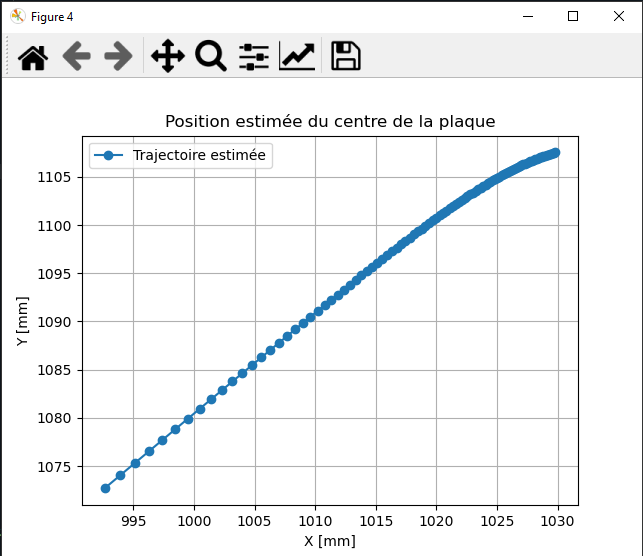


Figure : Résultats de simulation avec le modèle 1

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’attient 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 sur le schéma ci-dessous).

Effecteur

Poulie

Tambour

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

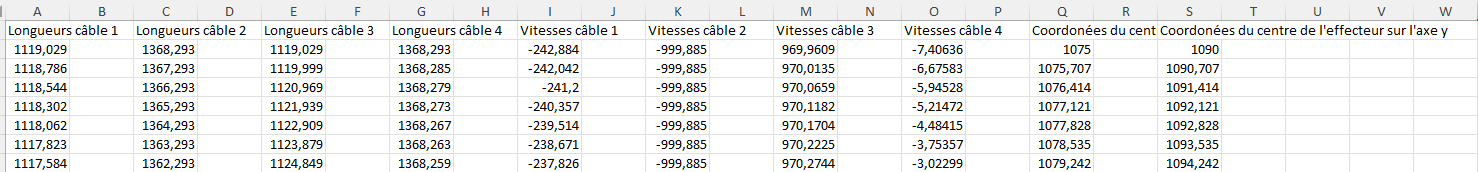
En lance ensuite le code de ce nouveau modèle, pour faire le test avec notre modèle géométrique. Ce code crée un document Excel dans lequel sont regroupé les données des simulations :

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 |



On devrait donc avoir à la position initiale du robot les longueurs de câbles 1119.09 mm (pour les câbles 1 et 3), et 1368.293 (pour les câbles 2 et 4).

Une image contenant texte, nombre, capture d’écran, Parallèle

Le contenu généré par l’IA peut être incorrect.

Pour la fin de la simulation, on remarque l’effecteur n’atteint pas exactement la position demandé mais nous donne des valeurs de longueurs de câbles pour des positions très proches (en jaune sur l’image ci-dessus), on peut se dire que si on obtient des valeurs de longueurs de câbles relativement proches de ces dernières, notre modèle est validé

Résultats du programme Python (Modèle 2)



Tableau : Résultats des test 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. 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 8: 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).



Figure 9: 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 :

* : 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).
* : la longueur de câble directement sur la gouttière de la poulie (en orange dans le schéma ci-dessous).
* : la longueur entre la poulie et le tambour (en jaune dans le schéma ci-dessous).

Effecteur

Poulie

Tambour

Figure 10: Schéma de la complexification

Mise en équations :

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

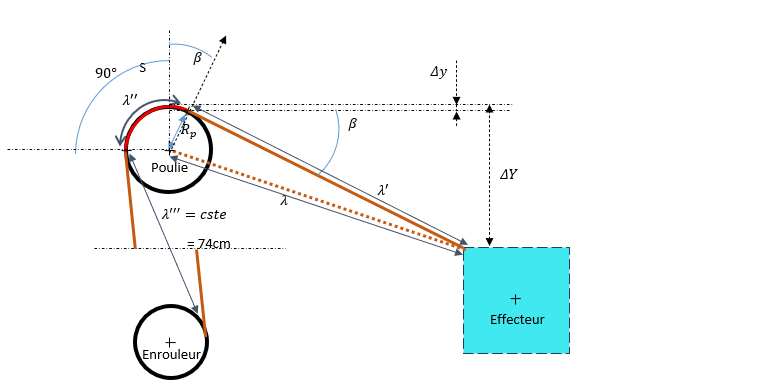


Figure 11: Schéma prise en compte de la longueur totale de câble

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’’’.

**(22) :**

**(23) :**  ou

**(24) :**

**(25) :** λ’’’ = cste = 74 cm

**(26) :**

L’expression de **(23)** **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 °.**

**Estimations faites :**

Remarque :

Ces approximations marchent pour des positions où l’effecteur n’est pas trop proche des poulies (notamment selon l’axe ).

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 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 () et notre vecteur des entrées , on a :



Le modèle géométrique inverse de notre système est donné par :

On notera les composantes du vecteur et les composantes du vecteur .

La fonction f, permet de calculer les entrées de notre système , à partir des variables de sorties de notre système ( Cette fonction vectorielle découle des équations (13), (14), (15) et (16) présentés précédemment.

L’équation **(17)**, nous donne :

 ,

Or les sont des valeurs constantes, on a donc :

**(18)**  :

Or, la relation entre , par définition, s’écrit :

**(19)**  :

On en déduit :

**(20)** :

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

**(21)  :**

Avec les , les composantes du vecteur et les , les composantes du vecteur

Par dérivation, on en déduit :

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **i =** |  |  |  |  |
| **1** |  |  |  |  |
| **2** |  |  |  |  |
| **3** | 0 |  |  |  |
| **4** | 0 |  |  |  |

Les sont les coordonnées du point de la poulie i (les points . Et les sont les coordonnées du point d’accroche i sur l’effecteur par rapport au centre de l’effecteur (les points . On a donc :

Tableau  : Tableau d’assignation des paramètres

Modèle cinématique directe

Pour le modèle cinématique direct, 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 , pour une matrice Jacobienne noté .

On a :

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

Modèle dynamique

Formalisme des écritures :

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 :

* **f** est le vecteur des forces extérieurs sur la plaque (avec des coordonnées en et )
* **τ** est un vecteur 1 x 1 contenant le moment autour de l’axe auquel la plaque est soumise.
* **M** est la matrice de taille n x n (avec n le nombre de DDL = 3)

Améliorations possibles

**Modèle en 3D :**

**Consigne de vitesse :** L’implantation d’une vitesse dans notre 2ème modèle, 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 phase de vitesse de notre système.

Annexes :

Fiche de PJE

Une image contenant texte, capture d’écran, Police

Le contenu généré par l’IA peut être incorrect.

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 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 |

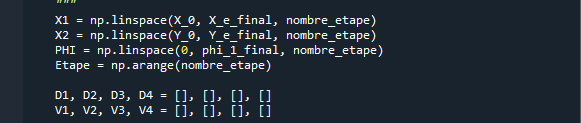
Crée une animation 3D du mouvement de la plaque à partir des longueurs de câbles

|  |
| --- |
| 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).



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.

Une image contenant texte, capture d’écran, Police, logiciel

Le contenu généré par l’IA peut être incorrect.

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 :

|  |
| --- |
| 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 :

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 :

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 (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 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 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  # Fonction principale de simulation  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 déplcement de l'effecteur du robot parallèle à câble de sa postion inital (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 du centre de l'effecteur en fonction de l'itération choisi  - le 4ème affiche la rotation du centre de l'effecteur en fonction de l'itération choisi  - Le 5ème affiche les vitesses linéaires des câbles en fonction de l'itération choisi  - Le 6ème affiche les vitesse de rotation des moteur en fonction de l'itération choisi    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  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 loongueurs des câbles  longeur\_cable\_1 = ["Longueurs câble 1"] + list(D1)  longeur\_cable\_2 = ["Longueurs câble 2"] + list(D2)  longeur\_cable\_3 = ["Longueurs câble 3"] + list(D3)  longeur\_cable\_4 = ["Longueurs câble 4"] + list(D4)  # 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 = [longeur\_cable\_1, longeur\_cable\_2, longeur\_cable\_3, longeur\_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( » dans le terminale. Voici un tableau explicatif de l’utilité de chaque paramètres :

|  |  |  |
| --- | --- | --- |
| Nom du paramètre | Type | Description |
|  | Float | Coordonnées sur l'axe x finale du centre de l'effecteur initialement |
|  | Float | Coordonnées sur l'axe y finale du centre de l'effecteur initialement |
|  | Float | Angle de rotation finale autour de l'axe z du centre l'effecteur initialement |
|  | Float | Coordonnées sur l'axe x finale du centre de l'effecteur |
|  | Float | Coordonnées sur l'axe y finale du centre de l'effecteur |
|  | Float | Angle de rotation finale autour de l'axe z du centre l'effecteur |
|  | Float | Vitesse minimale de l’effecteur |
|  | Float | Taille des pas de la simulation (en secondes) |
|  | Int | Nombre de points pour effectuer la simulation |
|  | Int | Valeur en % de la bande d'arrêt |

Tableau 4: 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 : . Ces 3 vecteurs seront initialisés avec les coordonnées initiaux de l’effecteur .

Une image contenant texte, capture d’écran, Police

Le contenu généré par l’IA peut être incorrect.

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.

Une image contenant texte, capture d’écran, Police

Le contenu généré par l’IA peut être incorrect.

Puis, l’on va renommer les paramètres initiaux comme étant . Et créer le vecteur des vitesses. Avant d’entrer dans la boucle de simulation.

Une image contenant texte, capture d’écran, Police

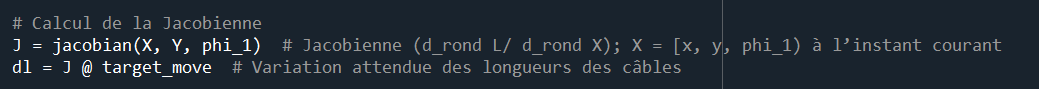
Le contenu généré par l’IA peut être incorrect.

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

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

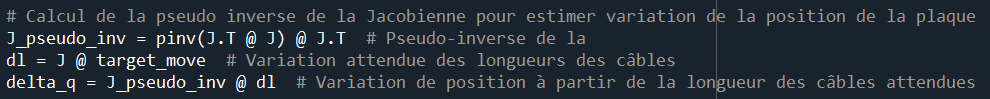
Une image contenant texte, capture d’écran, logiciel, Police

Le contenu généré par l’IA peut être incorrect.

L’on va ensuite utiliser la fonction jacobian précédemment définie pour calculer la Jacobienne à ces coordonnées ( . Cette Jacobienne va nous servir pour calculer la longueur attendues 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 :



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

Une image contenant texte, capture d’écran, Police

Le contenu généré par l’IA peut être incorrect.

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

Une image contenant texte, capture d’écran, Police

Le contenu généré par l’IA peut être incorrect.

Une image contenant texte, Police, capture d’écran

Le contenu généré par l’IA peut être incorrect.Pour finir on met une condition d’arrêt de la boucle, si les coordonnées (en de l’effecteur se trouvent dans un intervalle de , la simulation s’arrête et on renvoi l’itération où la simulation s’est arrêtée.

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

Résulats complets modèle 2

|  |  |
| --- | --- |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

To do list

* Simulation :
  + Vérifier que le nombre de step est bien liée à la position du robot
  + Expliquer le 2ème sur le Script Python (README.txt ?)
  + Complexification du 2ème modèle
  + Calcul pseudo inverse de la Jacobienne avec données numérique pour modèle direct et l’implémenter dans le code
  + Vérifier modèle géométrique directe et inverse, modèle cinématique direct
  + Consigne de vitesse avec V\_min
  + Vidéo pour expérience
  + Photo expérience
  + Discuter avec les autres groupes
* Tensions des câbles :
  + Voir avec le groupe d’Augustin et Nicolas où ils en sont
  + Voir comment relier la tension des câbles à la position de l’effecteur
  + Test statique sur la tensions des câbles avec notre simulation (cf M. Margerit)
* Modèle dynamique :
  + S’y coller…
* Modèle 3D :
  + …