



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

**\_**

**Projet Mécatronique**

Gwezheneg RIVIERE

Ngatam THIEBAUT

Table des matières

[1](#_Toc198727699)

[Introduction et contexte : 3](#_Toc198727700)

[Contexte 3](#_Toc198727701)

[Robot parallèle à câbles 3](#_Toc198727702)

[Objectifs 3](#_Toc198727703)

[Schémas : 4](#_Toc198727704)

[Schéma des nominations : 4](#_Toc198727705)

[Schéma des dimensions : 5](#_Toc198727706)

[Schéma des bases : 6](#_Toc198727707)

[Modèle géométrique : 7](#_Toc198727708)

[Fermetures géométriques 7](#_Toc198727709)

[Modèle géométrique direct 8](#_Toc198727710)

[Modèle géométrique inverse 9](#_Toc198727711)

[Modèle cinématique 11](#_Toc198727712)

[Calcul Jacobienne et modèle cinématique inverse 11](#_Toc198727713)

[Modèle cinématique directe 13](#_Toc198727714)

[Vérification du modèle géométrique 14](#_Toc198727715)

[Test, objectif et attentes 14](#_Toc198727716)

[Résultats du programme Python 15](#_Toc198727717)

[Résultats du test sur le robot 17](#_Toc198727718)

[Modèle dynamique 18](#_Toc198727719)

[Formalisme des écritures : 18](#_Toc198727720)

[Annexes : 19](#_Toc198727721)

[Données du sytème : 19](#_Toc198727722)



Introduction et contexte :

Contexte

Robot parallèle à câbles

Objectifs

Schémas :

Schéma des nominations :

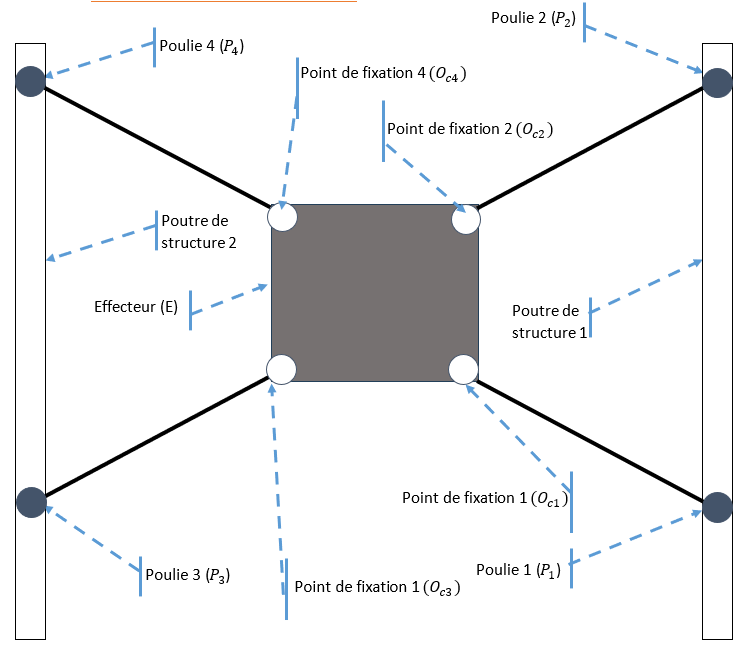


Figure : Schéma des nominations

Schéma des dimensions :

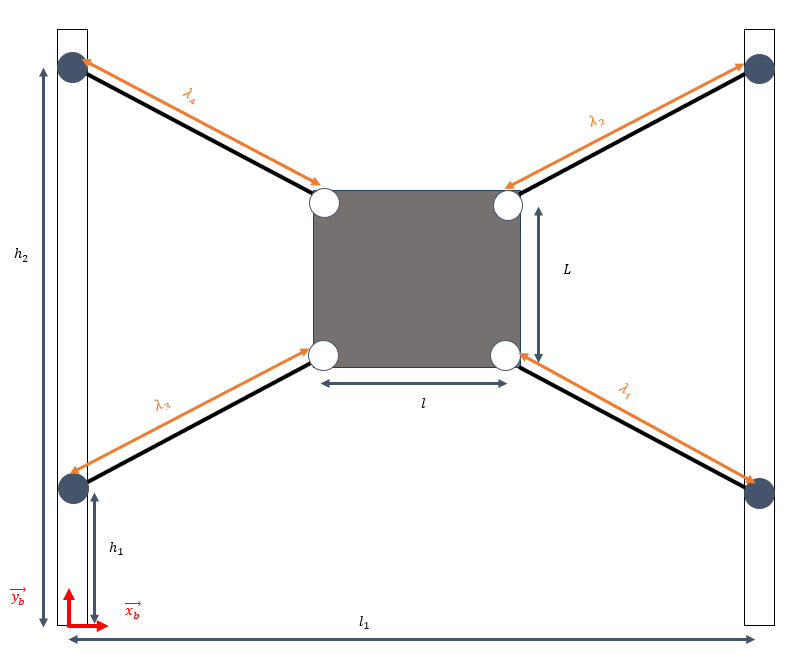


Figure : Schéma des dimensions

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

Schéma des bases et repères :

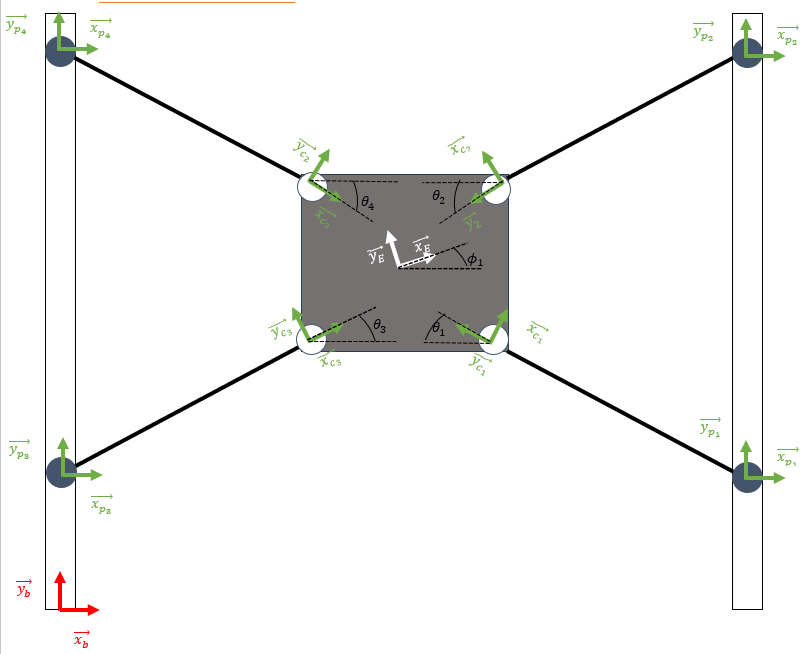


Figure : Schéma des bases et répères

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

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 :

Vérification du modèle géométrique

Test, objectif et attentes

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 parallèle 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 |
|  |  |
|  |  |
|  |  |

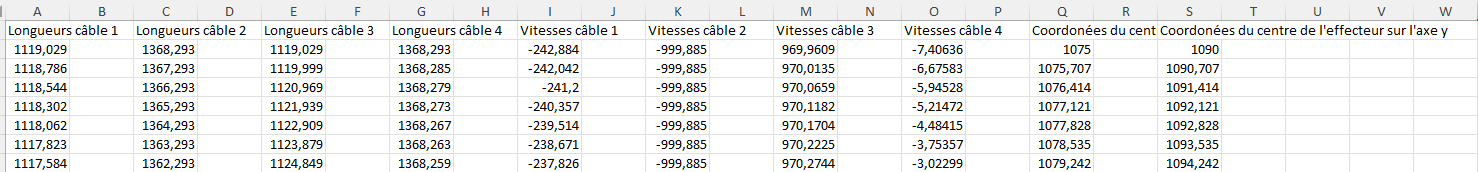
Tableau : Tableau des coordonnées pour le test

L’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.

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 parallèle 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

En lançant notre programme python, on a dans le document Excel qui regroupe les données de simulation :

|  |
| --- |
| animation\_2(X\_0, Y\_0, 0, X\_0 + 500, Y\_0 + 500, 0, 0.001, 1000, 0) # Notons que le dernier 0, correspond à l’intervalle en % dans lequel la simulation s’arrête si les coordonnées de l’effecteur atteigne cet intervalle. |

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 test sur le robot



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

Rappelons à quoi ressemble notre modèle :



Figure : 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 : 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 :

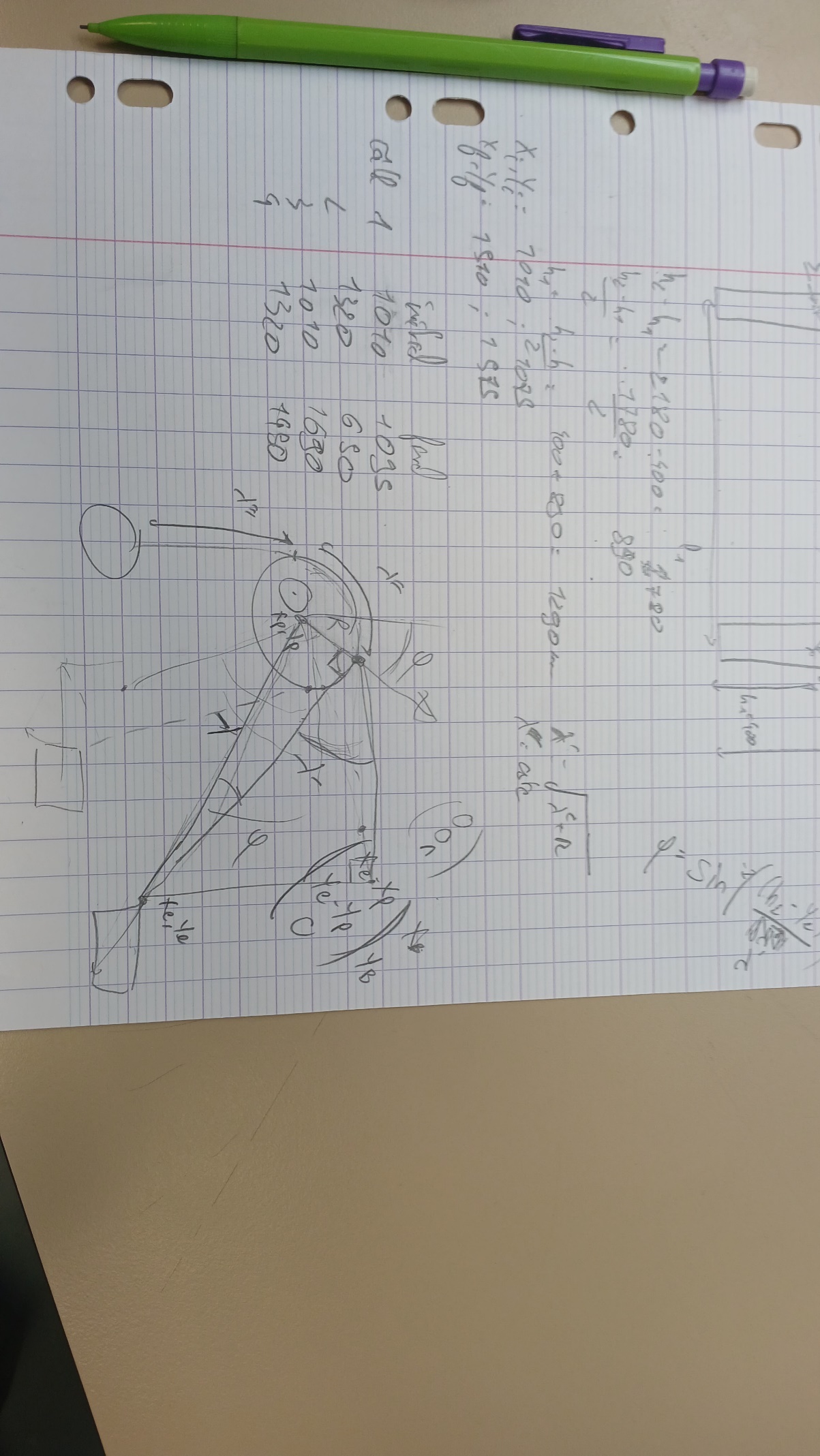
* : la longueur de câble entre le point d’attache de l’effecteur et le point d’arrivée de la poulie (dans le cercle bleu ci-contre et dans le schéma ci-dessous).
* : la longueur de câble directement sur la gouttière de la poulie (en orange dans le schéma ci-dessous).
* : la longueur entre la poulie et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

Figure : Schéma de la complexification

Effecteur

Poulie

Tambour



Modèle dynamique

Formalisme des écritures :

A noter que l’on se place dans un premier temps dans le cas ou les centres géométriques et massiques soont 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)

Annexes :

Données du sytème :

|  |
| --- |
| ## 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 = 2150 # (en mm) distance entre 2 pieds 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  ## Coefficients pour le calcul  r = K \* (e\*\*2 + (rho\*\*2)/2\*np.pi) # 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 |

To do list

* Simulation :
  + Vérifier que le nombre de step est bien liée à la position du robot
  + Expliquer le 1er modèle n’atteint pas les coordonnées voulus et pourquoi on a dû passer sur le 2ème modèle
  + Expliquer le 2ème modèle
  + Écrire les résultats des test avec le 2ème modèle
  + Expliquer le 1er modèle et le 2ème sur le Script Python (README.txt ?)
  + Calcul pseudo inverse de la Jacobienne avec données numérique pour modèle direct et l’implémenter dans le code
* 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. Margeri)
* Modèle dynamique :
  + S’y coller…
* Modèle 3D :
  + …

Explication du code de la simulation :

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  # 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  ## 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 |

Fonctions subsidiaires :

Matrice de Rotation 2D pour un angle :

|  |
| --- |
| def rotation\_matrix(phi):  return np.array([  [np.cos(phi), -np.sin(phi)],  [np.sin(phi), np.cos(phi)]  ]) |

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

|  |
| --- |
| def compute\_attachment\_points(X, Y, phi):  R = rotation\_matrix(phi)  return np.array([[X, Y]]) + (v\_attache @ R.T) |

Calcul des longueurs des câbles entre les centres des poulies et les points d’attaches sur l’effecteur :

|  |
| --- |
| def cable\_lengths(X, Y, phi):  A = compute\_attachment\_points(X, Y, phi) # Calcul les coordonées des points d'attache de la plaque  return 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 |

Calcul de la Jacobienne (variation des longueurs de câbles par rapport aux coordonnées)

|  |
| --- |
| def jacobian(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 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  dl = J @ target\_move # Variation attendue des longueurs des câbles  lambda\_reg = 1e-4 # Paramètre de régularisation    # Calcul de la pseudo inverse de la Jacobienne pour estimer variation de la position de la plaque  J\_pseudo\_inv = pinv(J.T @ J + lambda\_reg \* np.eye(3)) @ J.T # Pseudo-inverse de la Jacobienne  delta\_q = J\_pseudo\_inv @ (l\_traj[-1] + dl - l\_traj[-1]) # Variation de position  # 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\_moins; Valeur finale \* epsilon\_plus]  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')  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])  return plate, cables, center  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("Tailles 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")  p\_traj\_vec = np.array(l\_traj)  P1, P2, P3, P4 = 360\*p\_traj\_vec[:,0]/(2\*np.pi), 360\*p\_traj\_vec[:,1]/(2\*np.pi), 360\*p\_traj\_vec[:,2]/(2\*np.pi), 360\*p\_traj\_vec[:,3]/(2\*np.pi)  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()      # 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 :

A 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.