République de Côte d'Ivoire

Institut National Polytechnique
Félix HOUPHOUËT-BOIGNY

ESI
Ecole Supérieure d'Industrie

RAPPORT DE ROBOTIQUE

SUJET :

SIMULATEUR DE ROBOT MANIPULATEUR 2D

Année académique : 2021-2022

Présenté par **ABOI Koman Guy-Parfait**, élève en deuxième année d'ingénieur en Science et Technologie de l'Information et de la Communication (STIC)

Enseignant : M. KONE Siriky, enseignant au DFR GEE

LISTE DES FIGURES

Figure 1: Les repères liés au robot 6
Figure 2: Interface d'utilisation 10
Figure 3:Fonctionnement de l'application 12
Figure 4: Cas d'utilisation avec champs non renseigné 13
Figure 5: Cas d'utilisation Point non atteignable 14

LISTES DES TABLEAUX

Table 1:Paramètres initiaux du robot 5

SOMMAIRE

LISTE DES FIGURES 0
LISTES DES TABLEAUX 2
SOMMAIRE 3
INTRODUCTION 4
I. Présentation de projet 4
II. Etude du projet 5
III. Réalisation du robot 10
CONCLUSION 23
REFERENCES 24
TABLE DES MATIERES 25

INTRODUCTION

Le progrès technologique est un phénomène imparable. Aujourd'hui, la technologie de pointe a inauguré une ère où les tâches manuelles sont confiées à plusieurs reprises à de nouveaux systèmes tels que les robots. Un robot est un appareil autonome capable d'effectuer certaines tâches sans assistance humaine (lors de son fonctionnement). La mise en place de tels systèmes nécessite un domaine de recherche, à savoir la robotique. Ce domaine est responsable du basculement manuel vers une machine autonome. Compte tenu des services rendus par ces systèmes, il existe une demande croissante de robots sur le marché. Pour nous immerger dans cet univers, après le cours de robotique, il nous a été demandé de mettre en place un simulateur de robotique. Il s'agit d'une application informatique qui simule le fonctionnement d'un robot en deux dimensions. Pour remplir notre mission, nous présenterons de façon détaillée le thème, après quoi nous aborderons l'étude théorique et enfin nous terminerons par la réalisation de notre application.

I. Présentation de projet

1. Présentation du contexte

Les cours de robotique enseignés sont conçus pour introduire la robotique. Cela se fera par l'apprentissage de diverses bases liées à la mise en œuvre des robots manipulateurs. Parmi eux, nous étudions le modèle géométrique direct (MGD) et le modèle géométrique inverse (IGM). Afin de capitaliser sur les connaissances théoriques acquises lors de l'apprentissage du modèle, il nous a été demandé d'implémenter une application 2D programmée en python pour simuler le comportement d'un robot manipulateur.

2. Présentation du thème

Le sujet du projet sur lequel nous travaillons est « Conception et réalisation d'un simulateur de robot manipulateur planaire simple ». Notre rôle est de construire un programme informatique capable de simuler le comportement robotique d'un manipulateur dont les caractéristiques seront décrites dans le cahier des charges. Ce robot doit être planaire (dans un plan) et sera chargé de se déplacer du point de départ au point d'arrivée lors de la traversée de points intermédiaires (étapes).

3. Cahier de charges

Le programme devra répondre aux spécifications suivantes :

- Comporter trois liens (L0, L1 et L2) dont les valeurs doivent être saisies par l'utilisateur
- Comporter deux angles (thêta 1 et 2) dont les valeurs doivent être saisies

- Se déplacer d'un point à un autre (points étant à sa portée)
- Déterminer la position du point de départ (point A) en fonction des paramètres entrées
- Déterminer et afficher les paramètres d'un point transitoire (le point Pi).

4. Objectifs du projet

L'objectif de ce projet est de simuler le fonctionnement d'un robot 2D en utilisant le langage python. Ce programme vise à nous initier à la robotique moderne

II. Etude du projet

1. Paramètres initiaux du robot

Table 1:Paramètres initiaux du robot

Symboles	Signification
L_0	Lien entre la base du robot et l'articulation A_1
L_1	Lien entre l'articulation A_1 et l'articulation A_2
L_2	Lien entre l'articulation A_2 et l'organe terminal
A_1	Articulation se trouvant à l'intersection de L0 et L1
A_2	Articulation se trouvant à l'intersection de L1 et L2
Θ_1 (degré)	Angle entre x_0 de R0 et x_1 de R1, correspondant à une rotation autour de z_1
Θ_2 (degré)	Angle entre x_1 de R1 et x_2 deR2 correspondant à une rotation autour de z_2

2. Construction graphique en 2D du robot

La construction du robot commence par la représentation de différents points de repère. Le cadre R0 connecté à la base du robot, le cadre R1 avec A1 comme origine et le cadre R2 avec A2 comme origine. Les axes étiquetés différemment suivent la configuration dans le schéma suivant :

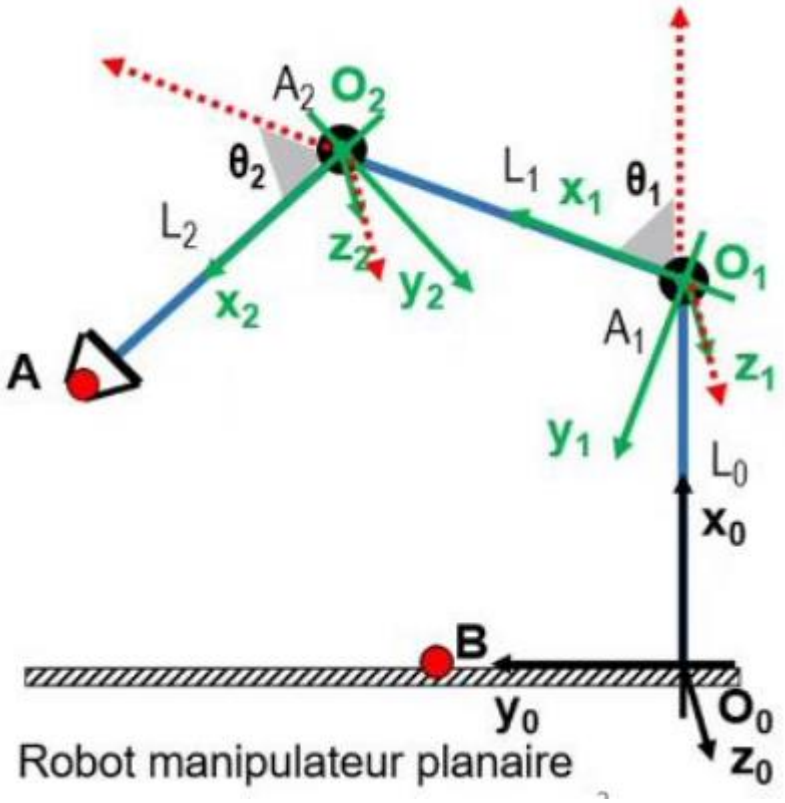

Figure 1: Les repères liés au robot

- Les matrices de passage
  - La matrice de passage du repère R1 au repère R0

$$^{0}T_{1} = \begin{pmatrix} C\Theta_1 & -S\Theta_1 & 0 & L_0 \\ S\Theta_1 & C\Theta_1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

  - La matrice de passage du repère R2 au repère R1

$$^{1}T_{2} = \begin{pmatrix} C\Theta_2 & -S\Theta_2 & 0 & L_1 \\ S\Theta_2 & C\Theta_2 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

  - La matrice de passage du repère R2 au repère R0

$$^{0}T_{2} = {}^{0}T_{1}.{}^{1}T_{2}$$

$$ {}^{0}T_2 = \begin{pmatrix} C\Theta_1 C\Theta_2 - S\Theta_1 S\Theta_2 & -C\Theta_1 S\Theta_2 - S\Theta_1 C\Theta_2 & 0 & L_1 C\Theta_1 + L_0 \\ S\Theta_1 C\Theta_2 + C\Theta_1 S\Theta_2 & -S\Theta_1 S\Theta_2 + C\Theta_1 C\Theta_2 & 0 & L_1 S\Theta_1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} $$

- Les coordonnées des articulations dans les différents repères
  - Articulation A_1 dans le repère R_0

$$ \begin{pmatrix} x_{A_0} \\ y_{A_0} \\ z_{A_0} \\ 1 \end{pmatrix}_{R_0} = \begin{pmatrix} L_0 \\ 0 \\ 0 \\ 1 \end{pmatrix} $$

  - Articulation A_2 dans le repère R_1

$$ \begin{pmatrix} x_{A_1} \\ y_{A_1} \\ z_{A_1} \end{pmatrix}_{R_1} = \begin{pmatrix} L_1 \\ 0 \\ 0 \end{pmatrix} $$

  - L'organe terminal A dans le repère R_2

$$ \begin{pmatrix} x_{A_2} \\ y_{A_2} \\ z_{A_2} \end{pmatrix}_{R_2} = \begin{pmatrix} L_2 \\ 0 \\ 0 \end{pmatrix} $$

3. Position initiale A de la pince du robot

Cela consiste à trouver les coordonnées du point A dans le repère R_0. Il s'agit alors du Modèle Géométrique Direct (MGD). Les matrices de passage étant déjà déterminées, on a :

$$ \begin{pmatrix} x_{A_0} \\ y_{A_0} \\ Z_{A_0} \\ 1 \end{pmatrix}_{R_0} = {}^{0}T_2 \begin{pmatrix} L_2 \\ 0 \\ 0 \\ 1 \end{pmatrix}_{R_2} $$

$$ \begin{pmatrix} x_{A_0} \\ y_{A_0} \\ Z_{A_0} \\ 1 \end{pmatrix}_{R_0} = \begin{pmatrix} L_2(C\Theta_1 C\Theta_2 - S\Theta_1 S\Theta_2) + L_1 C\Theta_1 + L_0 \\ L_2(S\Theta_1 C\Theta_2 + C\Theta_1 S\Theta_2) + L_1 S\Theta_1 \\ 0 \\ 1 \end{pmatrix}_{R_0} $$

4. Equation de la trajectoire de A et B

La pince du robot décrit une droite, l'équation de la trajectoire est donc de la forme : $y = ax + b$

$a = \frac{y_B - y_A}{x_B - x_A}$ et $b = y_B - ax_B$

$$y = \frac{y_B - y_A}{x_B - x_A} \; x + y_B - ax_B$$

5. Définition du pas selon l'axe x

Selon l'axe x, le pas vaut :

$$pasx = \frac{x_B - x_A}{nbre\ de\ pas}$$

6. Coordonnées des points P1, P2, …, Pn

Soit $P_i = P_1, P_2, \ldots, P_n$ une famille de points. Les P_i sont les points transitoires à atteindre pour parvenir à la destination B du robot. Ils sont des points transitoires dont les coordonnées se déterminent comme suit :

$$x_i = x_A - i * pasx \quad \text{et} \quad y_i = ax_i + b \quad \text{ou} \quad y_i = y_A - i * pasy$$

$$\text{Avec } pasy = \frac{y_B - y_A}{nbre\ de\ pas}$$

7. Déterminons Θ1 et Θ2 pour chaque point chaque point Pi

La détermination de ces angles ramène au Modèle Géométrique Inverse (MGI). Pour cela, nous devons résoudre un système de type 6 de la forme :

$$\begin{cases} WS\Theta_2 = XC\Theta_1 + YS\Theta_1 + Z_1 \\ WC\Theta_2 = XS\Theta_1 + YC\Theta_1 + Z_2 \end{cases}$$

Le rapprochement à notre étude nous donne le système suivant :

$$\begin{cases} L_2 S\Theta_2 = Y_i C\Theta_1 + (L_0 - X_i) S\Theta_1 \\ L_2 C\Theta_2 = Y_i S\Theta_1 + (L_0 - X_i) C\Theta_1 - L_1 \end{cases}$$

- Déterminons Θ_1

Déterminer Θ_1 d'un P_i revient à résoudre une équation de type 2 de la forme :

$$B_1 S\Theta_1 + B_2 C\Theta_1 = B_3$$

Où :

$$\begin{cases} B_1 = -2Y_i L_1 \\ B_2 = 2L_1(L_0 - X_i) \\ B_3 = L_2^2 - Y_i^2 - (L_0 - X_i)^2 - L_1^2 \end{cases}$$

- Si $B_3=0$

$$\Theta_1 = ATAN(-B_2, B_1)$$

$$\Theta_1' = \Theta_1 + 180$$

- Si $B_3 \neq 0$

$$\begin{cases} S\Theta_1 = \dfrac{B_3 B_1 + \varepsilon\sqrt{B_1^2 + B_2^2 - B_3^2}}{B_1^2 + B_2^2} \\ C\Theta_1 = \dfrac{B_3 B_2 - \varepsilon B_1\sqrt{B_1^2 + B_2^2 - B_3^2}}{B_1^2 + B_2^2} \end{cases}$$

D'où : $\Theta_1 = ATAN2(S\Theta_1, C\Theta_1)$ si $B_1^2 + B_2^2 - B_3^2 \geq 0$

- Déterminons Θ_2

Déterminer Θ_2 d'un P_i revient à résoudre un système d'équations de type 3 (forme 1), on obtient :

$$Y_1' = L_2 S\Theta_1$$

$$Y_2' = L_2 C\Theta_1$$

$$\Theta_1 = ATAN2\left(\frac{Y_1'}{L_2}, \frac{Y_2'}{L_2}\right) \quad \text{Avec} \quad L_2 \neq 0$$

8. Mouvement du robot pour chaque pas Pi

Pour chaque P_i les paramètres X_i, Y_i, Θ_1 et Θ_2 sont déterminés, ainsi il ne reste plus qu'à les positionner dans le graphe. Les liens étant fixes, les nouvelles positions de l'articulation A_2 dans le repère R_0 sont bien connues :

$$X_{A_2} = L1 * C\theta_1 + L0$$

$$Y_{A_2} = L1 * S\theta_1$$

9. Courbe de trajectoire de la droite

Il s'agit du segment suivi par la pince du robot lors de son déplacement. C'est donc le segment se trouvant entre la position initiale de la pince du robot (le point A) et le point P_i atteint par la pince (si $P_i = P_n$ alors nous sommes au point d'arrivé B). Cette trajectoire est donc un segment appartenant à l'équation de la trajectoire.

III. Réalisation du robot

1. Présentation de l'application

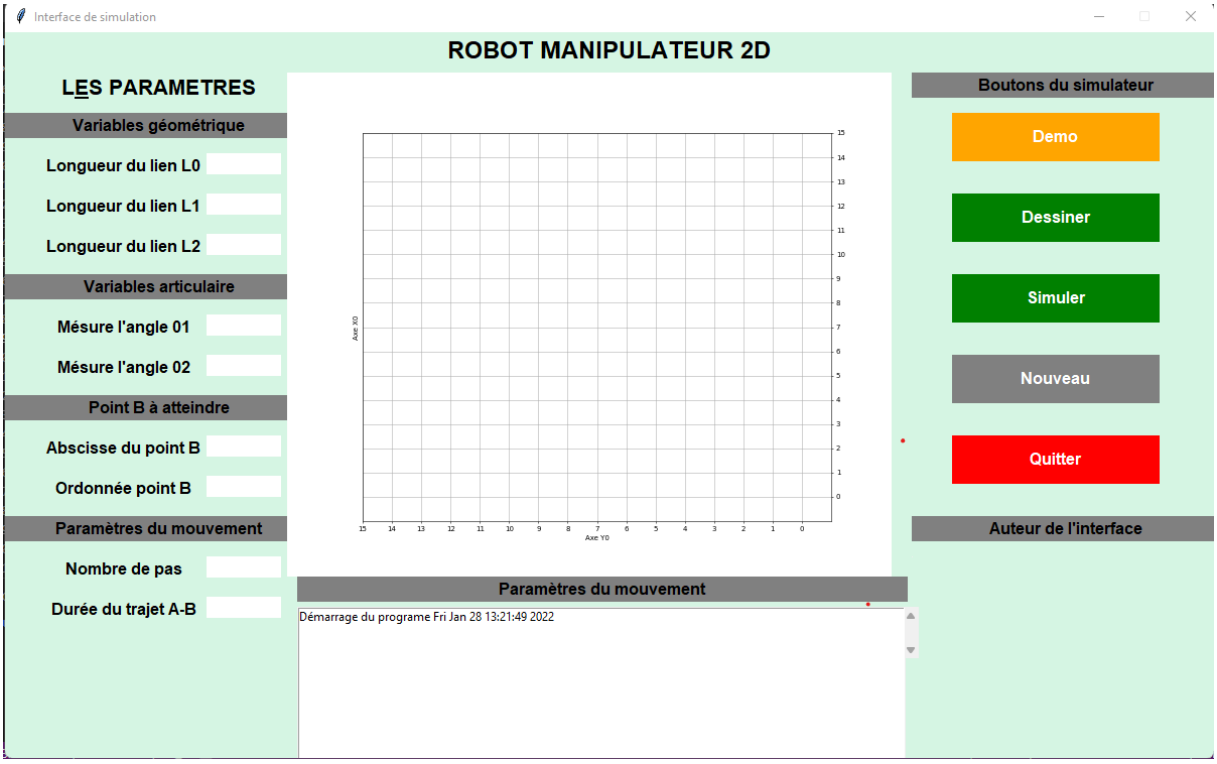

Figure 2: Interface d'utilisation

La figure ci-dessus présente l'interface principale de notre application. Elle se divise en quatre grandes parties.

La première partie, qui est située tout à gauche, présente les champs de saisie, d'affichage. Cette partie nommée **« Les paramètres »** est elle-même divisée en plusieurs sections. La section **« Variables géométrique »,** elle permet de saisir les informations sur la taille des liens du robot, l'utilisateur renseigne **L0, L1, L2.** La section **« Variables articulaire »,** elle permet de saisir les valeurs pour les deux angles au niveau des articulations en A2 et en A1. La section **« point B à atteindre »,** elle permet à l'utilisateur de saisir les coordonnées initiales du point B de destination. La section **« Paramètre du mouvement »,** elle permet de renseigner le nombre de pas et la durée de trajectoire A-B.

La deuxième partie, qui est le graphe (le repère), constitue le champ de dessin du robot ainsi que ses différentes configurations. Elle se situe au centre de l'application.

La troisième partie, tout à droite de l'interface, est composé des cinq 05 boutons définis par le cahier de charge. Le bouton **« Démo »** Permet de remplir automatiquement le champ des paramètres de simulation avec les valeurs par défaut stockées dans la base de données. Le bouton **« Dessiner »** Permet de dessiner le robot dans la zone de dessin à partir du contenu des champs des paramètres de simulation. Le bouton **« Simuler »** Permet de mettre le robot en mouvement, la pince allant du point A au point B conformément aux paramètres de simulation. Le bouton **« Nouveau »** Permet de vider les différents champs des paramètres de simulation et des informations de débogage. Le bouton **« Quitter »** Permet de se déconnecter puis fermer l'application.

La dernière partie, situé tout en bas au centre, permet de réaliser le débogage durant l'exécution du programme.

NB 1 : Les axes x et y sont inversés contrairement à ce que l'on a l'habitude de rencontrer. L'axe x est dirigé vers le haut tant dis que l'axe y est dirigé vers la gauche.

NB 2 : Il existe des contrôles sur les champs de saisie, le fonctionnement est tel que si l'on entre une valeur non attendue (par exemple, on entre de lettre à la place des chiffres) le programme détecte les erreurs et lance une alerte à l'utilisateur

2. Fonctionnalités de l'application

Cette application est riche en fonctionnalité. Les différentes fonctions encapsulées dans les boutons sont :

- « **Démo** » Permet de remplir automatiquement le champ des paramètres de simulation avec les valeurs par défaut stockées dans la base de données.
- « **Dessiner** » Permet de dessiner le robot dans la zone de dessin à partir du contenu des champs des paramètres de simulation
- « **Simuler** » Permet de mettre le robot en mouvement, la pince allant du point A au point B conformément aux paramètres de simulation.
- « **Nouveau** » Permet de vider les différents champs des paramètres de simulation et des informations de débogage.
- « **Quitter** » Permet de se déconnecter puis fermer l'application.

3. Fonctionnement de l'application

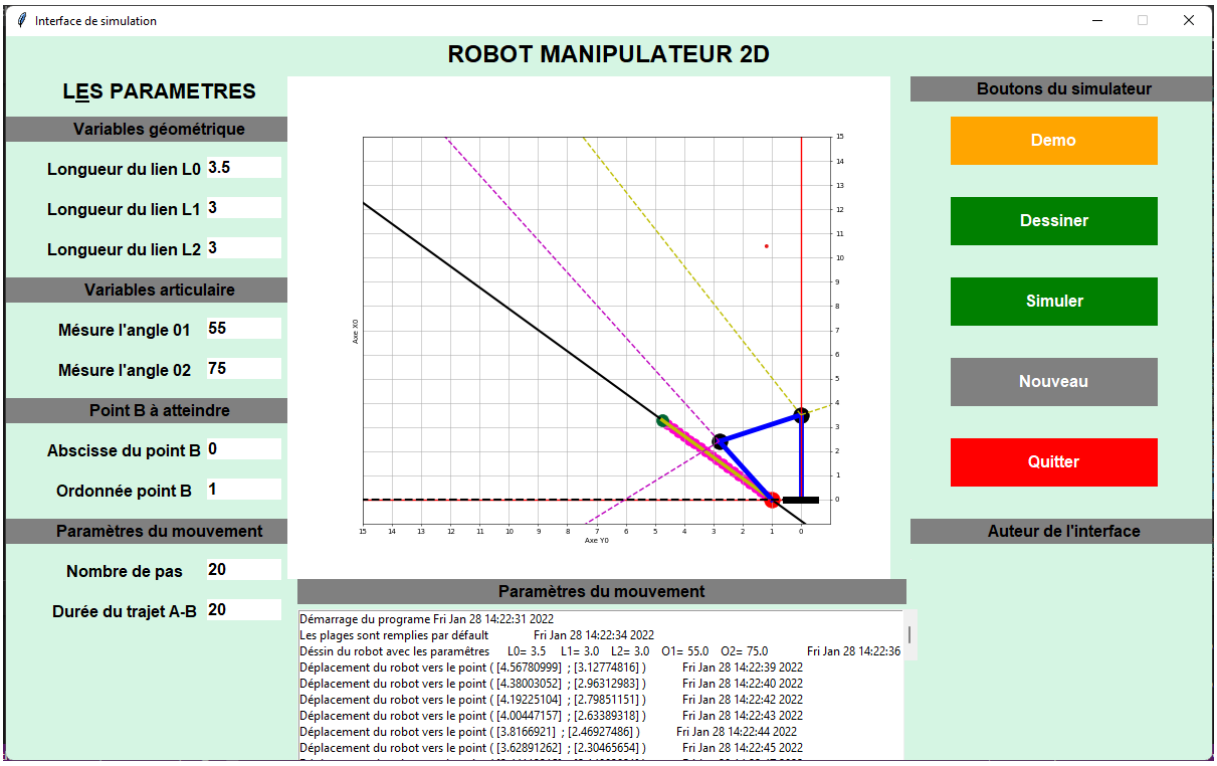

Figure 3:Fonctionnement de l'application

Il faut préciser qu'il existe plusieurs combinaisons dans lesquelles l'on peut faire fonctionner le robot. Parmi celles-ci, nous avons le fonctionnement de la figure

ci-dessus. Dans cette figure, le robot parcourt vingt (20) pas pour atteindre le point d'arrivée. Dans le code, on a :

```
etatBtnTrajectoire=True
etatBtnPas=True
etatBtnBip=False
etatBtnR0=True
etatBtnR1=True
etatBtnR2=True
```

On a donc autorisé le tracé de la trajectoire, la représentation des boutons de transition et les repères associer au point A1, A2 et à l'origine du repère. La vitesse de réalisation est de 1 pas/s.

Dans son fonctionnement général, Lorsqu'on ne renseigne pas un champ et qu'on désire dessiner le robot, une boite d'alerte apparait. Il en est de même lorsque le point n'est pas atteignable.

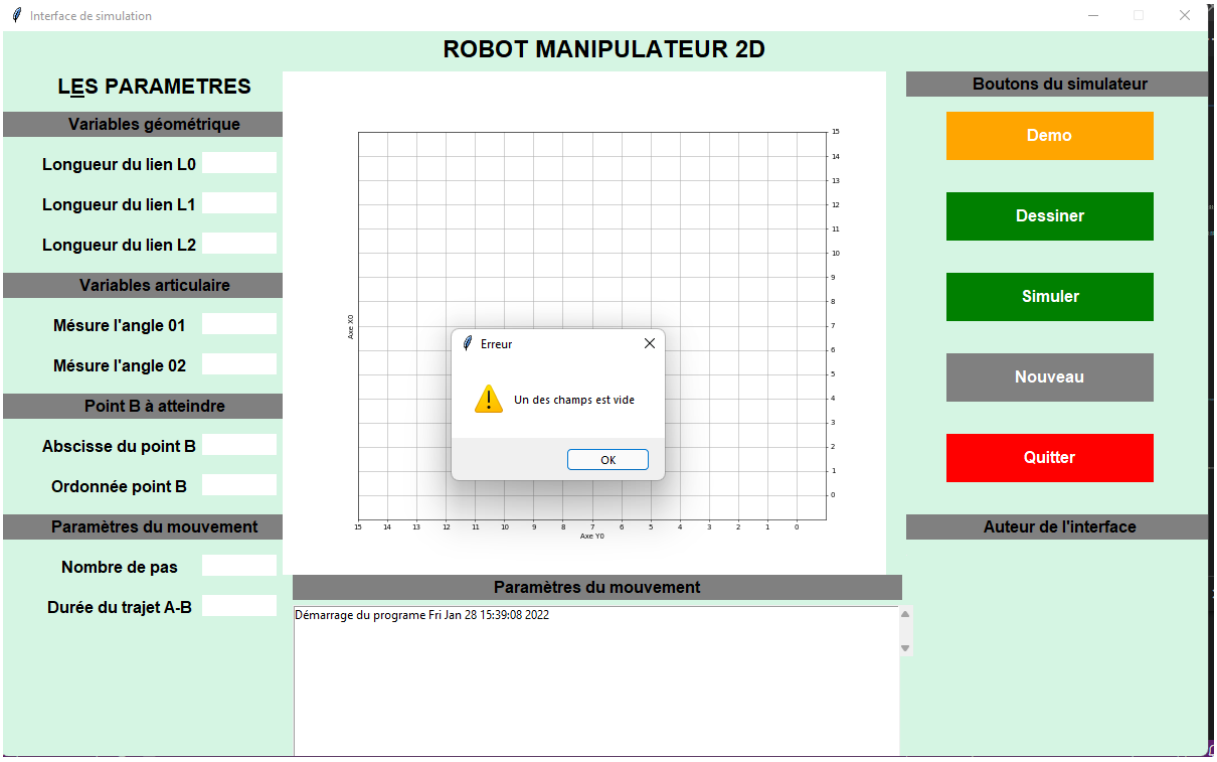

Figure 4: Cas d'utilisation avec champs non renseigné

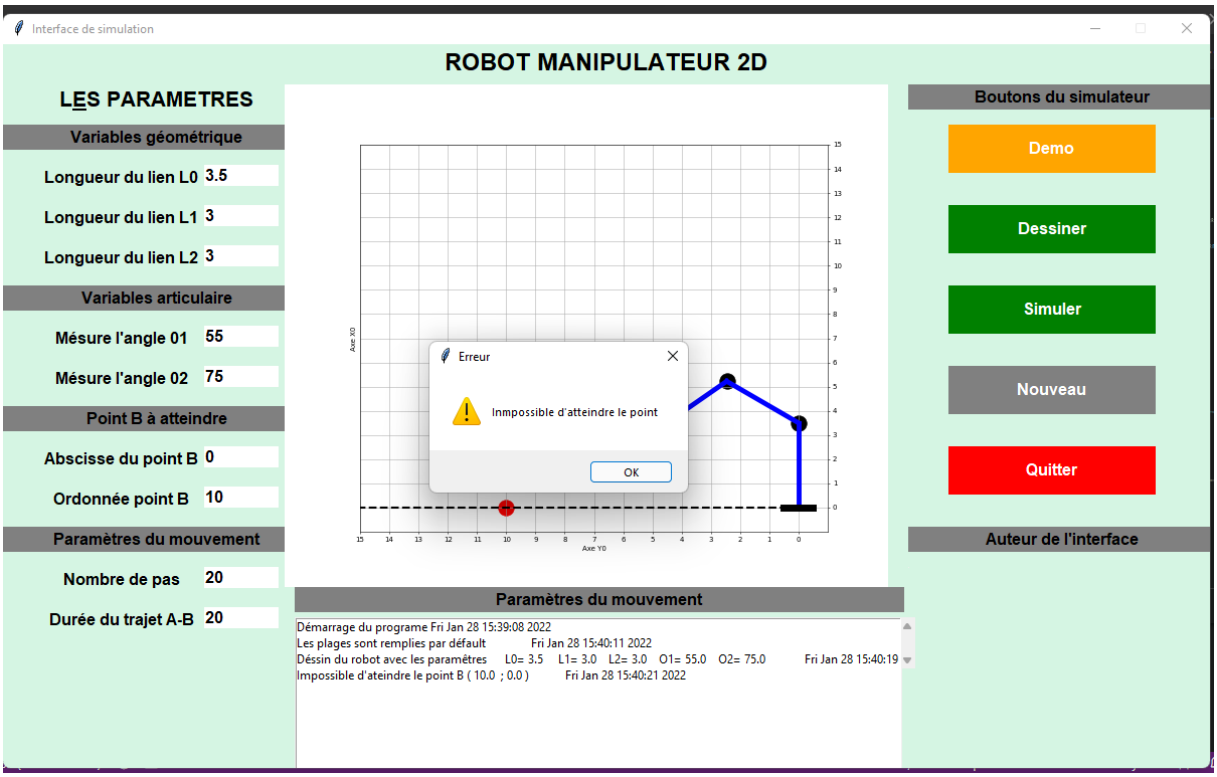

Figure 5: Cas d'utilisation Point non atteignable

NB : il faut bien remarquer que les axes x et y sont inversés contrairement à ce que l'on a l'habitude de rencontrer.

4. L'implémentation de quelques fonctions en python

Les librairies sont les programmes informatiques qui contiennent plusieurs méthodes servant à être utilisées dans un programme après leur importation.

- **Les différentes librairies importées**
  - **winsound** : utilisée pour les bips sonores.
  - **time** : utilisée pour mettre un programme en inactivité
  - **tkinter** : gérère les fenêtres graphiques en python
  - **math** : contient les fonctions mathématiques
  - **numpy** : contient des méthodes qui concernent les matrices
  - **matplotlib** : gère les repères et l'affichage graphique des fonctions mathématiques
  - **matplotlib.use("TkAgg")** : interface contenant le repère
  - **matplotlib.figure** : sorte de canevas qui contient l'espace de dessin du graphe (repère)
  - **FigureCanvasTkAgg** : L'interface graphique qui se couple à **tkinter** pour les dessins dans un repère
  - **Tkinter.messagebox :** Permet d'utiliser les boites d'alertes

```
-    import winsound
-    from tkinter import *
-    import tkinter.messagebox
-    import time
-    import matplotlib.pyplot as plt
-    import matplotlib
-    matplotlib.use("TkAgg")
-    import math as Math
-    import numpy as np
-    from matplotlib.figure import Figure
-    from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg
```

Avant la réalisation de nos objectifs, nous avons dû implémenter les calculs réalisés tout à l'heure. On récupère les valeurs saisie on calcule les matrices de passage entre les différents repère. De ce fait récupère les coordonnées du point A à la terminaison du robot

```
def calcul():

    L0 = recupValeurLien(txtL1)
    L1 = recupValeurLien(txtL2)
    L2 = recupValeurLien(txtL3)
    O1 = Math.radians(recupValeurAngle(txtTeta1))
    O2 = Math.radians(recupValeurAngle(txtTeta2))
    nbrePas = int(recupValeur(txtNbrePas))
    YB = recupValeurLien(txtYB)
    XB = recupValeurLien(txtXB)

    #LES MATRICES DE PASSAGE DIRECTE
    Mat0T1 = np.array([[Math.cos(O1),-
Math.sin(O1),0,L0],[Math.sin(O1),Math.cos(O1),0,0],[0,0,1,0],[0,0,0,1]])
    Mat1T2 = np.array([[Math.cos(O2),-
Math.sin(O2),0,L1],[Math.sin(O2),Math.cos(O2),0,0],[0,0,1,0],[0,0,0,1]])
    Mat0T2 = Mat0T1.dot(Mat1T2)
    A2=np.array([[L2],[0],[0],[1]])
    A21=np.array([[L1],[0],[0],[1]])
    A10=np.array([[L0],[0],[0],[1]])

    A0 = Mat0T2.dot(A2)

    A20 = Mat0T1.dot(A21)

    rotAngle = Math.degrees(Math.atan2(Mat0T2[[1],[0]], Mat0T2[[0],[0]]))
```

```
    Mat1T0 = np.array([[Math.cos(O1),Math.sin(O1),0,-L0*Math.cos(O1)],[-
Math.sin(O1),Math.cos(O1),0,L0*Math.sin(O1)],[0,0,1,0],[0,0,0,1]])
    Mat2T1 = np.array([[Math.cos(O2),Math.sin(O2),0,-L0*Math.cos(O2)],[-
Math.sin(O2),Math.cos(O2),0,L1*Math.sin(O2)],[0,0,1,0],[0,0,0,1]])
   # txtXA.insert(0,float(A0[1]))
    #txtYA.insert(0,float(A0[0]))
    return [nbrePas, L0, A20, A0, YB, XB, L1, L2]
```

L'implémentation des différentes fonctions de l'application est :

- **« Démo »**

Cette fonctionnalité est représenté par la fonction demo()

```
def demo():
    txtL1.delete(0,END)
    txtL1.insert(0,"3.5")
    txtL2.delete(0,END)
    txtL2.insert(0,"3")
    txtL3.delete(0,END)
    txtL3.insert(0,"3")
    txtTeta1.delete(0,END)
    txtTeta1.insert(0,"55")
    txtTeta2.delete(0,END)
    txtTeta2.insert(0,"75")
    txtNbrePas.delete(0,END)
    txtNbrePas.insert(0,"20")
    txtYB.delete(0,END)
    txtYB.insert(0,"1")
    txtXB.delete(0,END)
    txtXB.insert(0,"0")
    txtL9.delete(0,END)
    txtL9.insert(0,"20")
    listbox.insert(END, "Les plages sont remplies par défault            "+ str(time.ctime()) )
```

Pour les démos, on commence par la suppression des valeurs saisie dans les différentes zones et on les remplace par la valeur enregistrée dans la base de données. Au même moment on renseigne la liste de débogage en rajoutant l'instant d'appel de la fonction.

- **« Dessiner »**

On réalise les opérations de calculs sont effectués avant le dessin.

Cette fonctionnalité est représenté par la fonction fncInialisation()

```
def fncInitialisation():
    plot.cla()
    global I
    I=1
    result = calcul()
    L0 = result[1]
    A20 = result[2]
    A0 = result[3]
    YB = result[4]
    XB = result[5]
    nbreMaxPas = result[0]

    plot.set_xlabel('Axe Y0')
    plot.set_ylabel('Axe X0')
    plot.yaxis.set_ticks_position('right')
    plot.set_xticks(range(16))
    plot.set_yticks(range(16))
    plot.set_xlim((15,-1))
    plot.set_ylim((-1, 15))
    plot.grid(True)

    #tracer L0
    plot.plot([0.0,0.0],[0.0,L0],"b-",lw=7)
    #tracer L1
    plot.plot([0.0,A20[1]],[L0,A20[0]],"b-",lw=7)
    #tracer L2
    plot.plot([A20[1],A0[1]],[A20[0],A0[0]],"b-",lw=7)
    #Le point A
    plot.scatter([A0[1]], [A0[0]], s =500, color = 'red')
    #Le point B
    plot.scatter([YB], [XB], s =500, color = 'red')
    #Les Articulations
    plot.scatter([0.0], [L0], s =500, color = 'black')
    plot.scatter([A20[1]], [A20[0]], s =500, color = 'black')
    #La base et le sol
    plot.plot([-0.5,0.5],[0.0,0.0],"k-",lw=10)
    plot.plot([0.0,15.0],[0.0,0.0],"k--",lw=3)

    if etatBtnTrajectoire==True:
        XA0 = result[3][0]
        XB = result[5]
        YA0 = result[3][1]
        YB = result[4]
        a = (YA0-YB)/(XA0-XB)
        b = YB-a*XB
        x=range(-100,101)
        y = a*x + b
        #Trace la droite
        plot.plot(y,x,"k-",lw=3)
```

```
    plot.grid(True)
    graphique.draw()
    L1 = recupValeurLien(txtL2)
    L2 = recupValeurLien(txtL3)
    O1 = recupValeurAngle(txtTeta1)
    O2 = recupValeurAngle(txtTeta2)
    listbox.insert(END, "Déssin du robot avec les paramêtres      L0= " + str(L0) + "     L1= "+ str(L1) +
"    L2= " + str(L2) + "     O1= " + str(O1)+ "     O2= " + str(O2)+"            "+ str(time.ctime()) )
```

On récupère les valeurs retournées par calcul, on trace les différents segments du robot.

- « **Simuler** »

Cette fonctionnalité est représenté par la fonction avance()

Apres récupération des valeurs saisies, on applique la fonction calcul() est appliquée, on récupère les valeurs de sortie ce sont la longueur L0, la longueur L1, la longueur L2, les coordonnées de B(YB, XB), les coordonnées de l'articulation A2 dans le repère R0. Ensuite, on calcule la vitesse de simulation en faisant la durée de simulation par le nombre de pas. On vérifie si le point est atteignable, sinon on avertit l'utilisateur. Sinon on entre dans une boucle.

On commence par effacer l'ancienne figure, on définit les propriétés du nouveau graph puis on détermine le pas selon x et selon y. Des lors on applique la méthode de Paul, on détermine donc teta1 et teta2 pour chaque point de passage (intermédiaire). Après cela, on calcule les paramètres du graphe pour chaque point et le dessine

```
def avance():


    global X_Pi, Y_Pi, etatBtnPas, etatBtnBip, etatBtnR0, graphique, plot
    X_Pi = []
    Y_Pi = []
    result = calcul()
    nbrePas = result[0]
    L0 = result[1]
    L1 = result[6]
    L2 = result[7]
    YB = result[4]
    A0 = result[3]
    XB = result[5]
    A20 = result[2]
    tps = recupValeur(txtL9)
    vitesse= tps/nbrePas

    LT = L2+L1
```

```
    if(LT<YB or LT<XB):
        listbox.insert(END, "Impossible d'ateindre le point B ( "+ str(YB)+"  ; "+ str(XB)+" )          " +
str(time.ctime()))
        tkinter.messagebox.showwarning(title="Erreur",message="Inmpossible d'atteindre le point")
        return
    #vitesse = float(1/(int(txtL9.get())))
    #fig = plt.figure(figsize=(12, 10), dpi=50)

    ims = []

    for i in range(1,nbrePas+1):
        #plt.cla()
        plot.clear()
        #Definir les proprietes du nouveau graph
        plot.set_xlabel('Axe Y0')
        plot.set_ylabel('Axe X0')
        plot.yaxis.set_ticks_position('right')
        plot.set_xticks(range(16))
        plot.set_yticks(range(16))
        plot.set_xlim((15,-1))
        plot.set_ylim((-1, 15))
        plot.grid(True)
        #plt.xlim(15, -1)
        #plt.ylim(-1, 15)
        #Distance X entre deux pas
        disXPas = (XB-A0[0])/nbrePas
        if disXPas<0:
            disXPas = -disXPas
        #Distance Y entre deux pas
        disYPas = (YB-A0[1])/nbrePas
        if disYPas<0:
            disYPas = -disYPas
        if XB>=A0[0] :
            Xi = A0[0]+i*disXPas
        else:
            Xi = A0[0]-i*disXPas

        if YB>A0[1]:
            Yi = A0[1]+i*disYPas
        else:
            Yi = A0[1]-i*disYPas

        #LES CALCULS ---------------------------------->
        B1 = -2*Yi*L1
        B2 = 2*L1*(L0-Xi)
        B3 = L2**2-Yi**2-(L0-Xi)**2-L1**2
        teta_1=0
        teta_2=0
        SO1 = 0
```

```
CO1 = 0
epsi = 1
if B3==0 :
    teta_1 = Math.degrees(Math.atan2(-B2,B1))
else:
    if ((B1**2+B2**2-B3**2)>=0) :
        SO1 = (B3*B1+epsi*B2*Math.sqrt(B1**2+B2**2-B3**2))/(B1**2+B2**2)
        CO1 = (B3*B2-epsi*B1*Math.sqrt(B1**2+B2**2-B3**2))/(B1**2+B2**2)
        teta_1 = Math.degrees(Math.atan2(SO1,CO1))
    else:
        #conf.configure(bg="red")
        break
Yn1 = L2*SO1
Yn2 = L2*CO1
if L2!=0 :
    teta_2 = Math.degrees(Math.atan2(Yn1/L2,Yn2/L2))
else:
    #conf.configure(bg="red")
    break
#conf.configure(bg="green")
XA1i =L1*Math.cos(Math.radians(teta_1))+L0
YA1i =L1*Math.sin(Math.radians(teta_1))
#Position des Pi

#Trajectoire
if etatBtnTrajectoire==True:
    XA0 = result[3][0]
    XB = result[5]
    YA0 = result[3][1]
    YB = result[4]
    a = (YA0-YB)/(XA0-XB)
    b = YB-a*XB
    x=range(-100,101)
    y = a*x + b
    #Trace la droite
    plot.plot(y,x,"k-",lw=3)
    #Droite entre A et Pi
    plot.plot([A0[1],Yi],[A0[0],Xi],"y-",lw=5)
#sauvegarde les coordonnees des Pi
X_Pi.append(Xi)
Y_Pi.append(Yi)
#Les Pas
if etatBtnPas==True:
    for j in range(0,len(X_Pi)) :
        plot.scatter([Y_Pi[j]], [X_Pi[j]], s =200, color = '#FF00CC')
#for j in range(0,X_Pi.len()):
 #   print(X_Pi, Y_Pi)
#tracer L0
plot.plot([0.0,0.0],[0.0,L0],"b-",lw=7)
```

```
#tracer L1
plot.plot([0.0,YA1i],[L0,XA1i],"b-",lw=7)
#tracer L2
plot.plot([YA1i,Yi],[XA1i,Xi],"b-",lw=7)
#Point Pi
plot.scatter([Yi], [Xi], s =500, color = '#FF0000')
#Point A0
plot.scatter([0], [L0], s =500, color = 'black')
#Point A2
plot.scatter([YA1i], [XA1i], s =500, color = 'black')
if i!=0:
    #Le point A
    plot.scatter([A0[1]], [A0[0]], s =300, color = '#006633')
else:
    #Le point A
    plot.scatter([A0[1]], [A0[0]], s =500, color = '#FF0000')
if i==nbrePas:
    #Le point B
    plot.scatter([YB], [XB], s =300, color = '#FF0000')
else:
    #Le point B
    plot.scatter([YB], [XB], s =300, color = '#00FF33')
#Le repere R0
if etatBtnR0==True:
    plot.plot([0.0,0.0],[0.0,15.0],"r-",lw=2)
    plot.plot([0.0,15.0],[0.0,0.0],"r-",lw=2)
#Le repere R1
if etatBtnR1==True:
    m=(YA1i-0)/(XA1i-L0)
    c=YA1i-m*XA1i
    u = m*(16)+c
    plt.plot([0.0,u],[L0,16.0],"y--",lw=2)
    l = (-1/m)*(16)+YA1i+(1/m)*XA1i
    plot.plot([0.0,l],[L0,16.0],"y--",lw=2)
#Le repere R2
if etatBtnR2==True:
    m=(YA1i-Yi)/(XA1i-Xi)
    c=YA1i-m*XA1i
    u = m*16+c
    plot.plot([YA1i,u],[XA1i,16.0],"m--",lw=2)
    l = (-1/m)*(-16)+YA1i+(1/m)*XA1i
    plot.plot([YA1i,l],[XA1i,-16.0],"m--",lw=2)
#Le sol
plot.plot([-0.5,0.5],[0.0,0.0],"k-",lw=10)
plot.plot([0.0,15.0],[0.0,0.0],"k--",lw=3)
plot.grid(True)
graphique.draw()
graphique.get_tk_widget().pack()
schema.canvas.draw()
```

```
        schema.canvas.flush_events()
        listbox.insert(END, "Déplacement du robot vers le point ( "+ str(Yi)+"  ; "+ str(Xi)+" )          " +
str(time.ctime()))
        if etatBtnBip==True:
            #Le Bip
            winsound.Beep(440, 250)
        time.sleep(vitesse)
```

On termine en remplissant la zone de débogage pour chaque point de passage Pi

- « **Nouveau** »

Cette fonctionnalité est représenté par la fonction nouveau()

Elle consiste à effacer le contenu de tous champs de saisies. Et on renseigne la liste de débogage.

```
def nouveau():
    txtL1.delete(0,END)
    txtL2.delete(0,END)
    txtL3.delete(0,END)
    txtTeta1.delete(0,END)
    txtTeta1.delete(0,END)
    txtTeta2.delete(0,END)
    txtNbrePas.delete(0,END)
    txtYB.delete(0,END)
    txtXB.delete(0,END)
    txtL9.delete(0,END)
    listbox.insert(END, "Les valeurs des champs ont été reinitialisées          " + str(time.ctime()))
```

- « **Quitter** »

Cette fonctionnalité est représenté par la fonction interface.quit

CONCLUSION

Le but de ce projet est d'implémenter un programme informatique qui simule le fonctionnement d'un robot planaire. Pour y parvenir, nous avons mené une étude théorique du projet, qui nous a fourni des modèles mathématiques (équations) pour résoudre différents points du cahier des charges. Les équations mathématiques sont ensuite implémentées dans un programme informatique codé en python. La section mise en œuvre de notre rapport présente la mise en œuvre de l'application et une description de ses fonctionnalités. L'application développée est livrée avec ce document, nous pouvons donc constater que nous avons respecté nos spécifications.

REFERENCES

[1] https://www.w3schools.com/python/python_try_except.asp

[2] https://webdevdesigner.com/q/play-simple-beep-with-python-without-external-library-30422/

[3] http://www.proftnj.com/RGB3.htm

[4] https://www.alloprof.qc.ca/fr/eleves/bv/mathematiques/l-equation-de-droites-paralleles-ou-perpendicula-m1310

TABLE DES MATIERES

Table des matières

LISTE DES FIGURES 1
LISTES DES TABLEAUX 2
SOMMAIRE 3
INTRODUCTION 4
I. Présentation de projet 4
1. Présentation du contexte 4
2. Présentation du thème 4
3. Cahier de charges 4
4. Objectifs du projet 5
II. Etude du projet 5
1. Paramètres initiaux du robot 5
2. Construction graphique en 2D du robot 5
3. Position initiale A de la pince du robot 7
4. Equation de la trajectoire de A et B 7
5. Définition du pas selon l'axe x 8
6. Coordonnées des points P1, P2, …, Pn 8
7. Déterminons Θ1 et Θ2 pour chaque point chaque point Pi 8
8. Mouvement du robot pour chaque pas Pi 10
9. Courbe de trajectoire de la droite 10
III. Réalisation du robot 10
1. Présentation de l'application 10
2. Fonctionnalités de l'application 12
3. Fonctionnement de l'application 12
4. L'implémentation de quelques fonctions en python 14
CONCLUSION 23
REFERENCES 24
TABLE DES MATIERES 25