TP1 Robotique Forward/Inverse Kinematics

grosse grosse galère d'installation, réussi a installer le robot puis le robotokinematics avant de créer un kernel utilisable avec jupyter et enfin avoir un environnement pas trop dégueu

```
!which python
!which pip
```

```
/home/sedrenn/miniconda3/bin/python
/home/sedrenn/miniconda3/bin/pip
```

1 Tutoriel d'installation de la bibliothèque :

installation de la bibliothèque " le robot " nécessaire

```
import sys
sys.path.append('/home/sedrenn/lerobot-kinematics/lerobot-kinematics')
print(sys.executable)
```

```
/home/sedrenn/miniconda3/envs/lerobot/bin/python
```

```
import lerobot
print("lerobot OK ->", lerobot.__version__)
```

```
lerobot OK -> 0.4.2
```

2 Installation de la bibliothèque lerobot-kinematics

```
import lerobot_kinematics
```

3 Tester vos simulations/robots si tout fonctionne

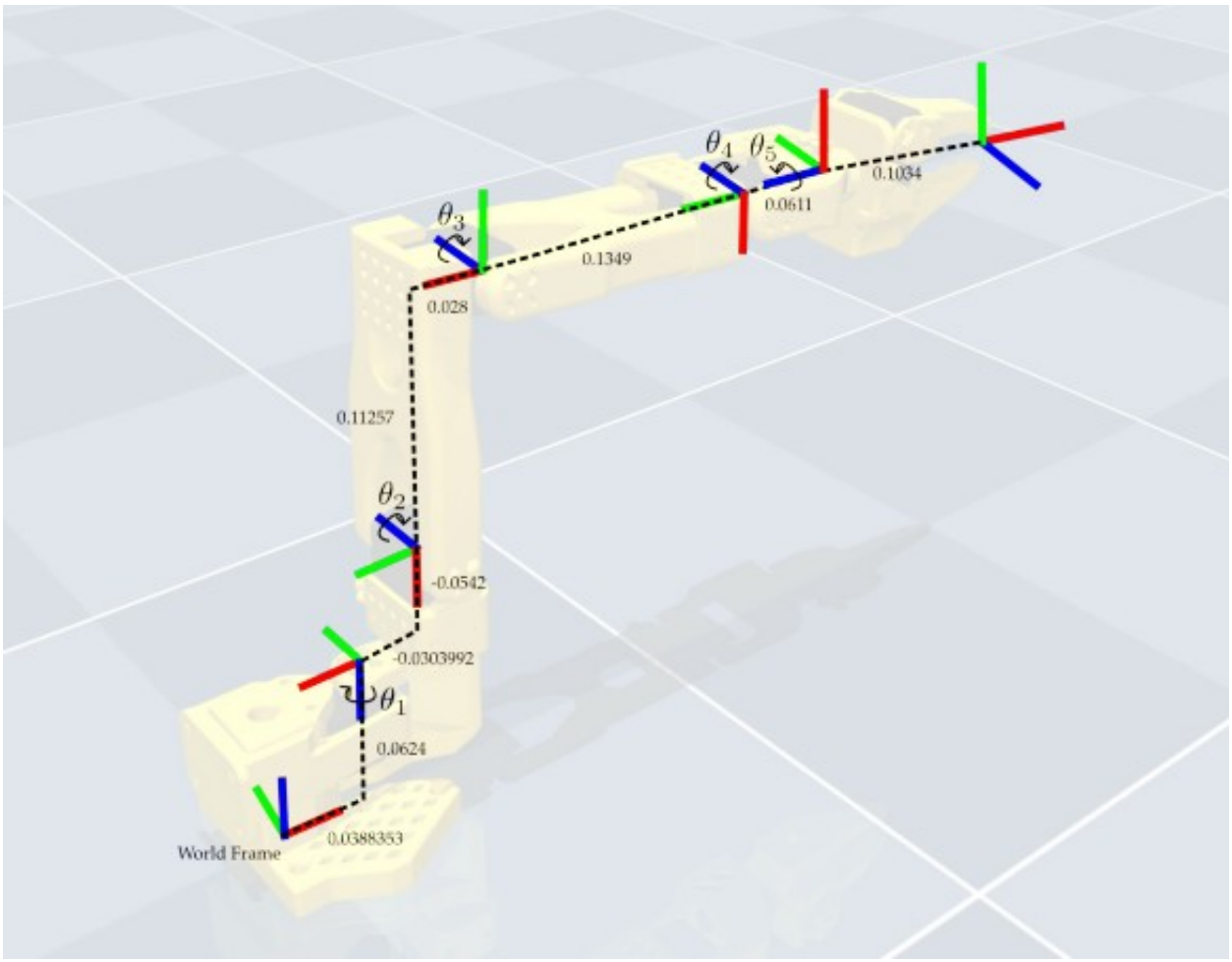
4 L'objectif de ce TP1 est de construire le modèle cinématique direct et inverse du robot SO101.

MODELE CINEMATIQUE DIRECT

5 voici les cadres de référence du robot SO-101

– Donnez le tableau de Denavitt-Hartenberg du robot
– Donnez les matrices de transformation associées pour chacune des articulations

On se fie a cette image pour en déduire le tableau DH>



θ (theta, rotation autour de z_i-1)	d (décalage le long de z_i-1)	a (décalage le long de x_i)	α (alpha, rotation autour de x_i)
0	0.0624	-0.0388	-180°
θ_1	-0.0542	-0.030	-90°
θ_2	0	-0.028	0°
θ_3 - 90°	0	0.11257	0°
θ_4 + 180°	0	0	90°
θ_5 - 90°	-0.1349 -0.0611 -0.1034	0	0°

Signification des colonnes :

- **θ (theta)** : rotation autour de l'axe **z_i-1**
- **d** : décalage le long de l'axe **z_i-1**
- **a** : décalage le long de l'axe **x_i** (longueur du lien)
- **α (alpha)** : rotation autour de l'axe **x_i**

```
### définition des variables ###

# Lien 0
theta0 = 0
d0 = 0.0624
a0 = -0.0388
alpha0 = -180

# Lien 1
# theta1 = variable (θ1)
d1 = -0.0542
a1 = -0.030
alpha1 = -90

# Lien 2
# theta2 = variable (θ2)
d2 = 0
a2 = -0.028
alpha2 = 0

# Lien 3
# theta3 = variable (θ3 - 90°)
d3 = 0
a3 = 0.11257
alpha3 = 0

# Lien 4
# theta4 = variable (θ4 + 180°)
d4 = 0
a4 = 0
alpha4 = 90

# Lien 5
# theta5 = variable (θ5 - 90°)
d5 = -0.1349 - 0.0611 - 0.1034
a5 = 0
alpha5 = 0
```

Matrice de transformation homogène Une matrice de transformation homogène s'écrit :

T =\begin{bmatrix} R & p \ 0;0;0 & 1 \end{bmatrix}

où :

- (R) = matrice de rotation (3 \times 3)
- (p = (x, y, z)^T) = vecteur position

La dernière ligne ([0;0;0;1]) permet d'effectuer les produits matriciels successifs en coordonnées homogènes.

```python
import sympy as sp

# Définir les variables symboliques pour les angles
theta1, theta2, theta3, theta4, theta5 = sp.symbols('theta1 theta2
theta3 theta4 theta5')

# Fonction pour générer la matrice DH symbolique
def dh_matrix(a, alpha, d, theta):
    """
    Retourne la matrice de transformation homogène basée sur les
paramètres DH
    """
    return sp.Matrix([
        [sp.cos(theta), -sp.sin(theta)*sp.cos(alpha),
sp.sin(theta)*sp.sin(alpha), a*sp.cos(theta)],
        [sp.sin(theta),  sp.cos(theta)*sp.cos(alpha), -
sp.cos(theta)*sp.sin(alpha), a*sp.sin(theta)],
        [0,              sp.sin(alpha),               sp.cos(alpha),
d],
        [0,              0,                           0,
1]
    ])

# Conversion des degrés en radians
deg2rad = sp.pi / 180

# DH parameters: [a, alpha, d, theta]
dh_params = [
    [0.0624, -0.0388, -180, 0],
    [-0.0542, -0.030, -90, theta1],
    [0, -0.028, 0, theta2],
    [0, 0.11257, 0, theta3 - 90*deg2rad],
    [0, 0, 90, theta4 + 180*deg2rad],
    [-0.1349, -0.0611, -0.1034, theta5 - 90*deg2rad]
]


# Calcul et affichage de chaque matrice T_{i/i+1}
T_total = sp.eye(4)
for i, param in enumerate(dh_params):
    a, alpha, d, theta = param
    alpha = alpha * deg2rad

    theta_expr = sp.expand_trig(theta)
    T = dh_matrix(a, alpha, d, theta_expr)
    T_total = T_total * T

    print(f"T_{i}/{i+1} = ")
    sp.pprint(T.evalf(4), use_unicode=True)  # <<< affichage arrondi
    print("\n")
```

T_0/1 =

$$\begin{bmatrix} 1.0 & 0 & 0 & 0.0624 \\ 0 & 1.0 & 0.0006772 & 0 \\ 0 & -0.0006772 & 1.0 & -180.0 \\ 0 & 0 & 0 & 1.0 \end{bmatrix}$$

T_1/2 =

$$\begin{bmatrix} \cos(\theta_1) & -1.0 \cdot \sin(\theta_1) & -0.0005236 \cdot \sin(\theta_1) & -0.0542 \cdot \cos(\theta_1) \\ \sin(\theta_1) & 1.0 \cdot \cos(\theta_1) & 0.0005236 \cdot \cos(\theta_1) & -0.0542 \cdot \sin(\theta_1) \\ 0 & -0.0005236 & 1.0 & -90.0 \\ 0 & 0 & 0 & 1.0 \end{bmatrix}$$

T_2/3 =

$$\begin{bmatrix} \cos(\theta_2) & -1.0 \cdot \sin(\theta_2) & -0.0004887 \cdot \sin(\theta_2) & 0 \\ \sin(\theta_2) & 1.0 \cdot \cos(\theta_2) & 0.0004887 \cdot \cos(\theta_2) & 0 \\ 0 & -0.0004887 & 1.0 & 0 \\ 0 & 0 & 0 & 1.0 \end{bmatrix}$$

T_3/4 =

$$\begin{bmatrix} \sin(\theta_3) & 1.0 \cdot \cos(\theta_3) & -0.001965 \cdot \cos(\theta_3) & 0 \\ -\cos(\theta_3) & 1.0 \cdot \sin(\theta_3) & -0.001965 \cdot \sin(\theta_3) & 0 \\ 0 & 0.001965 & 1.0 & 0 \\ 0 & 0 & 0 & 1.0 \end{bmatrix}$$

T_4/5 =

$$\begin{bmatrix} -\cos(\theta_4) & \sin(\theta_4) & 0 & 0 \\ -\sin(\theta_4) & -\cos(\theta_4) & 0 & 0 \\ 0 & 0 & 1.0 & 90.0 \\ 0 & 0 & 0 & 1.0 \end{bmatrix}$$

T_5/6 =

```
⎡sin(θ₅)   1.0·cos(θ₅)  0.001066·cos(θ₅)  -0.1349·sin(θ₅)⎤
⎢                                                       ⎥
⎢-cos(θ₅)  1.0·sin(θ₅)  0.001066·sin(θ₅)  0.1349·cos(θ₅) ⎥
⎢                                                       ⎥
⎢   0       -0.001066         1.0              -0.1034   ⎥
⎢                                                       ⎥
⎣   0           0              0                 1.0     ⎦
```

6 écrire un programme python qui calcule la position du end-effector en fonction des configurations du bras robotique en calculant la matrice de transformation globale dépendante des cadres de référence.

```python
import time
import mujoco
import numpy as np

def Rx(thetadeg):
    thetarad = np.deg2rad(thetadeg)
    c = np.cos(thetarad)
    s = np.sin(thetarad)
    return np.array([[1, 0, 0],
                     [0, c, -s],
                     [0, s, c]])

def Ry(thetadeg):
    thetarad = np.deg2rad(thetadeg)
    c = np.cos(thetarad)
    s = np.sin(thetarad)
    return np.array([[c, 0, s],
                     [0, 1, 0],
                     [-s, 0, c]])

def Rz(thetadeg):
    thetarad = np.deg2rad(thetadeg)
    c = np.cos(thetarad)
    s = np.sin(thetarad)
    return np.array([[c, -s, 0],
                     [s, c, 0],
                     [0, 0, 1]])

def get_gw1(theta1_deg):
    displacement = (0.0388353, 0.0, 0.0624)
    rotation = Rz(180) @ Rx(180) @ Rz(theta1_deg)
    pose = np.block([[rotation, np.array(displacement).reshape(3,1)],
```

```
[0, 0, 0, 1]])
    return pose


def get_g12(theta1_deg):
    displacement = (a1, 0.0, d1)
    rotation = Rz(theta1_deg) @ Rx(alpha1)
    pose = np.block([[rotation, np.array(displacement).reshape(3,1)],
[0, 0, 0, 1]])
    return pose

def get_g23(theta2_deg):
    displacement = (a2, 0.0, d2)
    rotation = Rz(theta2_deg) @ Rx(alpha2)
    pose = np.block([[rotation, np.array(displacement).reshape(3,1)],
[0, 0, 0, 1]])
    return pose


def get_g34(theta3_deg):
    displacement = (a3, 0.0, d3)
    rotation = Rz(theta3_deg - 90) @ Rx(alpha3)
    pose = np.block([[rotation, np.array(displacement).reshape(3,1)],
[0, 0, 0, 1]])
    return pose

def get_g45(theta4_deg):
    displacement = (a4, 0.0, d4)
    rotation = Rz(theta4_deg + 180) @ Rx(alpha4)
    pose = np.block([[rotation, np.array(displacement).reshape(3,1)],
[0, 0, 0, 1]])
    return pose

def get_g5t():
    displacement = (a5, 0.0, d5)
    rotation = np.eye(3)
    pose = np.block([
        [rotation, np.array(displacement).reshape(3, 1)],[0, 0, 0,
1]])
    return pose
# ---- Transformation globale ----

def get_end_effector_pose(theta1, theta2, theta3, theta4, theta5):
    g_w1 = get_gw1(theta1)
    g_12 = get_g12(theta1)
    g_23 = get_g23(theta2)
    g_34 = get_g34(theta3)
    g_45 = get_g45(theta4)
    g_4e = get_g5t()
```

```
    g_we = g_w1 @ g_12 @ g_23 @ g_34 @ g_45 @ g_4e
    return g_we

    # Produit matriciel des transformations
    g_we = g_w1 @ g_12 @ g_23 @ g_34 @ g_4e
    return g_we
```

Test fonction get end effector pose

```
 # Valeurs choisies pour le test
theta1 = 30.0
theta2 = -15.0
theta3 = 60.0
theta4 = -90.0
theta5 = 45.0

print("\n===== TEST DES TRANSFORMATIONS =====")
print("θ1 =", theta1, " | θ2 =", theta2, " | θ3 =", theta3," | θ4 =",
theta4, " | θ5 =", theta5)

# Transformation totale
g_we = get_end_effector_pose(theta1, theta2, theta3, theta4, theta5)

print("\n===== MATRICE FINALE END-EFFECTOR g_we =====\n", g_we)
```

```
===== TEST DES TRANSFORMATIONS =====
θ1 = 30.0  | θ2 = -15.0  | θ3 = 60.0  | θ4 = -90.0  | θ5 = 45.0

===== MATRICE FINALE END-EFFECTOR g_we =====
 [[-3.53553391e-01  8.66025404e-01 -3.53553391e-01  1.30302812e-01]
 [ 6.12372436e-01  5.00000000e-01  6.12372436e-01 -1.28426378e-01]
 [ 7.07106781e-01  4.32978028e-17 -7.07106781e-01  2.99172510e-01]
 [ 0.00000000e+00  0.00000000e+00  0.00000000e+00  1.00000000e+00]]
```

7 une fois que vous avez complété toutes les fonctions get_gxx(), on peut calculer la cinématique directe

```
def forward_kinematics(position_dict):
    # Calcul des matrices individuelles
    gw1 = get_gw1(position_dict['shoulder_pan'])
    g12 = get_g12(position_dict['shoulder_lift'])
    g23 = get_g23(position_dict['elbow_flex'])
    g34 = get_g34(position_dict['wrist_flex'])
    g45 = get_g45(position_dict['wrist_roll'])
    g5t = get_g5t()
```

```
    # Matrice globale du end-effector
    gwt = gw1 @ g12 @ g23 @ g34 @ g45 @ g5t

    # Extraction position et rotation
    position = gwt[0:3, 3]
    rotation = gwt[0:3, 0:3]

    return position, rotation
```

j'ai l'impression qu'en suivant ce code de l'énnoncé, il y a un décalage dans les theta

```
# Exemple de configuration (en degrés)
test_positions = {
'shoulder_pan': -45.0,      # θ1
'shoulder_lift': 45.0,    # θ2
'elbow_flex': -45.0,        # θ3
'wrist_flex': 10.0,        # θ4
'wrist_roll': 90.0,         # θ5
'gripper': 10
}

print("\n=== TEST DU ROBOT 5-DOF ===")
print("Configuration envoyée :")

for k, v in test_positions.items():
    print(f"  {k} : {v}°")

    # Appel de ta cinématique directe
    position, rotation = forward_kinematics(test_positions)

    print("\n--- Résultats ---")
    print("\nPosition de l'end-effector (x, y, z) :")
    print(position)

    print("\nOrientation (matrice de rotation 3x3) :")
    print(rotation)


=== TEST DU ROBOT 5-DOF ===
Configuration envoyée :
  shoulder_pan : -45.0°

--- Résultats ---

Position de l'end-effector (x, y, z) :
[ 0.18017828  0.0212132  -0.20825313]

Orientation (matrice de rotation 3x3) :
[[ 8.19152044e-01  1.88807946e-16 -5.73576436e-01]
 [-1.28787392e-16  1.00000000e+00  7.66593583e-17]
```

```
 [ 5.73576436e-01  1.10737435e-17  8.19152044e-01]]
  shoulder_lift : 45.0°

--- Résultats ---

Position de l'end-effector (x, y, z) :
[ 0.18017828  0.0212132  -0.20825313]

Orientation (matrice de rotation 3x3) :
[[ 8.19152044e-01  1.88807946e-16 -5.73576436e-01]
 [-1.28787392e-16  1.00000000e+00  7.66593583e-17]
 [ 5.73576436e-01  1.10737435e-17  8.19152044e-01]]
  elbow_flex : -45.0°

--- Résultats ---

Position de l'end-effector (x, y, z) :
[ 0.18017828  0.0212132  -0.20825313]

Orientation (matrice de rotation 3x3) :
[[ 8.19152044e-01  1.88807946e-16 -5.73576436e-01]
 [-1.28787392e-16  1.00000000e+00  7.66593583e-17]
 [ 5.73576436e-01  1.10737435e-17  8.19152044e-01]]
  wrist_flex : 10.0°

--- Résultats ---

Position de l'end-effector (x, y, z) :
[ 0.18017828  0.0212132  -0.20825313]

Orientation (matrice de rotation 3x3) :
[[ 8.19152044e-01  1.88807946e-16 -5.73576436e-01]
 [-1.28787392e-16  1.00000000e+00  7.66593583e-17]
 [ 5.73576436e-01  1.10737435e-17  8.19152044e-01]]
  wrist_roll : 90.0°

--- Résultats ---

Position de l'end-effector (x, y, z) :
[ 0.18017828  0.0212132  -0.20825313]

Orientation (matrice de rotation 3x3) :
[[ 8.19152044e-01  1.88807946e-16 -5.73576436e-01]
 [-1.28787392e-16  1.00000000e+00  7.66593583e-17]
 [ 5.73576436e-01  1.10737435e-17  8.19152044e-01]]
  gripper : 10°

--- Résultats ---

Position de l'end-effector (x, y, z) :
[ 0.18017828  0.0212132  -0.20825313]
```

```
Orientation (matrice de rotation 3x3) :
[[ 8.19152044e-01  1.88807946e-16 -5.73576436e-01]
 [-1.28787392e-16  1.00000000e+00  7.66593583e-17]
 [ 5.73576436e-01  1.10737435e-17  8.19152044e-01]]
```

8 comparez votre modèle cinématique direct et avec celui de la bibliothèque lerobot-kinematics

```
from lerobot_kinematics.lerobot.lerobot_Kinematics import get_robot
# Get the SO100 robot model
robot = get_robot("so101")

from lerobot_kinematics.lerobot.lerobot_Kinematics import lerobot_FK
import numpy as np
# Define joint angles (in radians)
qpos = np.array([0.0, 0.0, 0.0, 0.0, 0.0])  # All joints at zero
position
# Compute forward kinematics
end_effector_pose = lerobot_FK(qpos, robot)
# Extract position and orientation
position = end_effector_pose[:3]  # [X, Y, Z]
orientation = end_effector_pose[3:]  # [gamma, beta, alpha]
print(f"Position: {position}")
print(f"Orientation: {orientation}")
```

```
---------------------------------------------------------------------------
-----
Exception                                 Traceback (most recent call
last)
Cell In[47], line 10
      8 qpos = np.array([0.0, 0.0, 0.0, 0.0, 0.0])  # All joints at
zero position
      9 # Compute forward kinematics
---> 10 end_effector_pose = lerobot_FK(qpos, robot)
     11 # Extract position and orientation
     12 position = end_effector_pose[:3]  # [X, Y, Z]

File
~/lerobot-kinematics/lerobot_kinematics/lerobot/lerobot_Kinematics.py:
112, in lerobot_FK(qpos_data, robot)
    110 def lerobot_FK(qpos_data, robot):
    111     if len(qpos_data) != len(robot.qlim[0]):
--> 112         raise Exception("The dimensions of qpose_data are not
the same as the robot joint dimensions")
    113     # Get the end effector's homogeneous transformation matrix
(T is an SE3 object)
    114     T = robot.fkine(qpos_data)
```

```
Exception: The dimensions of qpose_data are not the same as the robot
joint dimensions
```