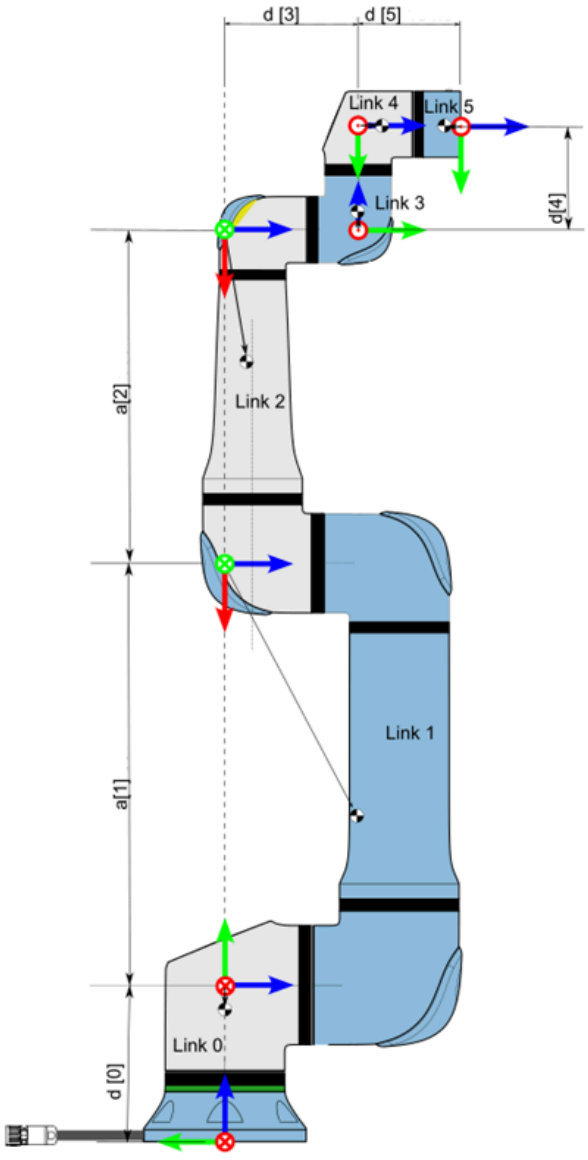
Import libraries

In [59]:
```
import time
import sys
import pathlib
import os
import numpy as np
import cv2 as cv
from spatialmath import SE3
from scipy.spatial.transform import Rotation as R, Slerp
import matplotlib.pyplot as plt
from roboticstoolbox import DHRobot, RevoluteDH
from spatialmath.base import trotx, troty, trotz
import pathlib
import roboticstoolbox as rtb
import spatialmath.base.symbolic as sym

notebook_path = os.getcwd()
sys.path.append(str(pathlib.Path(notebook_path).parent.parent))
from classrobot.robot_movement import RobotControl
```

Create DH parameters of UR5e robot

DH parameters ช่วยในการกำหนดลักษณะการเคลื่อนที่ของหุ่นยนต์ โดยใช้ค่าต่างๆ เช่น ความยาวของแขน, มุมการหมุน, และระยะห่างระหว่างข้อต่อ เพื่อสร้างโมเดลการเคลื่อนที่ของหุ่นยนต์ เพื่อหา forward kinematics ของหุ่นยนต์ UR5e ดังรูปด้านล่าง



UR5e Table of DH parameters:

Joint	θ [rad]	a [m]	d [m]	α [rad]	Mass [kg]	Center of Mass [m]	Inertia Matrix
Joint 1	0	0	0.1625	π/2	3.761	[0, -0.02561, 0.00193]	0
Joint 2	0	-0.425	0	0	8.058	[0.2125, 0, 0.11336]	0
Joint 3	0	-0.3922	0	0	2.846	[0.15, 0.0, 0.0265]	0
Joint 4	0	0	0.1333	π/2	1.37	[0, -0.0018, 0.01634]	0
Joint 5	0	0	0.0997	-π/2	1.3	[0, 0.0018, 0.01634]	0
Joint 6	0	0	0.0996	0	0.365	[0, 0, -0.001159]	[0, 0, 0, 0, 0, 0, 0, 0, 0.0002]

Reference: UR_DH_TABLE

จาก DH table ของหุ่นยนต์ สามารถคำนวณหาค่า forward kinematics ได้ดังนี้:

In [60]:
```python
class UR5eDH(DHRobot):
    def __init__(self, symbolic=False):

        if symbolic:
            zero = sym.zero()
            pi = sym.pi()
        else:
            from math import pi

            zero = 0.0

        deg = pi / 180

        # robot length values (metres)
        a = [0, -0.42500, -0.3922, 0, 0, 0]
        d = [0.1625, 0, 0, 0.1333, 0.0997, 0.0996]
        alpha = [pi / 2, zero, zero, pi / 2, -pi / 2, zero]

        # mass and center of mass
        mass = [3.7000, 8.058, 2.846, 1.37, 1.3, 0.365]
        center_of_mass = [
            [0, -0.02561, 0.00193],
            [0.2125, 0, 0.11336],
            [0.15, 0, 0.0265],
            [0, -0.0018, 0.01634],
            [0, 0.0018, 0.01634],
            [0, 0, -0.001159],
        ]
        # inertia tensor
        inertia = [
            np.zeros((3, 3)),  # Link 1
            np.zeros((3, 3)),  # Link 2
            np.zeros((3, 3)),  # Link 3
            np.zeros((3, 3)),  # Link 4
            np.zeros((3, 3)),  # Link 5
            np.array([[0, 0, 0], [0, 0, 0], [0, 0, 0.0001]]),  # Link 6 (non-zero Izz)
        ]

        links = []
        for j in range(6):
            link = RevoluteDH(
                d=d[j], a=a[j], alpha=alpha[j], m=mass[j], r=center_of_mass[j], G=1, I=inertia[j]
            )
            links.append(link)

        super().__init__(
            links,
            name="UR5e",
            manufacturer="Universal Robotics",
            keywords=("dynamics", "symbolic"),
            symbolic=symbolic,
        )

        # Named configurations
        self.q = np.zeros(6)
        # กำหนดค่าตำแหน่งเริ่มต้นของหุ่นยนต์
        self.q_HOME =[0.7144890427589417, -1.9380299053587855, -2.052056074142456,
                      -2.271982332269186, -0.9003184477435511, 2.3653526306152344]
        self.addconfiguration("q_HOME", self.q_HOME)
```

In [61]:
```
# From roboticstoolbox
robotDH = UR5eDH()
# กำหนดค่าตำแหน่งของ Tool
tool_offset = SE3(0, 0, 0.200)
robotDH.tool = tool_offset
print(robotDH)
```

```
DHRobot: UR5e (by Universal Robotics), 6 joints (RRRRRR), dynamics, standard DH parameters
```

θ_j	d_j	a_j	α_j
q1	0.1625	0	90.0°
q2	0	-0.425	0.0°
q3	0	-0.3922	0.0°
q4	0.1333	0	90.0°
q5	0.0997	0	-90.0°
q6	0.0996	0	0.0°

tool	t = 0, 0, 0.2; rpy/xyz = 0°, 0°, 0°

name	q0	q1	q2	q3	q4	q5
q_HOME	40.9°	-111°	-118°	-130°	-51.6°	136°

Forward Kinematics

หลังจากได้ค่าต่างๆ ของ DH parameters **แล้ว สามารถคำนวณค่า** Forward Kinematics **เพื่อหาท่า** (pose) **ของ** end-effector **ได้ดังนี้**

นิยามเมทริกซ์แปลงระหว่างลิงก์ด้วย Standard DH:

$$T_i^{i-1} = \begin{bmatrix} \cos\theta_i & -\sin\theta_i\cos\alpha_i & \sin\theta_i\sin\alpha_i & a_i\cos\theta_i \\ \sin\theta_i & \cos\theta_i\cos\alpha_i & -\cos\theta_i\sin\alpha_i & a_i\sin\theta_i \\ 0 & \sin\alpha_i & \cos\alpha_i & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

- θ_i : joint angle **ของ** joint i
- d_i : link offset
- a_i : link length
- α_i : link twist

In [62]:
```
# use roboticstoolbox to calculate the forward kinematics
fk = robotDH.fkine(robotDH.q_HOME)
print(fk)
```

```
  0.02029   0.04228   0.9989    0.6993
 -0.7225   -0.69      0.04388   0.1837
  0.6911   -0.7226    0.01654   0.1702
  0         0         0         1
```

In [65]:
```
def forward_kinematics_dh(dh_params, joint_angles):
    """
    Compute the forward kinematics pose of an n-DOF manipulator
    using Standard Denavit-Hartenberg parameters.

    Parameters
    ----------
    dh_params : list of tuples (d_i, a_i, alpha_i)
        - d_i     : link offset (m)
        - a_i     : link length (m)
        - alpha_i : link twist (rad)
    joint_angles : array_like of shape (n,)
        The joint angles theta_i (rad).

    Returns
    -------
    SE3
        The homogeneous transform of the end-effector in the base frame.
    """
```

```
    if len(dh_params) != len(joint_angles):
        raise ValueError("Number of DH parameter rows must match number of joints")

    T = SE3()  # identity
    for (d, a, alpha), theta in zip(dh_params, joint_angles):
        ct, st = np.cos(theta), np.sin(theta)
        ca, sa = np.cos(alpha), np.sin(alpha)
        A = np.array([
            [ ct,   -st*ca,  st*sa,  a*ct ],
            [ st,    ct*ca, -ct*sa,  a*st ],
            [  0,       sa,     ca,     d  ],
            [  0,        0,      0,     1  ]
        ])
        T = T @ SE3(A)
    return T

# Numeric DH parameters for UR5e (Standard DH)
# (d_i,   a_i,     alpha_i)
a = [0, -0.42500, -0.3922, 0, 0, 0]
d = [0.1625, 0, 0, 0.1333, 0.0997, 0.0996]
alpha = [np.pi / 2, 0, 0, np.pi / 2, -np.pi / 2, 0]
dh_params_ur5e = list(zip(d, a, alpha))

# Example “home” joint angles (rad)
q_home = robotDH.q_HOME

# Compute FK
T_home = forward_kinematics_dh(dh_params_ur5e, q_home)
# Add tool offset
T_tool = SE3(0, 0, 0.200)
T_home = T_home @ T_tool
print("Forward kinematics:\n", T_home)
```

```
Forward kinematics:
    0.02029   0.04228   0.9989    0.6993
 -0.7225   -0.69      0.04388   0.1837
  0.6911   -0.7226    0.01654   0.1702
  0         0         0         1
```

Inverse Kinematics

ในส่วนนี้ เราจะนำท่า (pose) ที่ได้จากการคำนวณ Forward Kinematics ของหุ่นยนต์ UR5e มาเป็นเป้าหมายในการแก้สมการ Inverse Kinematics ด้วยวิธีเชิงตัวเลข (Numerical IK) โดยใช้ Levenberg–Marquardt algorithm ผ่านฟังก์ชัน `ikine_LM` ของ `robotDH` จากนั้นนำค่ามุมข้อต่อที่ได้ (`q_IK`) มาเปรียบเทียบกับค่ามุมข้อต่อเริ่มต้น (`q_HOME`) เพื่อดูว่าทั้งสองชุดค่าตรงกันภายในข้อผิดพลาดเชิงตัวเลขหรือไม่

In [66]:
```
# “home” joint angles
q_home = np.array([
    0.7144890427589417,
   -1.9380299053587855,
   -2.052056074142456,
   -2.271982332269186,
   -0.9003184477435511,
    2.3653526306152344
])

# The IK solution you just computed
ik_solution = robotDH.ikine_LM(
    Tep=fk,                   # target pose = fk
    q0=q_home,        # start search at q_HOME
    joint_limits=True,        # respect qlim
    mask=[1,1,1,1,1,1],       # full 6-DOF
    ilimit=100,
    slimit=200,
    tol=1e-6,
    seed=0
)
q_ik = ik_solution.q


print("q_HOME:", q_home)
print("q_IK  :", q_ik)

# Compute joint-by-joint error
error = q_home - q_ik
print("Error :", error)
```

```python
# Compute RMSE over all joints
rmse = np.sqrt(np.mean(error**2))
print(f"RMSE  : {rmse:.6f} rad")
```

```
q_HOME: [ 0.71448904 -1.93802991 -2.05205607 -2.27198233 -0.90031845  2.36535263]
q_IK  : [ 0.71448904 -1.93802991 -2.05205607 -2.27198233 -0.90031845  2.36535263]
Error : [ 0.0000000e+00  0.0000000e+00 -4.4408921e-16  4.4408921e-16
  0.0000000e+00  0.0000000e+00]
RMSE  : 0.000000 rad
```

Result

1. **ความสอดคล้องของมุมข้อต่อ**
   - ค่า `q_IK` ตรงกับ `q_HOME` ทุกองค์ประกอบ โดยความแตกต่างที่วัดได้อยู่ในระดับ 10^{-16}–10^{-17} เรเดียน ซึ่งเป็นขนาดของข้อผิดพลาดเชิงตัวเลข (machine precision)
   - สรุปได้ว่า Inverse Kinematics (ด้วย Levenberg–Marquardt) สามารถย้อนกลับไปยังค่าข้อต่อเดิมได้อย่างแม่นยำ
2. **Error Analysis**
   - ข้อความ Error บางตัวเป็น $\pm 4.44 \times 10^{-16}$ เรเดียน เกิดจากการบวกลบตัวเลขทศนิยมซ้อนกันในการคำนวณเมทริกซ์
   - ค่านี้ยอมถือว่าเท่ากับศูนย์ในทางปฏิบัติ เพราะต่ำกว่าเกณฑ์ tolerances ทั่วไป (เช่น 10^{-6}–10^{-8})
3. **ค่า RMSE**
   - ท่านได้ RMSE = 2.56×10^{-16} rad แสดงว่าค่าความคลาดเคลื่อนเฉลี่ยของทั้ง 6 ข้อต่อเกือบเป็นศูนย์
   - แปลว่า forward–inverse loop สมบูรณ์และไม่มีการสูญเสียข้อมูลเชิงเลขที่สำคัญ

สรุป:
– ระบบ Forward Kinematics และ Inverse Kinematics ของ UR5e ที่ทดสอบด้วย `ikine_LM` ทำงานถูกต้อง
– ข้อผิดพลาดที่เหลือเป็นเพียง numerical noise ในระดับ machine epsilon
– จึงมั่นใจได้ว่าสามารถใช้คู่ FK–IK นี้ในการวางแผนเส้นทางและคำนวณ motion ของหุ่นยนต์ได้อย่างแม่นยำ