In [1]:
```
import numpy as np
from scipy.linalg import expm, logm
from ikin_api import IKinSpace, IKinBody
```

Forward Kinematics Practice

In [2]:
```
# Function to form the skew-symmetric matrix of a vector
def skew_symmetric(vec):
    return np.array([[0, -vec[2, 0], vec[1, 0]],
                     [vec[2, 0], 0, -vec[0, 0]],
                     [-vec[1, 0], vec[0, 0], 0]])
```

RRP Robot

In [3]:
```
# Let's build up the M matrix
R = np.array([[0, 1, 0 ],
              [1, 0, 0 ],
              [0, 0, -1]])

print("Rotation matrix R:\n", R)
print("Shape of R:", R.shape)
print("Determinant of R:", np.linalg.det(R))

t = np.array([[0],
              [0 ],
              [-6]])

print("Translation vector t:\n", t)
print("Shape of t:", t.shape)

M = np.block([[R, t],
              [0, 0, 0, 1]])

print("Homogeneous transformation matrix M:\n", M)
print("Shape of M:", M.shape)
```

```
Rotation matrix R:
 [[ 0  1  0]
 [ 1  0  0]
 [ 0  0 -1]]
Shape of R: (3, 3)
Determinant of R: 1.0
Translation vector t:
 [[ 0]
 [ 0]
 [-6]]
Shape of t: (3, 1)
Homogeneous transformation matrix M:
 [[ 0  1  0  0]
 [ 1  0  0  0]
 [ 0  0 -1 -6]
 [ 0  0  0  1]]
Shape of M: (4, 4)
```

In [4]:

```python
# Joint 0 Screw axis (fixed frame)
w_s_0 = np.array([[0 ],
                  [0 ],
                  [-1]])
print("Angular velocity (w_s_0):\n", w_s_0)
q_s_0 = np.array([[-2],
                  [0 ],
                  [-2]])
print("Point on the screw axis (q_s_0):\n", q_s_0)
v_s_0 = np.cross(-w_s_0.flatten(), q_s_0.flatten()).reshape(3, 1)
print("Linear velocity (v_s_0):\n", v_s_0)
s_0 = np.block([[w_s_0],
                [v_s_0]])
print("Screw axis for joint 0 (s_0):\n", s_0)
print("Shape of s_0:", s_0.shape)

# Joint 0 Screw axis (body frame)
w_b_0 = np.array([[0],
                  [0],
                  [1]])
print("Angular velocity (w_b_0):\n", w_b_0)
q_b_0 = np.array([[0 ],
                  [-2],
```

```
                  [-4]])
print("Point on the screw axis (q_b_0):\n", q_b_0)
v_b_0 = np.cross(-w_b_0.flatten(), q_b_0.flatten()).reshape(3, 1)
print("Linear velocity (v_b_0):\n", v_b_0)
b_0 = np.block([[w_b_0],
                [v_b_0]])
print("Screw axis for joint 0 (b_0):\n", b_0)
print("Shape of b_0:", b_0.shape)

# Joint 1 Screw axis (fixed frame)
w_s_1 = np.array([[0 ],
                  [0 ],
                  [-1]])
print("Angular velocity (w_s_1):\n", w_s_1)
q_s_1 = np.array([[-2],
                  [0 ],
                  [-4]])
print("Point on the screw axis (q_s_1):\n", q_s_1)
v_s_1 = np.cross(-w_s_1.flatten(), q_s_1.flatten()).reshape(3, 1)
print("Linear velocity (v_s_1):\n", v_s_1)
s_1 = np.block([[w_s_1],
                [v_s_1]])
print("Screw axis for joint 1 (s_1):\n", s_1)
print("Shape of s_1:", s_1.shape)

# Joint 1 Screw axis (body frame)
w_b_1 = np.array([[0],
                  [0],
                  [1]])
print("Angular velocity (w_b_1):\n", w_b_1)
q_b_1 = np.array([[0 ],
                  [-2],
                  [-2]])
print("Point on the screw axis (q_b_1):\n", q_b_1)
v_b_1 = np.cross(-w_b_1.flatten(), q_b_1.flatten()).reshape(3, 1)
print("Linear velocity (v_b_1):\n", v_b_1)
b_1 = np.block([[w_b_1],
                [v_b_1]])
print("Screw axis for joint 1 (b_1):\n", b_1)
print("Shape of b_1:", b_1.shape)
```

```
# Joint 2 Screw axis (fixed frame)
w_s_2 = np.array([[0],
                  [0],
                  [0]])
print("Angular velocity (w_s_2):\n", w_s_2)
v_s_2 = np.array([[0 ],
                  [0 ],
                  [-1]])
print("Linear velocity (v_s_2):\n", v_s_2)
s_2 = np.block([[w_s_2],
                [v_s_2]])
print("Screw axis for joint 2 (s_2):\n", s_2)
print("Shape of s_2:", s_2.shape)

# Joint 2 Screw axis (body frame)
w_b_2 = np.array([[0],
                  [0],
                  [0]])
print("Angular velocity (w_b_2):\n", w_b_2)
v_b_2 = np.array([[0],
                  [0],
                  [1]])
print("Linear velocity (v_b_2):\n", v_b_2)
b_2 = np.block([[w_b_2],
                [v_b_2]])
print("Screw axis for joint 2 (b_2):\n", b_2)
print("Shape of b_2:", b_2.shape)
```

```
Angular velocity (w_s_0):
 [[ 0]
 [ 0]
 [-1]]
Point on the screw axis (q_s_0):
 [[-2]
 [ 0]
 [-2]]
Linear velocity (v_s_0):
 [[ 0]
 [-2]
 [ 0]]
Screw axis for joint 0 (s_0):
 [[ 0]
 [ 0]
 [-1]
 [ 0]
 [-2]
 [ 0]]
Shape of s_0: (6, 1)
Angular velocity (w_b_0):
 [[0]
 [0]
 [1]]
Point on the screw axis (q_b_0):
 [[ 0]
 [-2]
 [-4]]
Linear velocity (v_b_0):
 [[-2]
 [ 0]
 [ 0]]
Screw axis for joint 0 (b_0):
 [[ 0]
 [ 0]
 [ 1]
 [-2]
 [ 0]
 [ 0]]
Shape of b_0: (6, 1)
Angular velocity (w_s_1):
```

```
 [[ 0]
 [ 0]
 [-1]]
Point on the screw axis (q_s_1):
 [[-2]
 [ 0]
 [-4]]
Linear velocity (v_s_1):
 [[ 0]
 [-2]
 [ 0]]
Screw axis for joint 1 (s_1):
 [[ 0]
 [ 0]
 [-1]
 [ 0]
 [-2]
 [ 0]]
Shape of s_1: (6, 1)
Angular velocity (w_b_1):
 [[0]
 [0]
 [1]]
Point on the screw axis (q_b_1):
 [[ 0]
 [-2]
 [-2]]
Linear velocity (v_b_1):
 [[-2]
 [ 0]
 [ 0]]
Screw axis for joint 1 (b_1):
 [[ 0]
 [ 0]
 [ 1]
 [-2]
 [ 0]
 [ 0]]
Shape of b_1: (6, 1)
Angular velocity (w_s_2):
 [[0]
```

```
 [0]
 [0]]
Linear velocity (v_s_2):
 [[ 0]
 [ 0]
 [-1]]
Screw axis for joint 2 (s_2):
 [[ 0]
 [ 0]
 [ 0]
 [ 0]
 [ 0]
 [-1]]
Shape of s_2: (6, 1)
Angular velocity (w_b_2):
 [[0]
 [0]
 [0]]
Linear velocity (v_b_2):
 [[0]
 [0]
 [1]]
Screw axis for joint 2 (b_2):
 [[0]
 [0]
 [0]
 [0]
 [0]
 [1]]
Shape of b_2: (6, 1)
```

In [5]:

```python
# Joint 0 Screw matrix (fixed frame)
bracket_w_s_0 = skew_symmetric(w_s_0)
print("Skew-symmetric matrix of w_s_0 (bracket_w_s_0):\n", bracket_w_s_0)

mat_s_0 = np.block([[bracket_w_s_0, v_s_0],
                    [0, 0, 0, 0]])
print("Matrix form of screw axis s_0 (mat_s_0):\n", mat_s_0)
print("Shape of mat_s_0:", mat_s_0.shape)

# Joint 0 Screw matrix (body frame)
```

```
bracket_w_b_0 = skew_symmetric(w_b_0)
print("Skew-symmetric matrix of w_b_0 (bracket_w_b_0):\n", bracket_w_b_0)

mat_b_0 = np.block([[bracket_w_b_0, v_b_0],
                    [0, 0, 0, 0]])
print("Matrix form of screw axis b_0 (mat_b_0):\n", mat_b_0)
print("Shape of mat_b_0:", mat_b_0.shape)

# Joint 1 Screw matrix (fixed frame)
bracket_w_s_1 = skew_symmetric(w_s_1)
print("Skew-symmetric matrix of w_s_1 (bracket_w_s_1):\n", bracket_w_s_1)

mat_s_1 = np.block([[bracket_w_s_1, v_s_1],
                    [0, 0, 0, 0]])
print("Matrix form of screw axis s_1 (mat_s_1):\n", mat_s_1)
print("Shape of mat_s_1:", mat_s_1.shape)

# Joint 1 Screw matrix (body frame)
bracket_w_b_1 = skew_symmetric(w_b_1)
print("Skew-symmetric matrix of w_b_1 (bracket_w_b_1):\n", bracket_w_b_1)

mat_b_1 = np.block([[bracket_w_b_1, v_b_1],
                    [0, 0, 0, 0]])
print("Matrix form of screw axis b_1 (mat_b_1):\n", mat_b_1)
print("Shape of mat_b_1:", mat_b_1.shape)

# Joint 2 Screw matrix (fixed frame)
bracket_w_s_2 = skew_symmetric(w_s_2)
print("Skew-symmetric matrix of w_s_2 (bracket_w_s_2):\n", bracket_w_s_2)

mat_s_2 = np.block([[bracket_w_s_2, v_s_2],
                    [0, 0, 0, 0]])
print("Matrix form of screw axis s_2 (mat_s_2):\n", mat_s_2)
print("Shape of mat_s_2:", mat_s_2.shape)

# Joint 2 Screw matrix (body frame)
bracket_w_b_2 = skew_symmetric(w_b_2)
print("Skew-symmetric matrix of w_b_2 (bracket_w_b_2):\n", bracket_w_b_2)

mat_b_2 = np.block([[bracket_w_b_2, v_b_2],
                    [0, 0, 0, 0]])
```

```
print("Matrix form of screw axis b_2 (mat_b_2):\n", mat_b_2)
print("Shape of mat_b_2:", mat_b_2.shape)
```

```
Skew-symmetric matrix of w_s_0 (bracket_w_s_0):
 [[ 0  1  0]
 [-1  0  0]
 [ 0  0  0]]
Matrix form of screw axis s_0 (mat_s_0):
 [[ 0  1  0  0]
 [-1  0  0 -2]
 [ 0  0  0  0]
 [ 0  0  0  0]]
Shape of mat_s_0: (4, 4)
Skew-symmetric matrix of w_b_0 (bracket_w_b_0):
 [[ 0 -1  0]
 [ 1  0  0]
 [ 0  0  0]]
Matrix form of screw axis b_0 (mat_b_0):
 [[ 0 -1  0 -2]
 [ 1  0  0  0]
 [ 0  0  0  0]
 [ 0  0  0  0]]
Shape of mat_b_0: (4, 4)
Skew-symmetric matrix of w_s_1 (bracket_w_s_1):
 [[ 0  1  0]
 [-1  0  0]
 [ 0  0  0]]
Matrix form of screw axis s_1 (mat_s_1):
 [[ 0  1  0  0]
 [-1  0  0 -2]
 [ 0  0  0  0]
 [ 0  0  0  0]]
Shape of mat_s_1: (4, 4)
Skew-symmetric matrix of w_b_1 (bracket_w_b_1):
 [[ 0 -1  0]
 [ 1  0  0]
 [ 0  0  0]]
Matrix form of screw axis b_1 (mat_b_1):
 [[ 0 -1  0 -2]
 [ 1  0  0  0]
 [ 0  0  0  0]
 [ 0  0  0  0]]
Shape of mat_b_1: (4, 4)
Skew-symmetric matrix of w_s_2 (bracket_w_s_2):
```

```
 [[0 0 0]
 [0 0 0]
 [0 0 0]]
Matrix form of screw axis s_2 (mat_s_2):
 [[ 0  0  0  0]
 [ 0  0  0  0]
 [ 0  0  0 -1]
 [ 0  0  0  0]]
Shape of mat_s_2: (4, 4)
Skew-symmetric matrix of w_b_2 (bracket_w_b_2):
 [[0 0 0]
 [0 0 0]
 [0 0 0]]
Matrix form of screw axis b_2 (mat_b_2):
 [[0 0 0 0]
 [0 0 0 0]
 [0 0 0 1]
 [0 0 0 0]]
Shape of mat_b_2: (4, 4)
```

In [6]:

```python
# Example theta for joints 0, 1, and 2
theta_0 = 0.25
theta_1 = 0.50
theta_2 = 0.75

# Product of exponentials formula (fixed frame)
T_1_in_0_theta_s = expm(mat_s_0 * theta_0) @ expm(mat_s_1 * theta_1) @ expm(mat_s_2 * theta_2) @ M

print("Transformation matrix T_1_in_0_theta_s (fixed frame):\n", T_1_in_0_theta_s)
print("Shape of T_1_in_0_theta_s:", T_1_in_0_theta_s.shape)

# Product of exponentials formula (body frame)
T_1_in_0_theta_b = M @ expm(mat_b_0 * theta_0) @ expm(mat_b_1 * theta_1) @ expm(mat_b_2 * theta_2)

print("Transformation matrix T_1_in_0_theta_b (body frame):\n", T_1_in_0_theta_b)
print("Shape of T_1_in_0_theta_b:", T_1_in_0_theta_b.shape)
```

```
Transformation matrix T_1_in_0_theta_s (fixed frame):
 [[ 0.68163876  0.73168887  0.         -0.53662226]
 [ 0.73168887 -0.68163876  0.         -1.36327752]
 [ 0.          0.         -1.         -6.75      ]
 [ 0.          0.          0.          1.        ]]
Shape of T_1_in_0_theta_s: (4, 4)
Transformation matrix T_1_in_0_theta_b (body frame):
 [[ 0.68163876  0.73168887  0.         -0.53662226]
 [ 0.73168887 -0.68163876  0.         -1.36327752]
 [ 0.          0.         -1.         -6.75      ]
 [ 0.          0.          0.          1.        ]]
Shape of T_1_in_0_theta_b: (4, 4)
```

Velocity Kinematics Practice

PRPR Robot

In [7]:
```python
# Joint 0 Screw axis (fixed frame)
w_s_0 = np.array([[0],
                  [0],
                  [0]])
print("Angular velocity (w_s_0):\n", w_s_0)
v_s_0 = np.array([[-1],
                  [0 ],
                  [0 ]])
print("Linear velocity (v_s_0):\n", v_s_0)
s_0 = np.block([[w_s_0],
                [v_s_0]])
print("Screw axis for joint 0 (s_0):\n", s_0)
print("Shape of s_0:", s_0.shape)

# Joint 1 Screw axis (fixed frame)
w_s_1 = np.array([[-1],
                  [0 ],
                  [0 ]])
print("Angular velocity (w_s_1):\n", w_s_1)
q_s_1 = np.array([[-2],
                  [2 ],
```

```
                  [0 ]])
print("Point on the screw axis (q_s_1):\n", q_s_1)
v_s_1 = np.cross(-w_s_1.flatten(), q_s_1.flatten()).reshape(3, 1)
print("Linear velocity (v_s_1):\n", v_s_1)
s_1 = np.block([[w_s_1],
                [v_s_1]])
print("Screw axis for joint 1 (s_1):\n", s_1)
print("Shape of s_1:", s_1.shape)

# Joint 2 Screw axis (fixed frame)
w_s_2 = np.array([[0],
                  [0],
                  [0]])
print("Angular velocity (w_s_2):\n", w_s_2)
v_s_2 = np.array([[0],
                  [1],
                  [0]])
print("Linear velocity (v_s_2):\n", v_s_2)
s_2 = np.block([[w_s_2],
                [v_s_2]])
print("Screw axis for joint 2 (s_2):\n", s_2)
print("Shape of s_2:", s_2.shape)

# Joint 3 Screw axis (fixed frame)
w_s_3 = np.array([[-1],
                  [0 ],
                  [0 ]])
print("Angular velocity (w_s_3):\n", w_s_3)
q_s_3 = np.array([[-2],
                  [6 ],
                  [0 ]])
print("Point on the screw axis (q_s_3):\n", q_s_3)
v_s_3 = np.cross(-w_s_3.flatten(), q_s_3.flatten()).reshape(3, 1)
print("Linear velocity (v_s_3):\n", v_s_3)
s_3 = np.block([[w_s_3],
                [v_s_3]])
print("Screw axis for joint 3 (s_3):\n", s_3)
print("Shape of s_3:", s_3.shape)
```

```
Angular velocity (w_s_0):
 [[0]
 [0]
 [0]]
Linear velocity (v_s_0):
 [[-1]
 [ 0]
 [ 0]]
Screw axis for joint 0 (s_0):
 [[ 0]
 [ 0]
 [ 0]
 [-1]
 [ 0]
 [ 0]]
Shape of s_0: (6, 1)
Angular velocity (w_s_1):
 [[-1]
 [ 0]
 [ 0]]
Point on the screw axis (q_s_1):
 [[-2]
 [ 2]
 [ 0]]
Linear velocity (v_s_1):
 [[0]
 [0]
 [2]]
Screw axis for joint 1 (s_1):
 [[-1]
 [ 0]
 [ 0]
 [ 0]
 [ 0]
 [ 2]]
Shape of s_1: (6, 1)
Angular velocity (w_s_2):
 [[0]
 [0]
 [0]]
Linear velocity (v_s_2):
```

```
 [[0]
 [1]
 [0]]
Screw axis for joint 2 (s_2):
 [[0]
 [0]
 [0]
 [0]
 [1]
 [0]]
Shape of s_2: (6, 1)
Angular velocity (w_s_3):
 [[-1]
 [ 0]
 [ 0]]
Point on the screw axis (q_s_3):
 [[-2]
 [ 6]
 [ 0]]
Linear velocity (v_s_3):
 [[0]
 [0]
 [6]]
Screw axis for joint 3 (s_3):
 [[-1]
 [ 0]
 [ 0]
 [ 0]
 [ 0]
 [ 6]]
Shape of s_3: (6, 1)
```

In [8]:

```python
# Joint 0 Screw matrix (fixed frame)
bracket_w_s_0 = skew_symmetric(w_s_0)
print("Skew-symmetric matrix of w_s_0 (bracket_w_s_0):\n", bracket_w_s_0)

mat_s_0 = np.block([[bracket_w_s_0, v_s_0],
                    [0, 0, 0, 0]])
print("Matrix form of screw axis s_0 (mat_s_0):\n", mat_s_0)
print("Shape of mat_s_0:", mat_s_0.shape)
```

```python
# Joint 1 Screw matrix (fixed frame)
bracket_w_s_1 = skew_symmetric(w_s_1)
print("Skew-symmetric matrix of w_s_1 (bracket_w_s_1):\n", bracket_w_s_1)

mat_s_1 = np.block([[bracket_w_s_1, v_s_1],
                    [0, 0, 0, 0]])
print("Matrix form of screw axis s_1 (mat_s_1):\n", mat_s_1)
print("Shape of mat_s_1:", mat_s_1.shape)

# Joint 2 Screw matrix (fixed frame)
bracket_w_s_2 = skew_symmetric(w_s_2)
print("Skew-symmetric matrix of w_s_2 (bracket_w_s_2):\n", bracket_w_s_2)

mat_s_2 = np.block([[bracket_w_s_2, v_s_2],
                    [0, 0, 0, 0]])
print("Matrix form of screw axis s_2 (mat_s_2):\n", mat_s_2)
print("Shape of mat_s_2:", mat_s_2.shape)

# Joint 3 Screw matrix (fixed frame)
bracket_w_s_3 = skew_symmetric(w_s_3)
print("Skew-symmetric matrix of w_s_3 (bracket_w_s_3):\n", bracket_w_s_3)

mat_s_3 = np.block([[bracket_w_s_3, v_s_3],
                    [0, 0, 0, 0]])
print("Matrix form of screw axis s_3 (mat_s_3):\n", mat_s_3)
print("Shape of mat_s_3:", mat_s_3.shape)
```

```
Skew-symmetric matrix of w_s_0 (bracket_w_s_0):
 [[0 0 0]
 [0 0 0]
 [0 0 0]]
Matrix form of screw axis s_0 (mat_s_0):
 [[ 0  0  0 -1]
 [ 0  0  0  0]
 [ 0  0  0  0]
 [ 0  0  0  0]]
Shape of mat_s_0: (4, 4)
Skew-symmetric matrix of w_s_1 (bracket_w_s_1):
 [[ 0  0  0]
 [ 0  0  1]
 [ 0 -1  0]]
Matrix form of screw axis s_1 (mat_s_1):
 [[ 0  0  0  0]
 [ 0  0  1  0]
 [ 0 -1  0  2]
 [ 0  0  0  0]]
Shape of mat_s_1: (4, 4)
Skew-symmetric matrix of w_s_2 (bracket_w_s_2):
 [[0 0 0]
 [0 0 0]
 [0 0 0]]
Matrix form of screw axis s_2 (mat_s_2):
 [[0 0 0 0]
 [0 0 0 1]
 [0 0 0 0]
 [0 0 0 0]]
Shape of mat_s_2: (4, 4)
Skew-symmetric matrix of w_s_3 (bracket_w_s_3):
 [[ 0  0  0]
 [ 0  0  1]
 [ 0 -1  0]]
Matrix form of screw axis s_3 (mat_s_3):
 [[ 0  0  0  0]
 [ 0  0  1  0]
 [ 0 -1  0  6]
 [ 0  0  0  0]]
Shape of mat_s_3: (4, 4)
```

In [9]:
```python
# Function to form the adjoint of a homogeneous transformation matrix
def adjoint(T):
    R = T[0:3, 0:3]
    t = T[0:3, 3].reshape(3, 1)
    block_t = skew_symmetric(t)
    Ad_T = np.block([[R, np.zeros((3, 3))], [block_t @ R, R]])
    return Ad_T
```

In [10]:
```python
# Following the equation sheet:
# Compute the product of exponential and then take the adjoint of the resulting matrix

# Example theta for joints 0, 1, 2, and 3
theta_0 = 0.25
theta_1 = 0.50
theta_2 = 0.75
theta_3 = 1.00

# Jacobian column 0 (fixed frame)
J_s_0 = s_0
print("Jacobian column for joint 0 (fixed frame) J_s_0:\n", J_s_0)

# Jacobian column 1 (fixed frame)
Ad_exp_s_0_theta_0 = adjoint(expm(mat_s_0 * theta_0))
J_s_1 = Ad_exp_s_0_theta_0 @ s_1
print("Jacobian column for joint 1 (fixed frame) J_s_1:\n", J_s_1)

# Jacobian column 2 (fixed frame)
Ad_exp_s_1_theta_1 = adjoint(expm(mat_s_0 * theta_0) @ expm(mat_s_1 * theta_1))
J_s_2 = Ad_exp_s_1_theta_1 @ s_2
print("Jacobian column for joint 2 (fixed frame) J_s_2:\n", J_s_2)

# Jacobian column 3 (fixed frame)
Ad_exp_s_2_theta_2 = adjoint(expm(mat_s_0 * theta_0) @ expm(mat_s_1 * theta_1) @ expm(mat_s_2 * theta_2))
J_s_3 = Ad_exp_s_2_theta_2 @ s_3
print("Jacobian column for joint 3 (fixed frame) J_s_3:\n", J_s_3)

# Jacobian matrix (fixed frame)
J_s = np.block([[J_s_0, J_s_1, J_s_2, J_s_3]])
print("Jacobian matrix in the fixed frame J_s:\n", J_s)
```

```
Jacobian column for joint 0 (fixed frame) J_s_0:
 [[ 0]
 [ 0]
 [ 0]
 [-1]
 [ 0]
 [ 0]]
Jacobian column for joint 1 (fixed frame) J_s_1:
 [[-1.]
 [ 0.]
 [ 0.]
 [ 0.]
 [ 0.]
 [ 2.]]
Jacobian column for joint 2 (fixed frame) J_s_2:
 [[ 0.        ]
 [ 0.        ]
 [ 0.        ]
 [ 0.        ]
 [ 0.87758256]
 [-0.47942554]]
Jacobian column for joint 3 (fixed frame) J_s_3:
 [[-1.        ]
 [ 0.        ]
 [ 0.        ]
 [ 0.        ]
 [ 2.27727131]
 [ 6.16851717]]
Jacobian matrix in the fixed frame J_s:
 [[ 0.         -1.          0.         -1.        ]
 [ 0.          0.          0.          0.        ]
 [ 0.          0.          0.          0.        ]
 [-1.          0.          0.          0.        ]
 [ 0.          0.          0.87758256  2.27727131]
 [ 0.          2.         -0.47942554  6.16851717]]
```

In [11]:

```
# Or equivalently:
# Take the adjoint of each exponential and then compute the product of the adjoints

# Example theta for joints 0, 1, 2, and 3
theta_0 = 0.25
```

```
theta_1 = 0.50
theta_2 = 0.75
theta_3 = 1.00

# Jacobian column 0 (fixed frame)
J_s_0 = s_0
print("Jacobian column for joint 0 (fixed frame) J_s_0:\n", J_s_0)

# Jacobian column 1 (fixed frame)
Ad_exp_s_0_theta_0 = adjoint(expm(mat_s_0 * theta_0))
J_s_1 = Ad_exp_s_0_theta_0 @ s_1
print("Jacobian column for joint 1 (fixed frame) J_s_1:\n", J_s_1)

# Jacobian column 2 (fixed frame)
Ad_exp_s_1_theta_1 = adjoint(expm(mat_s_1 * theta_1))
J_s_2 = Ad_exp_s_0_theta_0 @ Ad_exp_s_1_theta_1 @ s_2
print("Jacobian column for joint 2 (fixed frame) J_s_2:\n", J_s_2)

# Jacobian column 3 (fixed frame)
Ad_exp_s_2_theta_2 = adjoint(expm(mat_s_2 * theta_2))
J_s_3 = Ad_exp_s_0_theta_0 @ Ad_exp_s_1_theta_1 @ Ad_exp_s_2_theta_2 @ s_3
print("Jacobian column for joint 3 (fixed frame) J_s_3:\n", J_s_3)

# Jacobian matrix (fixed frame)
J_s = np.block([[J_s_0, J_s_1, J_s_2, J_s_3]])
print("Jacobian matrix in the fixed frame J_s:\n", J_s)
```

```
Jacobian column for joint 0 (fixed frame) J_s_0:
 [[ 0]
 [ 0]
 [ 0]
 [-1]
 [ 0]
 [ 0]]
Jacobian column for joint 1 (fixed frame) J_s_1:
 [[-1.]
 [ 0.]
 [ 0.]
 [ 0.]
 [ 0.]
 [ 2.]]
Jacobian column for joint 2 (fixed frame) J_s_2:
 [[ 0.        ]
 [ 0.        ]
 [ 0.        ]
 [ 0.        ]
 [ 0.87758256]
 [-0.47942554]]
Jacobian column for joint 3 (fixed frame) J_s_3:
 [[-1.        ]
 [ 0.        ]
 [ 0.        ]
 [ 0.        ]
 [ 2.27727131]
 [ 6.16851717]]
Jacobian matrix in the fixed frame J_s:
 [[ 0.         -1.          0.         -1.        ]
 [ 0.          0.          0.          0.        ]
 [ 0.          0.          0.          0.        ]
 [-1.          0.          0.          0.        ]
 [ 0.          0.          0.87758256  2.27727131]
 [ 0.          2.         -0.47942554  6.16851717]]
```

In [12]:

```
# Computing the twist V_s given the space Jacobian and joint velocities

# Example joint velocities for joints 0, 1, 2, and 3
theta_dot_0 = 0.1
theta_dot_1 = 0.2
```

```
theta_dot_2 = 0.3
theta_dot_3 = 0.4

theta_dot = np.array([[theta_dot_0],
                      [theta_dot_1],
                      [theta_dot_2],
                      [theta_dot_3]])

V_s = J_s @ theta_dot

print("Twist V_s given the space Jacobian and joint velocities:\n", V_s)
```

```
Twist V_s given the space Jacobian and joint velocities:
 [[-0.6       ]
 [ 0.        ]
 [ 0.        ]
 [-0.1       ]
 [ 1.17418329]
 [ 2.72357921]]
```

In [13]:

```
# Solving for the body twist V_b given the fixed frame twist V_s and the transformation matrix T_bs
R = np.array([[0,  -1, 0 ],
              [-1, 0,  0 ],
              [0,  0,  -1]])
t = np.array([[-4],
              [6 ],
              [0 ]])
M = np.block([[R, t],
              [0, 0, 0, 1]])

T_sb = expm(mat_s_0 * theta_0) @ expm(mat_s_1 * theta_1) @ expm(mat_s_2 * theta_2) @ expm(mat_s_3 * theta_3) @ M
print("Transformation matrix T_sb:\n", T_sb)

T_bs = np.linalg.inv(T_sb)
V_b = adjoint(T_bs) @ V_s
print("Body twist V_b given the fixed frame twist V_s and the transformation matrix T_bs:\n", V_b)
```

```
Transformation matrix T_sb:
 [[ 0.         -1.          0.         -4.25      ]
 [-0.0707372   0.         -0.99749499  6.16851717]
 [ 0.99749499  0.         -0.0707372  -2.27727131]
 [ 0.          0.          0.          1.        ]]
Body twist V_b given the fixed frame twist V_s and the transformation matrix T_bs:
 [[ 0.        ]
 [ 0.6       ]
 [ 0.        ]
 [-0.96148813]
 [ 0.1       ]
 [ 0.2608459 ]]
```

In [14]:
```python
# Or equivalently:
# Solve for the body Jacobian and multiply it by the joint velocities to get the body twist V_b

J_b = adjoint(T_bs) @ J_s
print("Body Jacobian J_b:\n", J_b)

V_b = J_b @ theta_dot
print("Body twist V_b given the body Jacobian and joint velocities:\n", V_b)
```

```
Body Jacobian J_b:
 [[ 0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00]
 [ 0.00000000e+00  1.00000000e+00  0.00000000e+00  1.00000000e+00]
 [ 0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00]
 [ 0.00000000e+00 -3.99698718e+00 -5.40302306e-01  1.58228342e-16]
 [ 1.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00]
 [ 0.00000000e+00  2.56643595e+00 -8.41470985e-01  8.98254244e-16]]
Body twist V_b given the body Jacobian and joint velocities:
 [[ 0.        ]
 [ 0.6       ]
 [ 0.        ]
 [-0.96148813]
 [ 0.1       ]
 [ 0.2608459 ]]
```

Inverse Kinematics Practice

In [15]:
```
# Desired end effector pose
T_1in0 = np.array([[-0.56364669, 0.63986581, -0.52237358, -7.20790756], [0.82386305, 0.38985626, -0.41141436, 3.06907233], [-0
# Initial pose of the end effector
M = np.array([[-1.00000000, 0.00000000, 0.00000000, -8.00000000], [0.00000000, 1.00000000, 0.00000000, 0.00000000], [0.0000000
# Fixed frame screw axes
S = np.array([[0.00000000, 0.00000000, -1.00000000, 0.00000000, 1.00000000], [0.00000000, 0.00000000, 0.00000000, 1.00000000,
# Initial guess for joint angles
theta_guess = np.array([0.,0.,0.,0.,0.])
# Error tolerances
eomg, ev = 0.01, 0.01

# Success is true if the function converged to a solution within the specified error tolerances, and false otherwise.
theta_out, success = IKinSpace(S, M, T_1in0, theta_guess, eomg, ev)
theta_out = theta_out.reshape(1,-1).T
print("Success:", success)
print("Output joint angles (theta_out):\n", theta_out.tolist())
```

```
Success: True
Output joint angles (theta_out):
 [[0.6768793845436224], [-0.9862744915154902], [0.25544840175290606], [-0.06068358212228473], [-0.46949147679369435]]
```