63340500025 / 63340500066

Question 1

```python
# Question 1
def endEffectorJacobianHW2(q):
    '''
    Input -
        q : format list 1x3 [[i_11, i_12, i_13]]
        q unit: rad
    ==================================================
    Procedure -
        1. Compute Rotation Matrix of Frame i vs. Global Frame, Position Matrix of Frame i vs. Global Frame and Position Matrix of End Effector vs. Global Frame.
        2. Declare a z-axis vector [0,0,1] in the global frame.
        3. Multiply each rotational matrix (R1, R2, R3) with the z-axis vector.
        4. Compute the cross product of angular Jacobian matrix and the position vector from each frame to the end effector.
        5. Concatenate the angular and linear Jacobian matrices.
    '''
    # 1. Compute Rotation Matrix of Frame i vs. Global Frame, Position Matrix of Frame i vs. Global Frame and Position Matrix of End Effector vs. Global Frame.
    R,P,_,p_e = FKHW2(q)

    '''
    where -
        R = Rotation Matrix of Frame i vs. Global Frame [3x3x4]
        P = Position Matrix of Frame i vs. Global Frame [3x4]
        p_e = Position Matrix of End Effector vs. Global Frame [1x3]
    '''
    R1,R2,R3 = R[:,:,0], R[:,:,1], R[:,:,2]   # → R^0_j
    P1,P2,P3 = P[:,0], P[:,1], P[:,2]         # → p^0_j

    # 2. Declare a z-axis vector [0,0,1] in the global frame.
    rotateZ = np.array([0, 0, 1]).reshape(3, 1)   # → ẑ^j_j

    # 3. Multiply each rotational matrix (R1, R2, R3) with the z-axis vector.
    angularJ01 = R1 @ rotateZ
    angularJ02 = R2 @ rotateZ
    angularJ03 = R3 @ rotateZ                 # } J^i_ω(q) (แยกเป็น Joint 1-3)
    # We will get angular Jacobian matrix [3x3]
    angularJacobian = np.concatenate((angularJ01, angularJ02, angularJ03), axis=1)   # → J^e_ω(q)

    # 4. Compute the cross product of angular Jacobian matrix and the position vector of each frame to the end effector.
    linearJ01 = np.cross(angularJ01.reshape(1, 3), (p_e - P1)).reshape(3, 1)
    linearJ02 = np.cross(angularJ02.reshape(1, 3), (p_e - P2)).reshape(3, 1)
    linearJ03 = np.cross(angularJ03.reshape(1, 3), (p_e - P3)).reshape(3, 1)   # } J^i_v(q) (แยกเป็น Joint 1-3)
    # We will get linear Jacobian matrix [3x3]
    linearJacobian = np.concatenate((linearJ01, linearJ02, linearJ03), axis=1)   # → J^e_v(q)

    '''
    Output -
        return format list 6x3
        [ [i_11, i_12, i_13],
          [i_21, i_22, i_23],
          [i_31, i_32, i_33],
          [i_41, i_42, i_43],
          [i_51, i_52, i_53],
          [i_61, i_62, i_63] ]
    '''
    # 5. Concatenate the angular and linear Jacobian matrices.
    return np.concatenate((angularJacobian, linearJacobian), axis=0)
```

$J^e_{\omega,j} = \rho^1_j \hat{z}^0_j = \rho^1_j (R^0_j \hat{z}^j_j)$

Return Jacobian Matrix

$$J^e(\vec{q}) = \begin{bmatrix} J^e_\omega(\vec{q}) \\ J^e_v(\vec{q}) \end{bmatrix}$$

Jacobian Matrix

$$\begin{bmatrix} J^e_\omega(\vec{q}) \\ J^e_v(\vec{q}) \end{bmatrix}$$

Angular Jacobian Matrix

$J^e_\omega(\vec{q}) = [\, J^e_{\omega,1} \,\vdots\, J^e_{\omega,2} \,\vdots\, \cdots \,\vdots\, J^e_{\omega,e} \,]$

$J^e_{\omega,j} = \rho^1_j \hat{Z}^0_j$ โดยที่ j คือ Frame ของ Joint

Linear Jacobian Matrix

$J^e_v(\vec{q}) = [\, J^e_{v,1} \,\vdots\, J^e_{v,2} \,\vdots\, \cdots \,\vdots\, J^e_{v,e} \,]$

$J^e_{v,j} = \rho^1_j (\hat{Z}^0_j \times (P^0_{0,e} - P^0_{0,j-1}) + (1-\rho^1_j) \hat{Z}^0_j$

Question 2

การคำนวณ Singularity ของหุ่นยนต์จะใช้สูตร

$$J^{-1} = \frac{1}{\det(J)} \cdot adj(J)$$

โจทย์กำหนดให้ใช้ Linear Jacobian Matrix ในการคำนวณ จะได้

$$J^e_v(\vec{q})^{-1} = \frac{1}{\det(J^e_v(\vec{q}))} \cdot adj(J^e_\omega(\vec{q}))$$

โดยหุ่นยนต์จะเข้าใกล้สภาวะ Singularity ก็ต่อเมื่อ

$$\| \det(J^e_v(\vec{q})) \| < \varepsilon \quad \text{โดยที่ } \varepsilon = 0.001$$

```python
# Question 2
def checkSingularityHW2(q):
    '''
    Input -
        q : format list 1x3 [[i_11, i_12, i_13]]
        q unit: rad
    ==================================================
    Procedure -
        1. Compute the Jacobian matrix, then select the last 3 rows (Linear Jacobian).
        2. Compute the determinant of the Jacobian matrix.
            - If the absolute of determinant is less than 0.001, then the robot is approaching a singularity state (True).
    '''

    # 1. Compute the Jacobian matrix, then select the last 3 rows (Linear Jacobian).
    endJacob = endEffectorJacobianHW2(q)[3:6,:]   # → J^e_ω(q)
    '''
    where -
        endJacob = End Effector Jacobian Matrix [6x3]
    ==================================================
    Output -
        return format bool
    '''
    # 2. Compute the determinant of the Jacobian matrix.
    #       If the absolute of determinant is less than 0.001, then the robot is approaching a singularity state (True).
    return np.abs(np.linalg.det(endJacob)) < 0.001
```

Question 3

```python
# Question 3
def computeEffortHW2(q,w):
    '''
    Input -
        q : format list 1x3 [[i_11, i_12, i_13]]
        q unit: rad
        type something here
    ==================================================
    Procedure -
        1. Compute the Rotational Matrix of End Effector vs. Global Frame (R_e) and Jacobian matrix (endJacob).
        2. Transpose wrench vector (w array) to make it a column vector and separate moment and force vector.
        3. Multiply moment and force vector with Rotational Matrix of End Effector vs. Global Frame (R_e) to get the wrench vector in the end effector frame.
        4. Multiply the wrench vector in the end effector frame with the transposed Jacobian matrix (endJacob) to get the joint effort vector.
        5. Transpose the joint effort vector to make it a row vector.
    '''
    # 1. Compute the Rotational Matrix of End Effector vs. Global Frame (R_e) and Jacobian matrix (endJacob).
    _,_,R_e,_ = FKHW2(q)
    endJacob = endEffectorJacobianHW2(q)

    '''
    where -
        R_e = Rotational Matrix of End Effector vs. Global Frame [3x3]
        endJacob = End Effector Jacobian Matrix [6x3]
    '''
    # 2. Transpose wrench vector (w array) to make it a column vector and separate moment and force vector.
    wt = np.transpose(w)

    # 3. Multiply moment and force vector with Rotational Matrix of End Effector vs. Global Frame (R_e) to get the wrench vector in the end effector frame.
    N0e = R_e @ wt[0:3]
    F0e = R_e @ wt[3:6]
    W0e = np.concatenate((N0e,F0e), axis = 0)   # } หา W^0_e
    # 4. Multiply the wrench vector in the end effector frame with the transposed Jacobian matrix (endJacob) to get the joint effort vector.
    # This is in column vector format. We need to transpose it to get the row vector.
    tau = np.transpose(endJacob) @ W0e   # ↳ คำนวณ τ
    '''
    Output -
        return format list 1x3
        [ [i_11, i_12, i_13] ]
    '''
    # 5. Transpose the joint effort vector to make it a row vector.
    return np.transpose(tau)
```

การคำนวณหา Effort (τ)

จากสูตร

$$\vec{\tau} = J^e(\vec{q})^T \vec{W}^0_e$$

โดยที่โจทย์ให้

$W^e_e = \begin{bmatrix} n^e \\ f^e \end{bmatrix}$; Wrench ที่ประกอบไปด้วย Moment ($\vec{n}^e$) และ Force ($\vec{f}^e$)

$\begin{bmatrix} n_x \\ n_y \\ n_z \end{bmatrix}$ $\begin{bmatrix} f_x \\ f_y \\ f_z \end{bmatrix}$

หา $\vec{W}^0_e$ โดย

$w^e_e = \begin{bmatrix} n^e_e \\ f^e_e \end{bmatrix}$ และ $n^0_e = R^0_e n^e_e$, $f^0_e = R^0_e f^e_e$

จากนั้นทำการคำนวณ $\vec{\tau}$