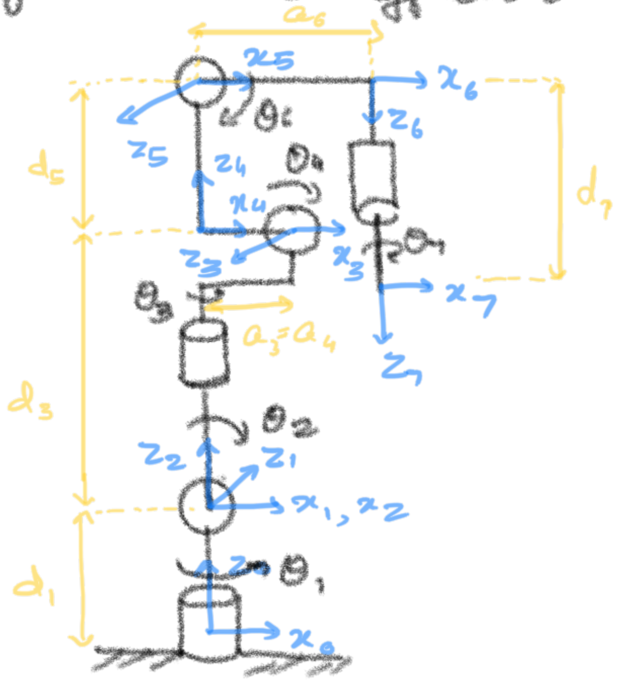
Prelab 3

1. Given: $\dot{q}_1, \dot{q}_2, \dot{q}_3, \dot{q}_4, \dot{q}_5, \dot{q}_6, \dot{q}_7$

To find: End effector $V_x, V_y, V_z, \omega_x, \omega_y, \omega_z$



$d_1 = 0.333\,m$

$d_3 = 0.316\,m$

$a_3 = a_4 = 0.0825\,m$

$d_5 = 0.384\,m$

$d_6 = 0.088\,m$

$d_7 = 0.21\,m$

DH Table ($\theta_5^* = 0$)

Link	a_i	α_i	d_i	θ_i
1	0	−90	d_1	θ_1^*
2	0	90	0	θ_2^*
3	a_3	90	d_3	θ_3^*
4	$-a_4$	−90	0	θ_4^*
5	0	90	d_5	0
6	a_6	90	0	θ_6^*
7	0	0	d_7	$\theta_7^* - 45°$

Approach

1. Finding $T_1^0, T_2^0, T_3^0, T_4^0, T_5^0, T_6^0, T_7^0$
2. Obtaining $\vec{O}_1^0, \vec{O}_2^0, \vec{O}_3^0, \vec{O}_4^0, \vec{O}_5^0, \vec{O}_6^0, \vec{O}_7^0$
3. Obtaining $R_1^0, R_2^0, R_3^0, R_4^0, R_5^0, R_6^0, R_7^0$
4. using $J_{v_i} = \hat{z}_{i-1} \times (\vec{O}_n - \vec{O}_{i-1})$ as all joints are revolute (using geometric approach)
5. using $J_{\omega_i} = \rho_i (R_{i-1}^0 \hat{z})$ where $\rho_i = 1$ for all cases, as all joints are revolute (geometric approach)
6. $J = [J_1 \; J_2 \; J_3 \; J_4 \; J_5 \; J_6 \; J_7]$

$J_v = [J_{v_1}\ J_{v_2}\ J_{v_3}\ J_{v_4}\ J_{v_5}\ J_{v_6}\ J_{v_7}]$

$J_\omega = [J_{\omega_1}\ J_{\omega_2}\ J_{\omega_3}\ J_{\omega_4}\ J_{\omega_5}\ J_{\omega_6}\ J_{\omega_7}]$

(7)
$$\begin{bmatrix} v_x \\ v_y \\ v_z \\ \omega_x \\ \omega_y \\ \omega_z \end{bmatrix} = \begin{bmatrix} J_v \\ J_\omega \end{bmatrix} \begin{bmatrix} \dot{q}_1 \\ \dot{q}_2 \\ \dot{q}_3 \\ \dot{q}_4 \\ \dot{q}_5 \\ \dot{q}_6 \\ \dot{q}_7 \end{bmatrix}$$

Steps (1), (2), (3) are in the attached ipynb file.

(4)
$$J_{v_i} = \hat{z}_{i-1} \times (\vec{O}_n - \vec{O}_{i-1})$$

$$\therefore J_{v_1} = \hat{z}_0 \times (\vec{O}_7 - \vec{O}_0)$$

$$= \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \times \left(\begin{bmatrix} \vec{O}_{7x} \\ \vec{O}_{7y} \\ \vec{O}_{7z} \end{bmatrix} - \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \right) = \begin{bmatrix} -\vec{O}_{7y} \\ \vec{O}_{7x} \\ 0 \end{bmatrix}$$

$$J_{v_2} = \hat{z}_1 \times (\vec{O}_7 - \vec{O}_1)$$

$$= \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} \times \left(\begin{bmatrix} \vec{O}_{7x} \\ \vec{O}_{7y} \\ \vec{O}_{7z} \end{bmatrix} - \begin{bmatrix} 0 \\ 0 \\ 0.333 \end{bmatrix} \right) = \begin{bmatrix} \vec{O}_{7z} - 0.333 \\ 0 \\ -\vec{O}_{7x} \end{bmatrix}$$

$$J_{v_3} = \hat{z}_2 \times (\vec{O}_7 - \vec{O}_2)$$

$$= \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \times \left(\begin{bmatrix} \vec{O}_{7x} \\ \vec{O}_{7y} \\ \vec{O}_{7z} \end{bmatrix} - \begin{bmatrix} 0 \\ 0 \\ 0.333 \end{bmatrix} \right) = \begin{bmatrix} -\vec{O}_{7y} \\ \vec{O}_{7x} \\ 0 \end{bmatrix}$$

$$J_{v_4} = \hat{z}_3 \times (\vec{O}_7 - \vec{O}_3)$$

$$= \begin{bmatrix} 0 \\ -1 \\ 0 \end{bmatrix} \times \left(\begin{bmatrix} \vec{O}_{7x} \\ \vec{O}_{7y} \\ \vec{O}_{7z} \end{bmatrix} - \begin{bmatrix} \vec{O}_{3x} \\ \vec{O}_{3y} \\ \vec{O}_{3z} \end{bmatrix} \right)$$

$$= \begin{bmatrix} -(\vec{O}_{7z} - \vec{O}_{3z}) \\ 0 \\ (\vec{O}_{7x} - \vec{O}_{3x}) \end{bmatrix}$$

$$J_{v_5} = \hat{z}_4 \times (\vec{O}_7 - \vec{O}_4)$$

$$J_{v5} = \hat{z}_4 \times (\vec{O}_7 - \vec{O}_4)$$

$$= \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \times \begin{bmatrix} \vec{O}_{7x} - \vec{O}_{4x} \\ \vec{O}_{7y} - \vec{O}_{4y} \\ \vec{O}_{7z} - \vec{O}_{4z} \end{bmatrix} = \begin{bmatrix} -(\vec{O}_{7z} - \vec{O}_{4z}) \\ (\vec{O}_{7x} - \vec{O}_{4x}) \\ 0 \end{bmatrix}$$

$$J_{v6} = \hat{z}_5 \times (\vec{O}_7 - \vec{O}_5)$$

$$= \begin{bmatrix} 0 \\ -1 \\ 0 \end{bmatrix} \times \begin{bmatrix} \vec{O}_{7x} - \vec{O}_{5x} \\ \vec{O}_{7y} - \vec{O}_{5y} \\ \vec{O}_{7z} - \vec{O}_{5z} \end{bmatrix} = \begin{bmatrix} -(\vec{O}_{7z} - \vec{O}_{5z}) \\ 0 \\ (\vec{O}_{7x} - \vec{O}_{5x}) \end{bmatrix}$$

$$J_{v7} = \hat{z}_6 \times (\vec{O}_7 - \vec{O}_6)$$

$$= \begin{bmatrix} 0 \\ 0 \\ -1 \end{bmatrix} \times \begin{bmatrix} \vec{O}_{7x} - \vec{O}_{6x} \\ \vec{O}_{7y} - \vec{O}_{6y} \\ \vec{O}_{7z} - \vec{O}_{6z} \end{bmatrix} = \begin{bmatrix} \vec{O}_{7y} - \vec{O}_{6y} \\ -(\vec{O}_{7x} - \vec{O}_{6x}) \\ 0 \end{bmatrix}$$

$$J_v = \begin{bmatrix} J_{v1} & J_{v2} & J_{v3} & J_{v4} & J_{v5} & J_{v6} & J_{v7} \end{bmatrix}$$

$$J_v = \left[\begin{array}{c|c|c|c|c|c|c} -\vec{O}_{7y} & \vec{O}_{7z} - 0.333 & -\vec{O}_{7y} & -(\vec{O}_{7z} - \vec{O}_{3z}) & -(\vec{O}_{7z} - \vec{O}_{4z}) & -(\vec{O}_{7z} - \vec{O}_{5z}) & \vec{O}_{7y} - \vec{O}_{6y} \\ \vec{O}_{7x} & 0 & \vec{O}_{7x} & 0 & \vec{O}_{7x} - \vec{O}_{4x} & 0 & -(\vec{O}_{7x} - \vec{O}_{6x}) \\ 0 & -\vec{O}_{7x} & 0 & (\vec{O}_{7x} - \vec{O}_{3x}) & 0 & \vec{O}_{7x} - \vec{O}_{5x} & 0 \end{array}\right]$$

⑤ $J_{\omega_i} = \rho_i (R^0_{i-1} \hat{z})$

$$J_{\omega_1} = 1 (R_0^0 \hat{z}) = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$$

$$J_{\omega_2} = 1 (R_1^0 \hat{z})$$

$$J_{\omega_3} = R_2^0 \hat{z}$$

$$J_{\omega_4} = R_3^0 \hat{z}$$

$$J_{\omega_5} = R_4^0 \hat{z}$$

$$J_{\omega_6} = R_5^0 \hat{z}$$

$$J_{\omega_7} = R_6^0 \hat{z}$$

$J_\omega = [J_{\omega 1}\ J_{\omega 2}\ J_{\omega 3}\ J_{\omega 4}\ J_{\omega 5}\ J_{\omega 6}\ J_{\omega 7}]$

For complete expression see ipynb file

2. Linear velocity end effector when $\dot{q}_1 = 50$ rad/s

$\dot{q}_2, \dot{q}_3, \dot{q}_4, \dot{q}_5, \dot{q}_6, \dot{q}_7 = 0$

$V_x = 0$

$V_y = 4.4$ m/s

$V_z = 0$

Yes FK reflects this velocity

3. <u>Condition for no solution:</u>

Under singular configuration., Eg: $\theta_3 = 0$, in general, when $\det(J_v) = 0$

<u>Condition for unique solution:</u>

Overdetermined system. (kinematically deficient robot). i.e. when rank(J) $\leq$ min(n, 6)

<u>Condition for infinitely many solutions:</u>

Underdetermined System (redundant robot like franka arm) i.e. when rank (J) $\leq$ min(m, 6)

4. Lab testing plan:

After writing code for velocity FK & IK, I shall use an arbitrary set of joint speed values & obtain some E.E. velocity values by passing into velocity FK. Using these E.E. velocity values in velocity IK, will see if I obtain the same joint velocities that I started with.

For the hardware testing, I will use a tachometer at each joint motor to get the values of joint velocities (when E.E values are set for testing)

```
import sympy as sym
import math

a = [0, 0, 0.0825, -0.0825, 0, 0.088, 0]                                              #sym.Symbol('a3')
alpha = [-math.pi/2, math.pi/2, math.pi/2, -math.pi/2, math.pi/2, math.pi/2, 0]
d = [0.333, 0, 0.316, 0, 0.384, 0, 0.210]                                              #sym.Symbol('d5')
theta = [sym.Symbol('theta_1'), sym.Symbol('theta_2'), sym.Symbol('theta_3'), sym.Symbol('theta4'), 0, sym.Symbol('theta_6'), sym.Symbol('theta_
zero_config = [0,0,0,0,0,0,0]

def T_calc(a, alpha, d, theta):
  T = []
  for i in range(7):
      c, s = sym.cos(theta[i]), sym.sin(theta[i])
      ca, sa = math.cos(alpha[i]), math.sin(alpha[i])
      t = sym.Matrix([[c, -s*ca, s*sa, a[i]*c], [ s, c*ca, -c*sa, a[i]*s], [0, sa, ca, d[i]], [0, 0, 0, 1]])
      T.append(t)

  return T

def T_wrt_0(T):
  T_1_0 = sym.simplify(sym.nsimplify(T[0],tolerance=1e-10,rational=True))
  T_2_0 = sym.simplify(sym.nsimplify(T[0] * T[1],tolerance=1e-10,rational=True))
  T_3_0 = sym.simplify(sym.nsimplify(T[0] * T[1] * T[2],tolerance=1e-10,rational=True))
  T_4_0 = sym.simplify(sym.nsimplify(T[0] * T[1] * T[2] * T[3],tolerance=1e-10,rational=True))
  T_5_0 = sym.simplify(sym.nsimplify(T[0] * T[1] * T[2] * T[3] * T[4],tolerance=1e-10,rational=True))
  T_6_0 = sym.simplify(sym.nsimplify(T[0] * T[1] * T[2] * T[3] * T[4] * T[5],tolerance=1e-10,rational=True))
  T_7_0 = sym.simplify(sym.nsimplify(T[0] * T[1] * T[2] * T[3] * T[4] * T[5] * T[6],tolerance=1e-10,rational=True))

  return T_1_0, T_2_0, T_3_0, T_4_0, T_5_0, T_6_0, T_7_0

def o_wrt_0(T_1_0, T_2_0, T_3_0, T_4_0, T_5_0, T_6_0, T_7_0):
  o_1_0 = sym.Matrix([T_1_0[3], T_1_0[7], T_1_0[11]])
  o_2_0 = sym.Matrix([T_2_0[3], T_2_0[7], T_2_0[11]])
  o_3_0 = sym.Matrix([T_3_0[3], T_3_0[7], T_3_0[11]])
  o_4_0 = sym.Matrix([T_4_0[3], T_4_0[7], T_4_0[11]])
  o_5_0 = sym.Matrix([T_5_0[3], T_5_0[7], T_5_0[11]])
  o_6_0 = sym.Matrix([T_6_0[3], T_6_0[7], T_6_0[11]])
  o_7_0 = sym.Matrix([T_7_0[3], T_7_0[7], T_7_0[11]])

  return o_1_0, o_2_0, o_3_0, o_4_0, o_5_0, o_6_0, o_7_0

def R_wrt_0(T_1_0, T_2_0, T_3_0, T_4_0, T_5_0, T_6_0, T_7_0):
  R_1_0 = sym.Matrix([[T_1_0[0], T_1_0[1], T_1_0[2]], [T_1_0[4], T_1_0[5], T_1_0[6]], [T_1_0[8], T_1_0[9], T_1_0[10]]])
  R_2_0 = sym.Matrix([[T_2_0[0], T_2_0[1], T_2_0[2]], [T_2_0[4], T_2_0[5], T_2_0[6]], [T_2_0[8], T_2_0[9], T_2_0[10]]])
  R_3_0 = sym.Matrix([[T_3_0[0], T_3_0[1], T_3_0[2]], [T_3_0[4], T_3_0[5], T_3_0[6]], [T_3_0[8], T_3_0[9], T_3_0[10]]])
  R_4_0 = sym.Matrix([[T_4_0[0], T_4_0[1], T_4_0[2]], [T_4_0[4], T_4_0[5], T_4_0[6]], [T_4_0[8], T_4_0[9], T_4_0[10]]])
  R_5_0 = sym.Matrix([[T_5_0[0], T_5_0[1], T_5_0[2]], [T_5_0[4], T_5_0[5], T_5_0[6]], [T_5_0[8], T_5_0[9], T_5_0[10]]])
  R_6_0 = sym.Matrix([[T_6_0[0], T_6_0[1], T_6_0[2]], [T_6_0[4], T_6_0[5], T_6_0[6]], [T_6_0[8], T_6_0[9], T_6_0[10]]])
  R_7_0 = sym.Matrix([[T_7_0[0], T_7_0[1], T_7_0[2]], [T_7_0[4], T_7_0[5], T_7_0[6]], [T_7_0[8], T_7_0[9], T_7_0[10]]])

  return R_1_0, R_2_0, R_3_0, R_4_0, R_5_0, R_6_0, R_7_0

def cross(b, c):
    t1 = b[1]*c[2] - c[1]*b[2]
    t2 = -(b[0]*c[2] - b[2]*c[0])
    t3 = b[0]*c[1] - b[1]*c[0]
    return sym.Matrix([t1, t2, t3])


def Jv_calc(o_1_0, o_2_0, o_3_0, o_4_0, o_5_0, o_6_0, o_7_0):
  z0 = sym.Matrix([0,0,1])
  z1 = sym.Matrix([0,1,0])
  z2 = sym.Matrix([0,0,1])
  z3 = sym.Matrix([0,-1,0])
  z4 = sym.Matrix([0,0,1])
  z5 = sym.Matrix([0,-1,0])
  z6 = sym.Matrix([0,0,-1])
  Jv1 = sym.simplify(cross(z0, o_7_0))
  Jv2 = sym.simplify(cross(z1, o_7_0 - o_1_0))
  Jv3 = sym.simplify(cross(z2, o_7_0 - o_2_0))
  Jv4 = sym.simplify(cross(z3, o_7_0 - o_3_0))
  Jv5 = sym.simplify(cross(z4, o_7_0 - o_4_0))
  Jv6 = sym.simplify(cross(z5, o_7_0 - o_5_0))
  Jv7 = sym.simplify(cross(z6, o_7_0 - o_6_0))
  Jv = sym.Matrix([Jv1.T, Jv2.T, Jv3.T, Jv4.T, Jv5.T, Jv6.T, Jv7.T])  #yields a row major ordered matrix (7 rows 3 columns)
  Jv = Jv.T #converts to 3 rows and 7 columns
  print('Jv shape =', Jv.shape)
  return Jv


def Jw_calc(R_1_0, R_2_0, R_3_0, R_4_0, R_5_0, R_6_0, R_7_0):
  z_hat = sym.Matrix([0,0,1])

  Jw1 = sym.simplify(z_hat)
  Jw2 = sym.simplify(R_1_0*z_hat)
  Jw3 = sym.simplify(R_2_0*z_hat)
  Jw4 = sym.simplify(R_3_0*z_hat)
  Jw5 = sym.simplify(R_4_0*z_hat)
  Jw6 = sym.simplify(R_5_0*z_hat)
  Jw7 = sym.simplify(R_6_0*z_hat)
  Jw = sym.Matrix([Jw1.T, Jw2.T, Jw3.T, Jw4.T, Jw5.T, Jw6.T, Jw7.T])  #yields a row major ordered matrix (7 rows 3 columns)
  Jw = Jw.T #converts to 3 rows and 7 columns
  print('Jw shape =', Jw.shape)
  return Jw


def J(Jv, Jw):
  J = sym.Matrix([Jv, Jw])
  print("Jacobian shape is =", J.shape)
  return J


#function calls (symbolic)
T = T_calc(a, alpha, d, theta)
T_1_0, T_2_0, T_3_0, T_4_0, T_5_0, T_6_0, T_7_0 = T_wrt_0(T)
o_1_0, o_2_0, o_3_0, o_4_0, o_5_0, o_6_0, o_7_0 = o_wrt_0(T_1_0, T_2_0, T_3_0, T_4_0, T_5_0, T_6_0, T_7_0)
R_1_0, R_2_0, R_3_0, R_4_0, R_5_0, R_6_0, R_7_0 = R_wrt_0(T_1_0, T_2_0, T_3_0, T_4_0, T_5_0, T_6_0, T_7_0)

print("T_1_0 =", T_1_0)
print("T_2_0 =", T_2_0)
```

```
print("T_3_0 =", T_3_0)
print("T_4_0 =", T_4_0)
print("T_5_0 =", T_5_0)
print("T_6_0 =", T_6_0)
print("T_7_0 =", T_7_0)
```

```
T_1_0 = Matrix([[cos(theta_1), 0, -sin(theta_1), 0], [sin(theta_1), 0, cos(theta_1), 0], [0, -1, 0, 333/1000], [0, 0, 0, 1]])
T_2_0 = Matrix([[cos(theta_1)*cos(theta_2), -sin(theta_1), sin(theta_2)*cos(theta_1), 0], [sin(theta_1)*cos(theta_2), cos(theta_1), sin(the
T_3_0 = Matrix([[-sin(theta_1)*sin(theta_3) + cos(theta_1)*cos(theta_2)*cos(theta_3), sin(theta_2)*cos(theta_1), sin(theta_1)*cos(theta_3)
T_4_0 = Matrix([[-(sin(theta_1)*sin(theta_3) - cos(theta_1)*cos(theta_2)*cos(theta_3))*cos(theta4) + sin(theta4)*sin(theta_2)*cos(theta_1),
T_5_0 = Matrix([[-(sin(theta_1)*sin(theta_3) - cos(theta_1)*cos(theta_2)*cos(theta_3))*cos(theta4) + sin(theta4)*sin(theta_2)*cos(theta_1),
T_6_0 = Matrix([[-sin(theta_1)*sin(theta_3)*cos(theta4 + theta_6) + sin(theta_2)*sin(theta4 + theta_6)*cos(theta_1) + cos(theta_1)*cos(thet
T_7_0 = Matrix([[(sin(theta_1)*cos(theta_3) + sin(theta_3)*cos(theta_1)*cos(theta_2))*sin(theta_7 - 3169251833/4035216761) - (sin(theta_1)*
```

```
print("o_1_0 =", o_1_0)
print("o_2_0 =", o_2_0)
print("o_3_0 =", o_3_0)
print("o_4_0 =", o_4_0)
print("o_5_0 =", o_5_0)
print("o_6_0 =", o_6_0)
print("o_7_0 =", o_7_0)
```

```
o_1_0 = Matrix([[0], [0], [333/1000]])
o_2_0 = Matrix([[0], [0], [333/1000]])
o_3_0 = Matrix([[-33*sin(theta_1)*sin(theta_3)/400 + 79*sin(theta_2)*cos(theta_1)/250 + 33*cos(theta_1)*cos(theta_2)*cos(theta_3)/400], [79
o_4_0 = Matrix([[33*(sin(theta_1)*sin(theta_3) - cos(theta_1)*cos(theta_2)*cos(theta_3))*cos(theta4)/400 - 33*sin(theta4)*sin(theta_2)*cos(
o_5_0 = Matrix([[48*(sin(theta_1)*sin(theta_3) - cos(theta_1)*cos(theta_2)*cos(theta_3))*sin(theta4)/125 + 33*(sin(theta_1)*sin(theta_3) -
o_6_0 = Matrix([[48*sin(theta4)*sin(theta_1)*sin(theta_3)/125 - 33*sin(theta4)*sin(theta_2)*cos(theta_1)/400 - 48*sin(theta4)*cos(theta_1)*
o_7_0 = Matrix([[48*sin(theta4)*sin(theta_1)*sin(theta_3)/125 - 33*sin(theta4)*sin(theta_2)*cos(theta_1)/400 - 48*sin(theta4)*cos(theta_1)*
```

```
print("R_1_0 =", R_1_0)
print("R_2_0 =", R_2_0)
print("R_3_0 =", R_3_0)
print("R_4_0 =", R_4_0)
print("R_5_0 =", R_5_0)
print("R_6_0 =", R_6_0)
print("R_7_0 =", R_7_0)
```

```
R_1_0 = Matrix([[cos(theta_1), 0, -sin(theta_1)], [sin(theta_1), 0, cos(theta_1)], [0, -1, 0]])
R_2_0 = Matrix([[cos(theta_1)*cos(theta_2), -sin(theta_1), sin(theta_2)*cos(theta_1)], [sin(theta_1)*cos(theta_2), cos(theta_1), sin(theta_
R_3_0 = Matrix([[-sin(theta_1)*sin(theta_3) + cos(theta_1)*cos(theta_2)*cos(theta_3), sin(theta_2)*cos(theta_1), sin(theta_1)*cos(theta_3)
R_4_0 = Matrix([[-(sin(theta_1)*sin(theta_3) - cos(theta_1)*cos(theta_2)*cos(theta_3))*cos(theta4) + sin(theta4)*sin(theta_2)*cos(theta_1),
R_5_0 = Matrix([[-(sin(theta_1)*sin(theta_3) - cos(theta_1)*cos(theta_2)*cos(theta_3))*cos(theta4) + sin(theta4)*sin(theta_2)*cos(theta_1),
R_6_0 = Matrix([[-sin(theta_1)*sin(theta_3)*cos(theta4 + theta_6) + sin(theta_2)*sin(theta4 + theta_6)*cos(theta_1) + cos(theta_1)*cos(thet
R_7_0 = Matrix([[(sin(theta_1)*cos(theta_3) + sin(theta_3)*cos(theta_1)*cos(theta_2))*sin(theta_7 - 3169251833/4035216761) - (sin(theta_1)*
```

```
Jv = Jv_calc(o_1_0, o_2_0, o_3_0, o_4_0, o_5_0, o_6_0, o_7_0)
Jw = Jw_calc(R_1_0, R_2_0, R_3_0, R_4_0, R_5_0, R_6_0, R_7_0)
Jacobian = J(Jv, Jw)
```

```
Jv shape = (3, 7)
Jw shape = (3, 7)
Jacobian shape is = (6, 7)
```

```
print("Jacobian is : ", Jacobian)
```

```
Jacobian is :  Matrix([[33*sin(theta4)*sin(theta_1)*sin(theta_2)/400 + 48*sin(theta4)*sin(theta_1)*cos(theta_2)*cos(theta_3)/125 + 48*sin(t
```

```
#function calls (example)
T = T_calc(a, alpha, d, zero_config)
T10, T20, T30, T40, T50, T60, T70 = T_wrt_0(T)
o10, o20, o30, o40, o50, o60, o70 = o_wrt_0(T10, T20, T30, T40, T50, T60, T70)
R10, R20, R30, R40, R50, R60, R70 = R_wrt_0(T10, T20, T30, T40, T50, T60, T70)
q_dot = sym.Matrix([50, 0, 0, 0, 0, 0, 0])  #assuming q1 velocity to be non zero
Jv = Jv_calc(o10, o20, o30, o40, o50, o60, o70)
Jw = Jw_calc(R10, R20, R30, R40, R50, R60, R70)
J = J(Jv, Jw)

v = J*q_dot
```

```
Jv shape = (3, 7)
Jw shape = (3, 7)
Jacobian shape is = (6, 7)
```

```
print("Jacobian is :", J)
print("End effector velocity is :", v)
```

```
Jacobian is : Matrix([[0, 49/100, 0, -87/500, 0, 21/100, 0], [11/125, 0, 11/125, 0, 11/125, 0, 0], [0, -11/125, 0, 11/2000, 0, 11/125, 0],
End effector velocity is : Matrix([[0], [22/5], [0], [0], [0], [50]])
```