Robotics project

Analyzing 6-DOF UR10 robot arm

Phase2 (Inverse velocity and Dynamics)

Import libraries

In [1]:
```
from sympy import *
import numpy as np
import math
from math import degrees
```

Forward Kinematics

In [2]:
```
# Define the symbolic variables
num_symbols = 6  # Number of symbols to generate
alpha_names = [f"alpha{i}" for i in range(num_symbols+1)]
a_names = [f"a{i}" for i in range(num_symbols+1)]
d_names = [f"d{i}" for i in range(num_symbols+1)]
theta_names = [f"theta{i}" for i in range(num_symbols+1)]

alpha = symbols(alpha_names)
a = symbols(a_names)
d = symbols(d_names)

# our thetas are function of time
theta = []
t = Symbol('t')
for i in theta_names:
    theta.append(Function(i)(t))

# Define DH table
DH_param = [[0, 0, d[1], theta[1]],
            [pi/2, 0, 0, theta[2]],
            [0, a[2], 0, theta[3]],
            [0, a[3], d[4], theta[4]],
            [-pi/2, 0, d[5], theta[5]],
            [pi/2, 0, d[6], theta[6]]]

# find homogeneous transformations(T_i_i-1)
T = []
for i in range(6):
    T_temp = [[cos(DH_param[i][3]), -sin(DH_param[i][3]), 0, DH_param[i][1]],
              [sin(DH_param[i][3])*cos(DH_param[i][0]),  cos(DH_param[i][3])*cos(DH_param[i][0]), -sin(DH_param[i][0]), -sin(DH_param[i][0])*(DH_param[i][2])],
              [sin(DH_param[i][3])*sin(DH_param[i][0]), cos(DH_param[i][3])*sin(DH_param[i][0]), cos(DH_param[i][0]), cos(DH_param[i][0])*DH_param[i][2]],
              [0, 0, 0, 1]]
    T.append(T_temp)
```

Part1

Finding Jacobian Matrix

1-1) Find T_0_i (like T_0_1, T_0_2, ...)

In [3]:
```
T_0_i = [T[0]]
result = T[0]
for n in range(1,len(T)):
    # Multiply the matrices
    result = [[sum(result[i][k] * T[n][k][j] for k in range(4)) for j in range(4)] for i in range(4)]
    # Simplify the result according to cosine multiplication rules
    simplified_result = [[trigsimp(expr) for expr in row] for row in result]
    T_0_i.append(simplified_result)
```

1-2) Find zi and oi

In [4]:
```
z = []
o = []
for T0i in T_0_i:
    z.append(np.array(T0i)[:3,2][np.newaxis].T)
    o.append(np.array(T0i)[:3,3][np.newaxis].T)
```

1-3) Creat J(theta)

Here all joints are Revolute. So we use below formula for creating J(theta):

```
Ji = [[zi*(oe-oi)],[zi]]
```

In [5]:
```
J = []
for i in range(len(z)):
    Jvi = np.cross(z[i].T, (o[len(o)-1]-o[i]).T).T
    Jwi = z[i]
    Ji = np.append(Jvi,Jwi)
    J.append(Ji)
J = np.array([[trigsimp(expr) for expr in row] for row in J]).T
```

In [6]:
```
Matrix(J)
```

Out[6]:

$$\left[\begin{array}{cc}
-a_2 \sin(\theta_1(t))\cos(\theta_2(t)) - a_3 \sin(\theta_1(t))\cos(\theta_2(t)+\theta_3(t)) + d_4\cos(\theta_1(t)) + d_5 \sin(\theta_2(t)+\theta_3(t)+\theta_4(t))\sin(\theta_1(t)) + d_6(-\sin(\theta_1(t))\sin(\theta_5(t))\cos(\theta_2(t)+\theta_3(t)+\theta_4(t)) + \cos(\theta_1(t))\cos(\theta_5(t))) & -(a_2\sin(\theta_2(t)) + a_3\sin(\theta_2(t)+\theta_3(t)) + d_5\cos(\theta_2(t)+\theta_3(t) - \\
a_2\cos(\theta_1(t))\cos(\theta_2(t)) + a_3\cos(\theta_2(t)+\theta_3(t))\cos(\theta_1(t)) + d_4\sin(\theta_1(t)) - d_5\sin(\theta_2(t)+\theta_3(t)+\theta_4(t))\cos(\theta_1(t)) + d_6(\sin(\theta_1(t))\cos(\theta_5(t)) + \sin(\theta_5(t))\cos(\theta_2(t)+\theta_3(t)+\theta_4(t))\cos(\theta_1(t))) & -(a_2\sin(\theta_2(t)) + a_3\sin(\theta_2(t)+\theta_3(t)) + d_5\cos(\theta_2(t)+\theta_3(t) - \\
0 & a_2\cos(\theta_2(t)) + a_3\cos(\theta_2(t)+\theta_3(t)) - d_5\sin(\theta_2(t)+\theta_3 \\
0 & \sin(\theta_1 \\
0 & -\cos(\theta \\
1 & 0
\end{array}\right.$$

Part2

Find Singularities

2-1) J singularities

In [7]:
```
J_det = Matrix(J).det()
```

In [8]:
```
J_det_simp = trigsimp(J_det)
```

In [9]:
```
J_det_simp
```

Out[9]: $a_2 a_3 (a_2 \cos(\theta_2(t)) + a_3 \cos(\theta_2(t) + \theta_3(t)) - d_5 \sin(\theta_2(t) + \theta_3(t) + \theta_4(t))) \sin(\theta_3(t)) \sin(\theta_5(t))$

2-2) Jv singularities

In [10]:
```
Jv = J[:3,:]
simple_JvJvT = simplify(Jv@(Jv.T))
```

In [15]:
```
simple_JvJvT.shape
```

Out[15]: (3, 3)

In [17]:
```
Jv_det = Matrix(simple_JvJvT).det()
```

In [21]:
```
np.array(Jv_det)
```

Out[21]:
```
array(-a2**6*sin(theta1(t))**2*sin(theta2(t))**4*cos(theta1(t))**2*cos(theta2(t))**2 - 2*a2**6*sin(theta1(t))**2*sin(theta2(t))**2*cos(theta1(t))**2*cos(the
ta2(t))**4 + a2**6*sin(theta1(t))*sin(2*theta1(t))*sin(theta2(t))**4*cos(theta1(t))*cos(theta2(t))**2 + a2**6*sin(theta1(t))*sin(2*theta1(t))*sin(theta2(t))
**2*cos(theta1(t))*cos(theta2(t))**4 - a2**6*sin(2*theta1(t))**2*sin(theta2(t))**4*cos(theta2(t))**2/4 - 4*a2**5*a3*sin(theta2(t) + theta3(t))*sin(theta1
(t))**2*sin(theta2(t))**3*cos(theta1(t))**2*cos(theta2(t))**2 - 4*a2**5*a3*sin(theta2(t) + theta3(t))*sin(theta1(t))**2*sin(theta2(t))*cos(theta1(t))**2*cos
(theta2(t))**4 + 4*a2**5*a3*sin(theta2(t) + theta3(t))*sin(theta1(t))*sin(2*theta1(t))*sin(theta2(t))**3*cos(theta1(t))*cos(theta2(t))**2 + 2*a2**5*a3*sin(t
heta2(t) + theta3(t))*sin(theta1(t))*sin(2*theta1(t))*sin(theta2(t))*cos(theta1(t))*cos(theta2(t))**4 - a2**5*a3*sin(theta2(t) + theta3(t))*sin(2*theta1(t))
**2*sin(theta2(t))**3*cos(theta2(t))**2 - 2*a2**5*a3*sin(theta1(t))**2*sin(theta2(t))**4*cos(theta2(t) + theta3(t))*cos(theta1(t))**2*cos(theta2(t)) - 8*a2*
*5*a3*sin(theta1(t))**2*sin(theta2(t))**2*cos(theta2(t) + theta3(t))*cos(theta1(t))**2*cos(theta2(t))**3 + 2*a2**5*a3*sin(theta1(t))*sin(2*theta1(t))*sin(th
eta2(t))**4*cos(theta2(t) + theta3(t))*cos(theta1(t))*cos(theta2(t)) + 4*a2**5*a3*sin(theta1(t))*sin(2*theta1(t))*sin(theta2(t))**2*cos(theta2(t) + theta3
(t))*cos(theta1(t))*cos(theta2(t))**3 - a2**5*a3*sin(2*theta1(t))**2*sin(theta2(t))**4*cos(theta2(t) + theta3(t))*cos(theta2(t))/2 - 2*a2**5*d4*sin(theta1
(t))**3*sin(theta2(t))**2*cos(theta1(t))*cos(theta2(t))**3 + a2**5*d4*sin(theta1(t))**2*sin(2*theta1(t))*sin(theta2(t))**2*cos(theta2(t))**3 + 2*a2**5*d4*si
n(theta1(t))*sin(theta2(t))**2*cos(theta1(t))**3*cos(theta2(t))**3 - a2**5*d4*sin(2*theta1(t))*sin(theta2(t))**2*cos(theta1(t))**2*cos(theta2(t))**3 + 2*a2*
*5*d5*sin(theta2(t) + theta3(t) + theta4(t))*sin(theta1(t))**2*sin(theta2(t))**4*cos(theta1(t))**2*cos(theta2(t)) + 8*a2**5*d5*sin(theta2(t) + theta3(t) + t
heta4(t))*sin(theta1(t))**2*sin(theta2(t))**2*cos(theta1(t))**2*cos(theta2(t))**3 - 2*a2**5*d5*sin(theta2(t) + theta3(t) + theta4(t))*sin(theta1(t))*sin(2*t
heta1(t))*sin(theta2(t))**4*cos(theta1(t))*cos(theta2(t)) - 4*a2**5*d5*sin(theta2(t) + theta3(t) + theta4(t))*sin(theta1(t))*sin(2*theta1(t))*sin(theta2(t))
**2*cos(theta1(t))*cos(theta2(t))**3 + a2**5*d5*sin(theta2(t) + theta3(t) + theta4(t))*sin(2*theta1(t))**2*sin(theta2(t))**4*cos(theta2(t))/2 - 4*a2**5*d5*s
in(theta1(t))**2*sin(theta2(t))**3*cos(theta2(t) + theta3(t) + theta4(t))*cos(theta1(t))**2*cos(theta2(t))**2 - 4*a2**5*d5*sin(theta1(t))**2*sin(theta2(t))*
cos(theta2(t) + theta3(t) + theta4(t))*cos(theta1(t))**2*cos(theta2(t))**4 + 4*a2**5*d5*sin(theta1(t))*sin(2*theta1(t))*sin(theta2(t))**3*cos(theta2(t) + th
eta3(t) + theta4(t))*cos(theta1(t))*cos(theta2(t))**2 + 2*a2**5*d5*sin(theta1(t))*sin(2*theta1(t))*sin(theta2(t))*cos(theta2(t) + theta3(t) + theta4(t))*cos
```

In [29]:
```
Jv_det_simp = simplify(Jv_det)
Jv_det_simp
```

Out[29]: *[Math Processing Error]*

2-3) Jw singularities

In [22]:
```
Jw = J[3:,:]
simple_JwJwT = simplify(Jw@(Jw.T))
```

In [23]:
```
simple_JwJwT.shape
```

Out[23]: (3, 3)

In [24]:
```
Jw_det = Matrix(simple_JwJwT).det()
```

In [26]:
```
Jw_det_simp = simplify(Jw_det)
Jw_det_simp
```

Out[26]:

$$3 \sin^4(\theta_2(t) + \theta_3(t) + \theta_4(t)) \sin^4(\theta_1(t)) \sin^2(\theta_5(t)) + 6 \sin^4(\theta_2(t) + \theta_3(t) + \theta_4(t)) \sin^2(\theta_1(t)) \sin^2(\theta_5(t)) \cos^2(\theta_1(t)) + 3 \sin^4(\theta_2(t) + \theta_3(t) + \theta_4(t)) \sin^2(\theta_5(t)) \cos^4(\theta_1(t)) - \sin^2(\theta_2(t) + \theta_3(t) + \theta_4(t)) \sin^4(\theta_1(t)) \sin^4(\theta_5(t)) \cos^2(\theta_2(t) + \theta_3(t) + \theta_4(t)) \cos^2(\theta_5(t)) + 3 \sin^2(\theta_2(t) + \theta_3(t) + \theta_4(t)) \sin^4(\theta_1(t)) \sin^4(\theta_5(t)) \cos^2(\theta_2(t) + \theta_3(t) + \theta_4(t)) - 2 \sin^2(\theta_2(t) + \theta_3(t) + \theta_4(t)) \sin^4(\theta_1(t)) \sin^2(\theta_5(t)) \cos^2(\theta_2(t) + \theta_3(t) + \theta_4(t)) \cos^4(\theta_5(t)) - \sin^2(\theta_2(t) + \theta_3(t) + \theta_4(t)) \sin^4(\theta_1(t)) \cos^2(\theta_2(t) + \theta_3(t) + \theta_4(t)) \cos^6(\theta_5(t)) - 3 \sin^2(\theta_2(t) + \theta_3(t) + \theta_4(t)) \sin^4(\theta_1(t)) \cos^2(\theta_2(t) + \theta_3(t) + \theta_4(t)) \cos^4(\theta_5(t)) + \sin^2(\theta_2(t) + \theta_3(t) + \theta_4(t)) \sin^4(\theta_1(t)) \cos^2(\theta_2(t) + \theta_3(t) + \theta_4(t)) \cos^2(\theta_5(t)) + 3 \sin^2(\theta_2(t) + \theta_3(t) + \theta_4(t)) \sin^4(\theta_1(t)) \cos^2(\theta_2(t) + \theta_3(t) + \theta_4(t)) + \sin^2(\theta_2(t) + \theta_3(t) + \theta_4(t)) \sin^4(\theta_1(t)) \cos^2(\theta_5(t)) + 3 \sin^2(\theta_2(t) + \theta_3(t) + \theta_4(t)) \sin^4(\theta_1(t)) - 2 \sin^2(\theta_2(t) + \theta_3(t) + \theta_4(t)) \sin^2(\theta_1(t)) \sin^4(\theta_5(t)) \cos^2(\theta_2(t) + \theta_3(t) + \theta_4(t)) \cos^2(\theta_1(t)) \cos^2(\theta_5(t)) + 6 \sin^2(\theta_2(t) + \theta_3(t) + \theta_4(t)) \sin^2(\theta_1(t)) \sin^4(\theta_5(t)) \cos^2(\theta_2(t) + \theta_3(t) + \theta_4(t)) \cos^2(\theta_1(t)) - 4 \sin^2(\theta_2(t) + \theta_3(t) + \theta_4(t)) \sin^2(\theta_1(t)) \sin^2(\theta_5(t)) \cos^2(\theta_2(t) + \theta_3(t) + \theta_4(t)) \cos^2(\theta_1(t)) \cos^4(\theta_5(t)) - 2 \sin^2(\theta_2(t) + \theta_3(t) + \theta_4(t)) \sin^2(\theta_1(t)) \cos^2(\theta_2(t) + \theta_3(t) + \theta_4(t)) \cos^2(\theta_1(t)) \cos^6(\theta_5(t)) - 6 \sin^2(\theta_2(t) + \theta_3(t) + \theta_4(t)) \sin^2(\theta_1(t)) \cos^2(\theta_2(t) + \theta_3(t) + \theta_4(t)) \cos^2(\theta_1(t)) \cos^4(\theta_5(t)) + 2 \sin^2(\theta_2(t) + \theta_3(t) + \theta_4(t)) \sin^2(\theta_1(t)) \cos^2(\theta_2(t) + \theta_3(t) + \theta_4(t)) \cos^2(\theta_1(t)) \cos^2(\theta_5(t)) + 6 \sin^2(\theta_2(t) + \theta_3(t) + \theta_4(t)) \sin^2(\theta_1(t)) \cos^2(\theta_2(t) + \theta_3(t) + \theta_4(t)) \cos^2(\theta_1(t)) + 2 \sin^2(\theta_2(t) + \theta_3(t) + \theta_4(t)) \sin^2(\theta_1(t)) \cos^2(\theta_1(t)) \cos^2(\theta_5(t)) + 6 \sin^2(\theta_2(t) + \theta_3(t) + \theta_4(t)) \sin^2(\theta_1(t)) \cos^2(\theta_1(t)) - \sin^2(\theta_2(t) + \theta_3(t) + \theta_4(t)) \sin^4(\theta_5(t)) \cos^2(\theta_2(t) + \theta_3(t) + \theta_4(t)) \cos^4(\theta_1(t)) \cos^2(\theta_5(t)) + 3 \sin^2(\theta_2(t) + \theta_3(t) + \theta_4(t)) \sin^4(\theta_5(t)) \cos^2(\theta_2(t) + \theta_3(t) + \theta_4(t)) \cos^4(\theta_1(t)) - 2 \sin^2(\theta_2(t) + \theta_3(t) + \theta_4(t)) \sin^2(\theta_5(t)) \cos^2(\theta_2(t) + \theta_3(t) + \theta_4(t)) \cos^4(\theta_1(t)) \cos^4(\theta_5(t)) - \sin^2(\theta_2(t) + \theta_3(t) + \theta_4(t)) \cos^2(\theta_2(t) + \theta_3(t) + \theta_4(t)) \cos^4(\theta_1(t)) \cos^6(\theta_5(t)) - 3 \sin^2(\theta_2(t) + \theta_3(t) + \theta_4(t)) \cos^2(\theta_2(t) + \theta_3(t) + \theta_4(t)) \cos^4(\theta_1(t)) \cos^4(\theta_5(t)) + \sin^2(\theta_2(t) + \theta_3(t) + \theta_4(t)) \cos^2(\theta_2(t) + \theta_3(t) + \theta_4(t)) \cos^4(\theta_1(t)) \cos^2(\theta_5(t)) + 3 \sin^2(\theta_2(t) + \theta_3(t) + \theta_4(t)) \cos^2(\theta_2(t) + \theta_3(t) + \theta_4(t)) \cos^4(\theta_1(t)) + \sin^2(\theta_2(t) + \theta_3(t) + \theta_4(t)) \cos^4(\theta_1(t)) \cos^2(\theta_5(t)) + 3 \sin^2(\theta_2(t) + \theta_3(t) + \theta_4(t)) \cos^4(\theta_1(t)) + 3 \sin^4(\theta_1(t)) \sin^2(\theta_5(t)) \cos^4(\theta_2(t) + \theta_3(t) + \theta_4(t)) + 3 \sin^4(\theta_1(t)) \sin^2(\theta_5(t)) \cos^2(\theta_2(t) + \theta_3(t) + \theta_4(t)) + 6 \sin^2(\theta_1(t)) \sin^2(\theta_5(t)) \cos^4(\theta_2(t) + \theta_3(t) + \theta_4(t)) \cos^2(\theta_1(t)) + 6 \sin^2(\theta_1(t)) \sin^2(\theta_5(t)) \cos^2(\theta_2(t) + \theta_3(t) + \theta_4(t)) \cos^2(\theta_1(t)) + 3 \sin^2(\theta_5(t)) \cos^4(\theta_2(t) + \theta_3(t) + \theta_4(t)) \cos^4(\theta_1(t)) + 3 \sin^2(\theta_5(t)) \cos^2(\theta_2(t) + \theta_3(t) + \theta_4(t)) \cos^4(\theta_1(t))$$

Part3

Dynamics Equations (Newton-Euler method)

3-1) Define suitable variables

- We have Rotation matrixes from forward kinematics. Here we need both R and R transpose:

In [7]:
```
DOF = 6
# Some variables which we had them from Phase1 and robot structure
T_num = T
T_num = [Matrix(T_num[i]).subs({'d1':0.128, 'd4': 0.16389, 'd5':0.1157, 'd6':0.0922, 'a2':0.6129, 'a3':0.5716}) for i in range(len(T_num))]
R = [np.array(T_num[i])[:3,:3] for i in range(DOF)]
R_trans = [np.array(T_num[i])[:3,:3].T for i in range(DOF)]
# Also we add R_n_e because we need it in inward iterations (Note: here We assume we dont have end effector
# and here we just set R_6_7 for preventing from error in code)
R_6_7 = np.array([[1, 0, 0],[0, 1, 0],[0, 0, 1]])
R.append(R_6_7)
R_trans.append(R_6_7)
```

- We need Inertia tensors of the links. We find them from Solidworks file.

In [8]:
```
# Units : kg.m^2
I_c0_0 = np.array([[0, 0, 0],[0, 0, 0],[0, 0, 0]])
I_c1_1 = np.array([[0.03, 0, 0],[0, 0.03, 0],[0, 0, 0.03]])
I_c2_2 = np.array([[0.05, 0, 0.01],[0, 1.23, 0],[0.01, 0, 1.23]])
I_c3_3 = np.array([[0.02, 0, 0],[0, 0.54, 0],[0, 0, 0.54]])
I_c4_4 = np.array([[0.003, 0, 0],[0, 0.0024, 0.00025],[0, 0.00025, 0.0028]])
I_c5_5 = np.array([[0.003, 0, 0],[0, 0.0024, -0.00025],[0, -0.00025, 0.0028]])
I_c6_6 = np.array([[0.00022, 0, 0],[0, 0.00024, 0],[0, 0, 0.0004]])
I = [I_c0_0, I_c1_1, I_c2_2, I_c3_3, I_c4_4, I_c5_5, I_c6_6]
```

- We need mass of each link. We find them from Solidworks file.

In [9]:
```
# Units : kg
m0, m1, m2, m3, m4, m5, m6 = 0, 8.3, 23.52, 12.56, 1.967, 1.967, 0.43
m = [m0, m1, m2, m3, m4, m5, m6]
```

- We need the distance of center of mass of link i from frame i. (P_i_ci)

In [10]:
```
P_0_c0 = np.array([[0],[0],[0]])
P_1_c1 = np.array([[0],[-0.01],[-0.01]])*10**(-3)
P_2_c2 = np.array([[0.25],[0],[0.17]])*10**(-3)
P_3_c3 = np.array([[0.26],[0],[0.05]])*10**(-3)
P_4_c4 = np.array([[0],[9.74],[-7.6]])*10**(-3)
P_5_c5 = np.array([[0],[-9.74],[-7.6]])*10**(-3)
P_6_c6 = np.array([[0],[-0.95],[-17.46]])*10**(-3)
Pc = [P_0_c0, P_1_c1, P_2_c2, P_3_c3, P_4_c4, P_5_c5, P_6_c6]
```

- We need the distance between frame i+1 and i. We can find them from fourth column of forward kinematics transformation matrix

In [11]:
```
p_0_1 = np.array(T_num[0][:3,3])
p_1_2 = np.array(T_num[1][:3,3])
p_2_3 = np.array(T_num[2][:3,3])
p_3_4 = np.array(T_num[3][:3,3])
p_4_5 = np.array(T_num[4][:3,3])
p_5_6 = np.array(T_num[5][:3,3])
# we define p_6_7 only for preventing from Error in inward iteration step
p_6_7 = np.array([[0],[0],[0]])
P = [p_0_1, p_1_2, p_2_3, p_3_4, p_4_5, p_5_6, p_6_7]
```

- We need some parametric varibales like theta_dot and theta_dotdot

In [12]:
```
theta_d = [i.diff(t) for i in theta]
theta_dd = [i.diff(t) for i in theta_d]
```

Out[12]:
```
'\nnum_symbols = 6  # Number of symbols to generate\ntheta_d_names = [f"theta_d{i}" for i in range(num_symbols+1)]\ntheta_d = symbols(theta_d_names)\ntheta_dd
_names = [f"theta_dd{i}" for i in range(num_symbols+1)]\ntheta_dd = symbols(theta_dd_names)'
```

- Some lists are needed and we should use them in Outward iterations.

In [13]:
```
w = [np.array([[0],[0],[0]])]
w_dot = [np.array([[0],[0],[0]])]
g = symbols('g')
v_dot = [np.array([[0],[0],[g]])]
vc_dot = []
F = [np.array([[0],[0],[0]])]
N = [np.array([[0],[0],[0]])]
```

- Some lists are needed and we should use them in Inward iterations

In [14]:
```
f = [np.array([[0],[0],[0]])]
n = [np.array([[0],[0],[0]])]
taw = []
```

3-2) Outward iterations

In [15]:
```
for i in range(DOF):
    #print(i)
    w_i1 = R_trans[i]@w[i] + np.array([[0],[0],[theta_d[i+1]]])
    #w_i1 = simplify(w_i1)
    w_dot_i1 = R_trans[i]@w_dot[i] + np.cross(R_trans[i]@w[i], np.array([[0],[0],[theta_d[i+1]]]),axis=0) + np.array([[0],[0],[theta_dd[i+1]]])
    #w_dot_i1 = simplify(w_dot_i1)
    v_dot_i1 = R_trans[i]@(np.cross(w_dot[i], P[i],axis=0)+np.cross(w[i],np.cross(w[i],P[i], axis=0), axis=0)+v_dot[i])
    #v_dot_i1 = simplify(v_dot_i1)
    vc_dot_i1 = np.cross(w_dot_i1, Pc[i+1], axis=0 )+np.cross(w_i1,np.cross(w_i1,Pc[i+1], axis=0), axis=0)+v_dot_i1
    F_i1 = m[i+1]*vc_dot_i1
    N_i1 = I[i+1]@w_dot_i1 + np.cross(w_i1, I[i+1]@w_i1, axis=0)

    w.append(w_i1)
    w_dot.append(w_dot_i1)
    v_dot.append(v_dot_i1)
    vc_dot.append(vc_dot_i1)
    F.append(F_i1)
    N.append(N_i1)
```

3-3) Inward iterations

In [16]:
```
for i in range(DOF,0,-1):
    fi = R[i]@f[len(f)-1] + F[i]
    ni = N[i] + R[i]@n[len(n)-1]+np.cross(Pc[i],F[i], axis=0)+np.cross(P[i], R[i]@f[len(f)-1], axis=0)
    tawi = ni[2]
    f.append(fi)
    n.append(ni)
    taw.append(tawi)
f.reverse()
n.reverse()
taw.reverse()
```

In [17]:
```
tau_copy = taw
replacements = []
for i in range(1,len(theta)):
    replacements.append((theta[i].diff(t).diff(t), Symbol(f'ddth{i}')))
    replacements.append((theta[i].diff(t), Symbol(f'dth{i}')))
    replacements.append((theta[i], Symbol(f'th{i}')))
tau_replace = []
for eq in tau_copy:
    tau_replace.append(eq[0].subs(replacements))
```

In [18]:
```
num_symbols = 6  # Number of symbols to generate
th_names = [f"th{i}" for i in range(num_symbols+1)]
th = symbols(th_names)
th_d_names = [f"dth{i}" for i in range(num_symbols+1)]
th_d = symbols(th_d_names)
th_dd_names = [f"ddth{i}" for i in range(num_symbols+1)]
th_dd = symbols(th_dd_names)
```

In [25]:
```python
M = []
for tau_i in tau_replace:
    m = []
    for i in range(1,len(th_dd)):
        m.append(tau_i.diff(th_dd[i]))
    M.append(m)
```

In [26]:
```python
G = []
for tau_i in tau_replace:
    G.append(tau_i.diff(g))
```

In [27]:
```python
tau_replace_cop3 = tau_replace
V = Matrix(tau_replace_cop3).subs({'ddth1':0, 'ddth2':0, 'ddth3':0,'ddth4':0,
                                   'ddth5':0,'ddth6':0, 'g':0})
```

check matrices with some values

In [32]:
```python
init_th = [0, 0, 0, 0, 0, 0]
init_dth = [1, 0, 0, 0, 0, 0]
init_ddth = [0, 0, 0, 0, 0, 0]
M_copy = M
M_num = Matrix(M_copy).subs({'th1':init_th[0], 'th2':init_th[1], 'th3':init_th[2], 'th4':init_th[3], 'th5':init_th[4],
                              'th6':init_th[5],'dth1':init_dth[0], 'dth2':init_dth[1], 'dth3':init_dth[2], 'dth4':init_dth[3],
                              'dth5':init_dth[4],'dth6':init_dth[5]})
V_copy = V
V_num = V_copy.subs({'th1':init_th[0], 'th2':init_th[1], 'th3':init_th[2], 'th4':init_th[3], 'th5':init_th[4],
                    'th6':init_th[5],'dth1':init_dth[0], 'dth2':init_dth[1], 'dth3':init_dth[2], 'dth4':init_dth[3],
                    'dth5':init_dth[4],'dth6':init_dth[5]})
G_copy = G
G_num = 9.81*Matrix(G_copy).subs({'th1':init_th[0], 'th2':init_th[1], 'th3':init_th[2], 'th4':init_th[3], 'th5':init_th[4],
                              'th6':init_th[5],'dth1':init_dth[0], 'dth2':init_dth[1], 'dth3':init_dth[2], 'dth4':init_dth[3],
                              'dth5':init_dth[4],'dth6':init_dth[5], 'g':9.81})
```

In [33]:
```python
M_num
```

Out[33]:

$$\begin{bmatrix} 12.7822265545108 & -0.0511883109442 & -0.0511883109442 & -0.0511883109442 & 0.0140356429114 & 9.7480355 \cdot 10^{-5} \\ -0.0511883109442 & 12.6494261034502 & 3.5311272994102 & 0.0344342513142 & -0.005508900948 & 0.000353124625 \\ -0.0511883109442 & 3.5311272994102 & 2.0002698082102 & 0.0344342513142 & -0.005508900948 & 0.000353124625 \\ -0.0511883109442 & 0.0344342513142 & 0.0344342513142 & 0.0344342513142 & -0.005508900948 & 0.000353124625 \\ 0.0140356429114 & -0.005508900948 & -0.005508900948 & -0.005508900948 & 0.0056286136372 & 3.053129 \cdot 10^{-5} \\ 9.7480355 \cdot 10^{-5} & 0.000353124625 & 0.000353124625 & 0.000353124625 & 3.053129 \cdot 10^{-5} & 0.000400388075 \end{bmatrix}$$

In [34]:
```python
V_num
```

Out[34]:

$$\begin{bmatrix} 1.94289029309402 \cdot 10^{-16} \\ 0.33300296241 \\ 0.33300296241 \\ 0.33300296241 \\ -0.06076103591 \\ -0.00048386825 \end{bmatrix}$$

In [35]:
```python
G_num
```

Out[35]:

$$\begin{bmatrix} 0 \\ 126.316773756 \\ 24.50271168 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

Solving dynamics equations

In [100]:
```python
from scipy.integrate import odeint
```

In [110]:
```python
def model(z,t):
    M_copy = M
    M_num = Matrix(M_copy).subs({'th1':z[0], 'th2':z[2], 'th3':z[4], 'th4':z[6], 'th5':z[8],'th6':z[10],
                                 'dth1':z[1], 'dth2':z[3], 'dth3':z[5], 'dth4':z[7],'dth5':z[9],'dth6':z[11]})
    V_copy = V
    V_num = Matrix(V_copy).subs({'th1':z[0], 'th2':z[2], 'th3':z[4], 'th4':z[6], 'th5':z[8],'th6':z[10],
                                 'dth1':z[1], 'dth2':z[3], 'dth3':z[5], 'dth4':z[7],'dth5':z[9],'dth6':z[11]})
    G_copy = G
    G_num = Matrix(G_copy).subs({'th1':z[0], 'th2':z[2], 'th3':z[4], 'th4':z[6], 'th5':z[8],'th6':z[10],
                                 'dth1':z[1], 'dth2':z[3], 'dth3':z[5], 'dth4':z[7],'dth5':z[9],'dth6':z[11],
                                 'g':9.81})
    M_num_inv = M_num.inv()
    theta_dd = np.array(M_num_inv)@(np.array(tau_num_input) - np.array(V_num) - np.array(G_num))
    return [z[1], theta_dd[0][0], z[3], theta_dd[1][0], z[5], theta_dd[2][0], z[7], theta_dd[3][0], z[9],
            theta_dd[4][0], z[11], theta_dd[5][0]]
```

In [ ]:
```python
tau_num_input = Matrix([0, 1, 1, 1, 1, 1])
theta_init = [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
t = np.linspace(0,1,5)
theta_solved = odeint(model, theta_init, t)
```