ENPM 662
HOMEWORK 5

Problem : KUKA WIIA dynamics

Consider a KUKA WIIA robot with a pen (L=10 cm) attached as the end effector of the robot along Z direction of the local frame (Figure 1). Assume that joint 3 is locked and will not be able to move so the Jacobian matrix is square matrix.

Assuming the robot motion is quasi-static (´q≅0∧´q≅0¿ , calculate joint torques that is required to compensate the robot weight and ensue that pen is pushed against the wall with 5 N while drawing the circle (Figure 2).

Find mass information from KUKA WIIA datasheet.

A. Step 1- Python code that parametrically calculates matrix g(q)

In [19]:
```
from sympy import Matrix, symbols, cos, sin, simplify, pi, pprint, diff
from numpy import linspace, matrix
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
import time
import numpy as np
import sympy as sp
```

In [20]:
```
#Defining the degrees of freedom:
DOF = 7
```

In [21]:
```
# Defining distances between the x axes of the joints:
d1 = 360
d3 = 420
d5 = 399.5
d7 = 115.5
```

In [22]:
```
#Defining the link parameters theta for x axes of all the joints:
theta1, theta2, theta4, theta5, theta6, theta7 = symbols('theta1, theta2, theta4, theta5, theta6, theta7')
```

In [23]:
```
#Calculating the transformation matrices between all the joints:
T01 = Matrix([[cos(theta1),0,-sin(theta1),0], [sin(theta1),0,cos(theta1),0], [0,-1,0,d1], [0,0,0,1]])

T12 = Matrix([[cos(theta2),0,sin(theta2),0], [sin(theta2),0,-cos(theta2),0], [0,1,0,0], [0,0,0,1]])

T23 = Matrix([[cos(0),0,-sin(0),0], [sin(0),0,-cos(0),0], [0,1,0,d3], [0,0,0,1]])

T34 = Matrix([[cos(theta4),0,-sin(theta4),0], [sin(theta4),0,cos(theta4),0], [0,-1,0,0], [0,0,0,1]])

T45 = Matrix([[cos(theta5),0,-sin(theta5),0], [sin(theta5),0,cos(theta5),0], [0,-1,0,d5], [0,0,0,1]])

T56 = Matrix([[cos(theta6),0,sin(theta6),0], [sin(theta6),0,-cos(theta6),0], [0,1,0,0], [0,0,0,1]])

T67 = Matrix([[cos(theta7),-sin(theta7),0,0], [sin(theta7),cos(theta7),0,0], [0,0,1,d7], [0,0,0,1]])
```

In [24]:
```
#Calculating transformation matrices from base frame to joint frames(excluding the locked joint 3):
T02 = T01 * T12
T04 = T02 * T23 * T34
T05 = T04 * T45
T06 = T05 * T56
T07 = T06 * T67
```

In [25]:
```
#Extracting corresponding Z vectors from the transformation matrices:
Z0 = Matrix([0, 0, 1])
Z1 = T01[0:3,2]
Z2 = T02[0:3,2]
Z4 = T04[0:3,2]
Z5 = T05[0:3,2]
Z6 = T06[0:3,2]
Z7 = T07[0:3,2]
```

In [26]:
```
#Extracting the position vector from tansformation matrix of base frame to end frame:
pv = T07[0:3, 3]
```

In [27]:
```
#Calculating Jacobian matrix using partial derivative of position vector:
pv = T07[0:3,3]
J1 = diff(pv,theta1)
J2 = diff(pv,theta2)
J3 = diff(pv,theta4)
J4 = diff(pv,theta5)
J5 = diff(pv,theta6)
J6 = diff(pv,theta7)
```

In [28]:
```
#Calculating the Z matrix using corresponding Z vectors and hence using it for Jacobian:
j = Matrix().col_insert(0,J1).col_insert(1,J2).col_insert(2,J3).col_insert(3,J4).col_insert(4,J5).col_insert(5,J6)
Z = Matrix().col_insert(0,Z0).col_insert(1,Z1).col_insert(2,Z2).col_insert(3,Z4).col_insert(4,Z5).col_insert(5,Z6)
J = Matrix().row_insert(0,j).row_insert(3,Z)
```

In [29]:
```
#Defining necessary variables required for dynamical equations:
m1,m2,m3,m4,m5,m6,m7,P = symbols('m1 m2 m3 m4 m5 m6 m7 P')
a,b,d,e,f,g=symbols('a b d e f g')
L1,L2,L3,L4,L5,L6=symbols('L1 L2 L3 L4 L5 L6')
theta = Matrix([float(pi/2), 0.0, float(-pi/2), 0.0, 0.00001, 0.0])
N = 200
j_angle = linspace(float(pi/2), float((5*pi)/2), num=N)
Force = Matrix([0,5,0,0,0,0])
q_store = sp.zeros(N,6)
```

In [30]:
```
# Calculating the Potential Enegry
PE = m1*L1/2 + m2*(L1 + L2*cos(b)/2) + m3*(L1 + (L2 + L3/2)*cos(b)) + m4*(L1 + L4*(-sin(b)*sin(d) + cos(b)*cos(d))/2
```

In [31]:
```
#Calculating Euler-Lagrange by differentiating potential with respect to all q_k:
EL1 = diff(PE,a)
EL2 = diff(PE,b)
EL4 = diff(PE,d)
EL5 = diff(PE,e)
EL6 = diff(PE,f)
EL7 = diff(PE,g)
Q_k = Matrix([EL1,EL2,EL4,EL5,EL6,EL7])
print("Matrix g(q) =>\n:", str(Q_k))
```

```
Matrix g(q) =>
: Matrix([[0], [-L2*m2*sin(b)/2 - m3*(L2 + L3/2)*sin(b) + m4*(L4*(-sin(b)*cos(d) - sin(d)*cos(b))/2 - (L2 + L3)*sin
(b)) + m5*(-(L2 + L3)*sin(b) + (L4 + L5/2)*(-sin(b)*cos(d) - sin(d)*cos(b))) + m6*(L6*((sin(b)*sin(d) - cos(b)*cos
(d))*sin(f)*cos(e) + (-sin(b)*cos(d) - sin(d)*cos(b))*cos(f))/2 - (L2 + L3)*sin(b) + (L4 + L5)*(-sin(b)*cos(d) - si
n(d)*cos(b))) + m7*(-(L2 + L3)*sin(b) + (L4 + L5)*(-sin(b)*cos(d) - sin(d)*cos(b)) + (L6 + P/2)*((sin(b)*sin(d) - c
os(b)*cos(d))*sin(f)*cos(e) + (-sin(b)*cos(d) - sin(d)*cos(b))*cos(f)))], [L4*m4*(-sin(b)*cos(d) - sin(d)*cos(b))/2
+ m5*(L4 + L5/2)*(-sin(b)*cos(d) - sin(d)*cos(b)) + m6*(L6*((sin(b)*sin(d) - cos(b)*cos(d))*sin(f)*cos(e) + (-sin
(b)*cos(d) - sin(d)*cos(b))*cos(f))/2 + (L4 + L5)*(-sin(b)*cos(d) - sin(d)*cos(b))) + m7*((L4 + L5)*(-sin(b)*cos(d)
- sin(d)*cos(b)) + (L6 + P/2)*((sin(b)*sin(d) - cos(b)*cos(d))*sin(f)*cos(e) + (-sin(b)*cos(d) - sin(d)*cos(b))*cos
(f)))], [-L6*m6*(-sin(b)*cos(d) - sin(d)*cos(b))*sin(e)*sin(f)/2 - m7*(L6 + P/2)*(-sin(b)*cos(d) - sin(d)*cos(b))*s
in(e)*sin(f)], [L6*m6*(-(-sin(b)*sin(d) + cos(b)*cos(d))*sin(f) + (-sin(b)*cos(d) - sin(d)*cos(b))*cos(e)*cos(f))/2
+ m7*(L6 + P/2)*(-(-sin(b)*sin(d) + cos(b)*cos(d))*sin(f) + (-sin(b)*cos(d) - sin(d)*cos(b))*cos(e)*cos(f))], [0]])
```

B. Step 2- Python code that parametrically calculates total joint torque (gravity + external force)

In [32]:
```
#Calculating the torque
T_plot = sp.zeros(6,200)
Torque = J.T*Force + Q_k
print(" total joint torque (gravity + external force)  =>\n:", str(Torque))
```

```
 total joint torque (gravity + external force)  =>
: Matrix([[5*(115.5*(sin(theta2)*sin(theta4)*cos(theta1) + cos(theta1)*cos(theta2)*cos(theta4))*cos(theta5) - 115.5
*sin(theta1)*sin(theta5))*sin(theta6) + 5*(115.5*sin(theta2)*cos(theta1)*cos(theta4) - 115.5*sin(theta4)*cos(theta
1)*cos(theta2))*cos(theta6) + 1997.5*sin(theta2)*cos(theta1)*cos(theta4) + 2100*sin(theta2)*cos(theta1) - 1997.5*si
n(theta4)*cos(theta1)*cos(theta2)], [-L2*m2*sin(b)/2 - m3*(L2 + L3/2)*sin(b) + m4*(L4*(-sin(b)*cos(d) - sin(d)*cos
(b))/2 - (L2 + L3)*sin(b)) + m5*(-(L2 + L3)*sin(b) + (L4 + L5/2)*(-sin(b)*cos(d) - sin(d)*cos(b))) + m6*(L6*((sin
(b)*sin(d) - cos(b)*cos(d))*sin(f)*cos(e) + (-sin(b)*cos(d) - sin(d)*cos(b))*cos(f))/2 - (L2 + L3)*sin(b) + (L4 + L
5)*(-sin(b)*cos(d) - sin(d)*cos(b))) + m7*(-(L2 + L3)*sin(b) + (L4 + L5)*(-sin(b)*cos(d) - sin(d)*cos(b)) + (L6 +
P/2)*((sin(b)*sin(d) - cos(b)*cos(d))*sin(f)*cos(e) + (-sin(b)*cos(d) - sin(d)*cos(b))*cos(f))) + 5*(115.5*sin(thet
a1)*sin(theta2)*sin(theta4) + 115.5*sin(theta1)*cos(theta2)*cos(theta4))*cos(theta6) + 577.5*(-sin(theta1)*sin(thet
a2)*cos(theta4) + sin(theta1)*sin(theta4)*cos(theta2))*sin(theta6)*cos(theta5) + 1997.5*sin(theta1)*sin(theta2)*sin
(theta4) + 1997.5*sin(theta1)*cos(theta2)*cos(theta4) + 2100*sin(theta1)*cos(theta2)], [L4*m4*(-sin(b)*cos(d) - sin
(d)*cos(b))/2 + m5*(L4 + L5/2)*(-sin(b)*cos(d) - sin(d)*cos(b)) + m6*(L6*((sin(b)*sin(d) - cos(b)*cos(d))*sin(f)*co
s(e) + (-sin(b)*cos(d) - sin(d)*cos(b))*cos(f))/2 + (L4 + L5)*(-sin(b)*cos(d) - sin(d)*cos(b))) + m7*((L4 + L5)*(-s
in(b)*cos(d) - sin(d)*cos(b)) + (L6 + P/2)*((sin(b)*sin(d) - cos(b)*cos(d))*sin(f)*cos(e) + (-sin(b)*cos(d) - sin
(d)*cos(b))*cos(f))) + 5*(-115.5*sin(theta1)*sin(theta2)*sin(theta4) - 115.5*sin(theta1)*cos(theta2)*cos(theta4))*c
os(theta6) + 577.5*(sin(theta1)*sin(theta2)*cos(theta4) - sin(theta1)*sin(theta4)*cos(theta2))*sin(theta6)*cos(thet
a5) - 1997.5*sin(theta1)*sin(theta2)*sin(theta4) - 1997.5*sin(theta1)*cos(theta2)*cos(theta4)], [-L6*m6*(-sin(b)*co
s(d) - sin(d)*cos(b))*sin(e)*sin(f)/2 - m7*(L6 + P/2)*(-sin(b)*cos(d) - sin(d)*cos(b))*sin(e)*sin(f) + 5*(-115.5*(s
in(theta1)*sin(theta2)*sin(theta4) + sin(theta1)*cos(theta2)*cos(theta4))*sin(theta5) + 115.5*cos(theta1)*cos(theta
5))*sin(theta6)], [L6*m6*(-(-sin(b)*sin(d) + cos(b)*cos(d))*sin(f) + (-sin(b)*cos(d) - sin(d)*cos(b))*cos(e)*cos
(f))/2 + m7*(L6 + P/2)*(-(-sin(b)*sin(d) + cos(b)*cos(d))*sin(f) + (-sin(b)*cos(d) - sin(d)*cos(b))*cos(e)*cos(f))
+ 5*(115.5*(sin(theta1)*sin(theta2)*sin(theta4) + sin(theta1)*cos(theta2)*cos(theta4))*cos(theta5) + 115.5*sin(thet
a5)*cos(theta1))*cos(theta6) - 5*(115.5*sin(theta1)*sin(theta2)*cos(theta4) - 115.5*sin(theta1)*sin(theta4)*cos(the
ta2))*sin(theta6)], [0]])
```

In [33]:
```
#Calculating the inverse Jacobian:
for i in range(0,N):
    x = -100.0 * (2*pi/5) * sin(j_angle[i])
    z = 100.0 * (2*pi/5) * cos(j_angle[i])
    p = Matrix([x, 0.0, z, 0.0, 0.0, 0.0])
    J_inv = J.evalf(3,subs={theta1: theta[0],theta2: theta[1], theta4: theta[2], theta5: theta[3], theta6: theta[4],
    t_dot = J_inv * p
```

In [34]:
```
#Plotting joint torques required over time:
theta = theta + (t_dot * (200/N))
q_store[i] = theta
for i in range(0,200):
    T_plot[0,i] = Torque[0].subs([(a,q_store[i,0]),(b,q_store[i,1]),(d,q_store[i,2]),(e,q_store[i,3]),(f,q_store[i,4
    T_plot[1,i] = Torque[1].subs([(a,q_store[i,0]),(b,q_store[i,1]),(d,q_store[i,2]),(e,q_store[i,3]),(f,q_store[i,4
    T_plot[2,i] = Torque[2].subs([(a,q_store[i,0]),(b,q_store[i,1]),(d,q_store[i,2]),(e,q_store[i,3]),(f,q_store[i,4
    T_plot[3,i] = Torque[3].subs([(a,q_store[i,0]),(b,q_store[i,1]),(d,q_store[i,2]),(e,q_store[i,3]),(f,q_store[i,4
    T_plot[4,i] = Torque[4].subs([(a,q_store[i,0]),(b,q_store[i,1]),(d,q_store[i,2]),(e,q_store[i,3]),(f,q_store[i,4
    T_plot[5,i] = Torque[5].subs([(a,q_store[i,0]),(b,q_store[i,1]),(d,q_store[i,2]),(e,q_store[i,3]),(f,q_store[i,4
```

C. Step 3- If robot draws the circle in 200 seconds, plot the joint torques required over time (between t=0 and t=200 s). (Plot 6 graphs. One of each joint: 1,2, 4, 5, 6, and 7)

In [18]:
```
T_plot= np.array(T_plot)
plt.xlabel('time(sec)')
plt.ylabel('Torque(N-mm)')
J = np.linspace(1, 200, num=200)
plt.subplot(3,2,1)
plt.plot(J, T_plot[0],linewidth=1, markersize=12)
plt.subplot(3,2,2)
plt.plot(J, T_plot[1],linewidth=1, markersize=12)
plt.subplot(3,2,3)
plt.plot(J, T_plot[2],linewidth=1, markersize=12)
plt.subplot(3,2,4)
plt.plot(J, T_plot[3],linewidth=1, markersize=12)
plt.subplot(3,2,5)
plt.plot(J, T_plot[4],linewidth=1, markersize=12)
plt.subplot(3,2,6)
plt.plot(J, T_plot[5],linewidth=1, markersize=12)
plt.show()
```