ME 639

INTRODUCTION TO ROBOTICS

Assignment 3 & 4 (Part – 2)

Submitted by:

Rhitosparsha Baishya
23310039
PhD (Mechanical Engineering)

CONTENTS

Question 11

Source Code: (Link to GitHub)

```
import sympy as sp

# Define symbols q, qd and tau for joint variables, joint velocities, and
joint torques respectively
n = int(input("Enter the number of joints: "))
q = sp.Matrix(sp.symbols('q:{}'.format(n)))
qd = sp.Matrix(sp.symbols('qd:{}'.format(n)))
tau = sp.Matrix(sp.symbols('tau:{}'.format(n)))

# Define D and V matrices
D = sp.Matrix([[0] * n for _ in range(n)])
V = sp.Matrix([0] * n)

# Take input values for the D(q) and V(q) matrices
print("\nEnter the values for the D(q) matrix : ")
for i in range(n):
    for j in range(n):
        D[i, j] = sp.simplify(sp.sympify(input(f"Enter the D({i},{j})(q)
expression: ")))

print("\nEnter the values for the V(q) matrix : ")
for i in range(n):
    V[i] = sp.simplify(sp.sympify(input(f"Enter the V({i})(q) expression: ")))

# Compute the Lagrangian L = 0.5*qd^T * D(q) * qd - V(q)^T * q
L = 0.5 * qd.dot(D * qd) - V.dot(q)

# Compute the equations of motion for the robot
eqns = [sp.Eq(sp.diff(L, qd[i]) - sp.diff(L, q[i]), tau[i]) for i in range(n)]

# Simplify the equations
eqns = [sp.simplify(eqn) for eqn in eqns]

# Print the matrices
print("\nD(q) matrix:")
print(D)
print("\nV(q) matrix:")
print(V)

# Print the equations
print("\nEquations of motion for the robot:")
for eqn in eqns:
    print(eqn)
```

Output:

```
Enter the number of joints: 2

Enter the values for the D(q) matrix :
Enter the D(0,0)(q) expression: m1 * l1**2 / 4 + m2 * l1**2 + I1
Enter the D(0,1)(q) expression: m2 * l1 * l2/4 * cos(q2 - q1)
Enter the D(1,0)(q) expression: m2 * l1 * l2/2 * cos(q2 - q1)
Enter the D(1,1)(q) expression: m2 * l2**2 / 4 + I2

Enter the values for the V(q) matrix :
Enter the V(0)(q) expression: m1 * g * l1/2 * sin(q1)
Enter the V(1)(q) expression: m2 * g * (l1 * sin(q1) + l2/2 * sin(q2))

D(q) matrix:
Matrix([[I1 + l1**2*m1/4 + l1**2*m2, l1*l2*m2*cos(q1 - q2)/4], [l1*l2*m2*cos(q1 - q2)/2, I2 + l2**2*m2/4]])

V(q) matrix:
Matrix([[g*l1*m1*sin(q1)/2], [g*m2*(2*l1*sin(q1) + l2*sin(q2))/2]])

Equations of motion for the robot:
Eq(tau0, g*l1*m1*sin(q1)/2 + 0.375*l1*l2*m2*qd1*cos(q1 - q2) + 0.25*qd0*(4*I1 + l1**2*m1 + 4*l1**2*m2))
Eq(tau1, g*l1*m1*q0*cos(q1)/2 + g*l1*m2*q1*cos(q1) + g*m2*(2*l1*sin(q1) + l2*sin(q2))/2 + 0.375*l1*l2*m2*qd0*qd1*sin(q1 - q2)
 + 0.375*l1*l2*m2*qd0*cos(q1 - q2) + 0.25*qd1*(4*I2 + l2**2*m2))
```

Question 12

Source Code: ([Link to GitHub]())

```
import numpy as np

print("Inverse kinematics of Stanford manipulator :\n")

# Take input for link lengths
l1 = float(input("Enter length of Link 1 (m): "))
l2 = float(input("Enter length of Link 2 (m): "))
l3 = float(input("Enter length of Link 3 (m): "))

# Take input for end-effector coordinates
xc = float(input("Enter x-coordinate of end-effector (m): "))
yc = float(input("Enter y-coordinate of end-effector (m): "))
zc = float(input("Enter z-coordinate of end-effector (m): "))

# Compute inverse kinematics solutions
r = np.sqrt(xc**2 + yc**2)
s = zc - l1
theta1_a = np.arctan2(xc,yc)
theta1_b = theta1_a + np.pi
theta2 = np.arctan2(r,s) + np.pi/2
d3 = np.sqrt(r**2 + s**2)

# Output the computed values
if d3 > l3:
    print("Point lies outside the workspace of the robot!")
else:
    print("\nInverse kinematics solutions :\n")
    print("Angle of link 1 ", np.rad2deg(theta1_a), " deg, ",
np.rad2deg(theta1_b), " deg")
    print("Angle of link 2 ", np.rad2deg(theta2), " deg")
    print("Link extension : ", d3, " m")
```

Output:

```
Inverse kinematics of Stanford manipulator :

Enter length of Link 1 (m): 0.2
Enter length of Link 2 (m): 0.15
Enter length of Link 3 (m): 0.3
Enter x-coordinate of end-effector (m): 0.15
Enter y-coordinate of end-effector (m): 0.1
Enter z-coordinate of end-effector (m): 0.25

Inverse kinematics solutions :

Angle of link 1  56.309932474020215  deg,  236.3099324740202  deg
Angle of link 2  164.498640433063  deg
Link extension :  0.18708286933869708  m
```

Question 13

Source Code: ([Link to GitHub]())

```python
import numpy as np

print("Inverse kinematics of SCARA manipulator :\n")

# Take input for link lengths
l1 = float(input("Enter length of Link 1 (m): "))
l2 = float(input("Enter length of Link 2 (m): "))
l3 = float(input("Enter length of Link 3 (m): "))
d1 = float(input("Enter height at which Joint 1 is located (m): "))

# Take input for end-effector coordinates
xc = float(input("Enter x-coordinate of end-effector (m): "))
yc = float(input("Enter y-coordinate of end-effector (m): "))
zc = float(input("Enter z-coordinate of end-effector (m): "))

# Compute inverse kinematics solutions
ox = xc
oy = yc
oz = zc
c2 = (ox**2 + oy**2 - l1**2 - l2**2)/(2 * l1 * l2)
q2 = np.arctan2(c2, np.sqrt(1 - c2))
q1 = np.arctan2(ox, oy) - np.arctan2(l1 + l2 * np.cos(q2), l2 * np.sin(q2))
d3 = d1 - zc

# Output the computed values
if oz > d1:
    print("Point lies outside the workspace of the robot!")
else:
    print("\nInverse kinematics solutions :\n")
    print("Angle of link 1 ", np.rad2deg(q1), " deg, ")
    print("Angle of link 2 ", np.rad2deg(q2), " deg")
    print("Link extension : ", d3, " m")
```

Output:

```
Inverse kinematics of SCARA manipulator :

Enter length of Link 1 (m): 0.3
Enter length of Link 2 (m): 0.2
Enter length of Link 3 (m): 0.1
Enter height at which Joint 1 is located (m): 0.4
Enter x-coordinate of end-effector (m): 0.25
Enter y-coordinate of end-effector (m): 0.15
Enter z-coordinate of end-effector (m): 0.3

Inverse kinematics solutions :

Angle of link 1  -38.043773699846184  deg,
Angle of link 2  -17.734332140549498  deg
Link extension :  0.10000000000000003  m
```

Question 14

Source Code: (Link to GitHub)

```
import numpy as np

def calculate_joint_velocities(jacobian, end_effector_linear_velocities,
end_effector_angular_velocities):
    try:
        end_effector_velocities =
np.concatenate((end_effector_linear_velocities,
end_effector_angular_velocities))
        jacobian_inv = np.linalg.pinv(jacobian)
        joint_velocities = np.dot(jacobian_inv, end_effector_velocities)
        return joint_velocities[:3], joint_velocities[3:]
    except np.linalg.LinAlgError:
        return None

num_links = int(input("Enter the number of links: "))

jacobian = np.empty((6, num_links))

print("\nEnter the elements of the Jacobian matrix (row by row):")
for i in range(6):
    for j in range(num_links):
        jacobian[i][j] = float(input(f"Jacobian[{i}][{j}]: "))

end_effector_linear_velocities = np.array([float(v) for v in input("\nEnter
end-effector linear velocities (comma-separated) (m/s): ").split(',')])
end_effector_angular_velocities = np.array([float(v) for v in input("Enter
end-effector angular velocities (comma-separated) (rad/s): ").split(',')])

# Calculate joint velocities
joint_linear_velocities, joint_angular_velocities =
calculate_joint_velocities(jacobian, end_effector_linear_velocities,
end_effector_angular_velocities)

if joint_linear_velocities is not None:
    print("\nJoint Linear Velocities:", joint_linear_velocities)
    print("Joint Angular Velocities:", joint_angular_velocities)
else:
    print("\nJacobian is singular. Cannot calculate joint velocities.")
```

Output:

```
Enter the number of links: 2

Enter the elements of the Jacobian matrix (row by row):
Jacobian[0][0]: -1.9869
Jacobian[0][1]: -0.9869
Jacobian[1][0]: 1.9909
Jacobian[1][1]: 0.2588
Jacobian[2][0]: 0
Jacobian[2][1]: 0
Jacobian[3][0]: 0
Jacobian[3][1]: 0
Jacobian[4][0]: 0
Jacobian[4][1]: 0
Jacobian[5][0]: 1
Jacobian[5][1]: 1

Enter end-effector linear velocities (comma-separated) (m/s): 0.2, 0.2, 0
Enter end-effector angular velocities (comma-separated) (rad/s): 0, 0, 0

Joint Linear Velocities: [ 0.08318945 -0.21303629]
Joint Angular Velocities: []
```

Question 17

Source Code: (Link to GitHub)

```
import numpy as np

print("Inverse kinematics of Spherical Wrist manipulator :\n")

# Take input for link lengths
q1 = float(input("Enter angle of Joint 1 (deg): "))
q2 = float(input("Enter angle of Joint 2 (deg): "))
q3 = float(input("Enter angle of Joint 3 (deg): "))

c1 = np.cos(q1)
c23 = np.cos(q2 + q3)
s1 = np.sin(q1)
s23 = np.sin(q2 + q3)

# Define the rotation matrix of the current end-effector orientation
R_current = np.array([[-c1 * c23, -c1 * s23, s1],
                      [s1 * c23, -s1 * s23, -c1],
                      [s23, c23, 0]])

# Take input of desired end-effector orientation
print("\nEnter the elements of the desired end-effector orientation matrix
(row by row):")
R_desired = np.empty((3,3))
for i in range(3):
    for j in range(3):
        R_desired[i][j] = float(input(f"R_desired[{i}][{j}]: "))

# Calculate the transformation matrix that transforms from the current to the
desired orientation
R_d2c = np.dot(R_current.T, R_desired)

# Extract the z-y-z Euler angles from the rotation matrix
phi = np.arctan2(R_d2c[1, 2], R_d2c[0, 2])
theta = np.arccos(R_d2c[2, 2])
psi = np.arctan2(R_d2c[2, 1], -R_d2c[2, 0])

# Print the results
print("\nDesired Rotation Matrix:")
print(R_desired)
print("\nCurrent Rotation Matrix:")
print(R_current)
print("\nCalculated Z-Y-Z Euler Angles:")
print("phi = ", np.degrees(phi), " deg")
```

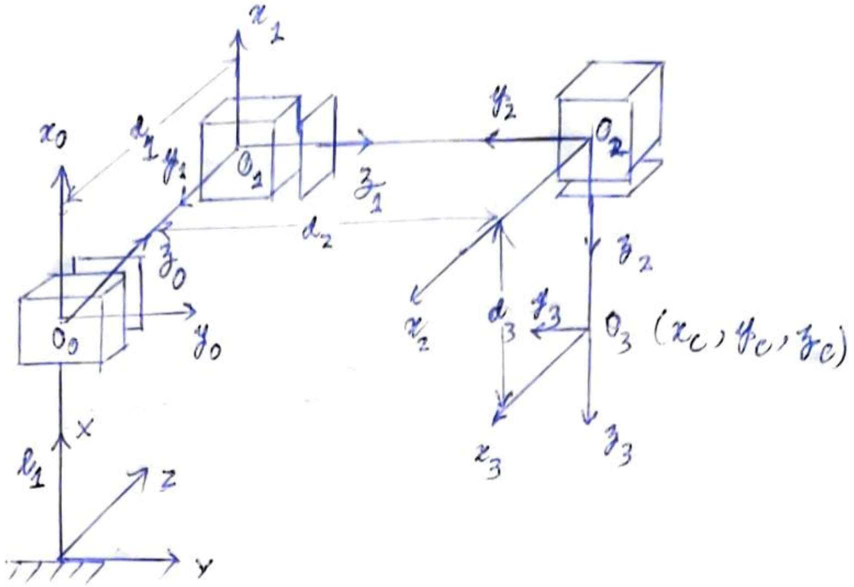
```python
print("theta = ", np.degrees(theta), " deg")
print("psi = ", np.degrees(psi), " deg")
```

Output:

```
Inverse kinematics of Spherical Wrist manipulator :

Enter angle of Joint 1 (deg): 15
Enter angle of Joint 2 (deg): 20
Enter angle of Joint 3 (deg): 30

Enter the elements of the desired end-effector orientation matrix (row by row):
R_desired[0][0]: 0.866
R_desired[0][1]: -0.5
R_desired[0][2]: 0
R_desired[1][0]: 0.5
R_desired[1][1]: 0.866
R_desired[1][2]: 0
R_desired[2][0]: 0
R_desired[2][1]: 0
R_desired[2][2]: 1

Desired Rotation Matrix:
[[ 0.866 -0.5    0.   ]
 [ 0.5    0.866  0.   ]
 [ 0.     0.     1.   ]]

Current Rotation Matrix:
[[ 0.73307303 -0.199323    0.65028784]
 [ 0.62750567  0.17061918  0.75968791]
 [-0.26237485  0.96496603  0.        ]]

Calculated Z-Y-Z Euler Angles:
phi =  105.21102434588396  deg
theta =  90.0  deg
psi =  160.56403508459258  deg
```

Question 18

Q18



DH parameters :

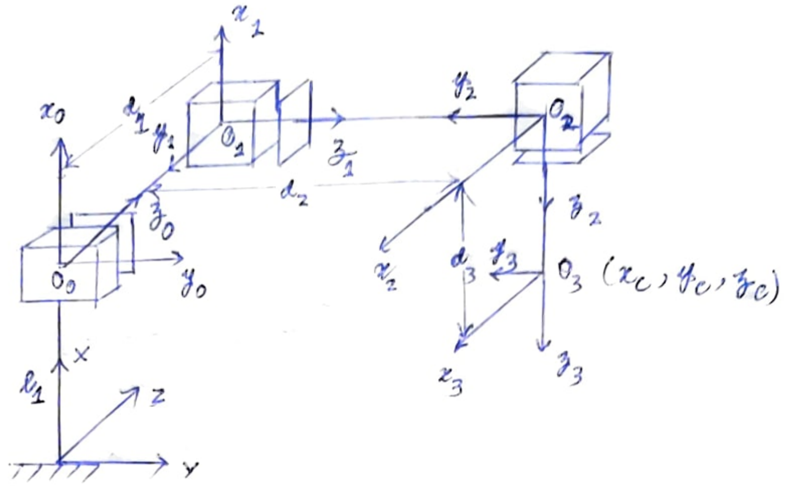
Link	d	θ	a	α
1	d_1^*	0	0	$-\pi/2$
2	d_2^*	$\pi/2$	0	$\pi/2$
3	d_3^*	$-\pi/2$	0	0

Output from subroutines:

```
Do you want to :
1. Use default values?
2. Enter your own values?
Enter option no. : 2

Enter the number of links : 3

Enter the following values row-wise, separated by commas :
Enter link lengths in m  : 2, 2, 2
Enter values of DH Parameters (Format : a, alpha, d, theta) : 0, -1.5708, 1, 0, 0, 1.5708, 1, 1.5708, 0, 0, 1, -1.5708
Enter joint types (R or P) : P, P, P
Enter end effector velocities (Format : v_x, v_y, v_z, w_x, w_y, w_z) :
0.3, 0.2, 0.1, 0, 0, 0

Link lengths (m) :
[2.0, 2.0, 2.0]

DH Parameters (Format : a, alpha, d, theta) :
[[ 0.     -1.5708  1.      0.    ]
 [ 0.      1.5708  1.      1.5708]
 [ 0.      0.      1.     -1.5708]]

Joint types :
['P', ' P', ' P']

End-effector velocities :
[0.3 0.2 0.1 0.  0.  0. ]

Manipulator Jacobian:
[[ 1.00000000e+00 -1.34924357e-11 -1.34924357e-11]
 [-0.00000000e+00 -9.99996327e-01 -9.99996327e-01]
 [ 0.00000000e+00  1.00000000e+00  1.00000000e+00]
 [ 0.00000000e+00  1.00000000e+00  1.00000000e+00]
 [ 1.00000000e+00 -3.67321860e-06 -3.67321860e-06]
 [-3.67320510e-06 -3.67319161e-06 -3.67319161e-06]]

End-effector Position:
[1.         0.99999633 0.99999265]

Joint Velocities:
[ 0.14999994 -0.01666649 -0.01666649]
```

Question 19

Q 19



DH parameters :

Link	d	θ	a	α
1	d_1^*	0	0	$-\pi/2$
2	d_2^*	$\pi/2$	0	$\pi/2$
3	d_3^*	$-\pi/2$	0	0

Let $(x_c, y_c$ & $z_c)$ denote the coordinates of the end-effector.

∴ The inverse kinematics solutions are :

$$\boxed{\begin{aligned} d_1 &= z_c \\ d_2 &= y_c \\ d_3 &= l_1 - x_c \end{aligned}}$$

REFERENCES

1. Mark W. Spong, Seth Hutchinson, and M. Vidyasagar. *Robot Dynamics and Control*. https://www.kramirez.net/Robotica/Tareas/Kinematics.pdf

2. Why every gfx/CV/robotics programmer should love SymPy (Part 1): https://mzucker.github.io/2018/04/06/why-every-gfx-cv-robotics-programmer-should-love-sympy.html

3. Solving symbolic equations with SymPy: https://scaron.info/blog/solving-symbolic-equations-with-sympy.html

4. NumPy Documentation: https://numpy.org/doc/stable/

5. SymPy Documentation: https://docs.sympy.org/latest/index.html

6. DH parameters for a PPP arm: https://robotics.stackexchange.com/questions/18554/dh-parameters-for-a-ppp-arm

7. Robotic 09_ inverse kinematics Example 02 (three link Robot Cartesian Robot PPP): https://www.youtube.com/watch?v=6k53jtwT9dk

8. ChatGPT: https://chat.openai.com/