IN3140 Assignment 2

Vetle H. Olavesen

March 18, 2024

Task 1 - Forward and inverse kinematics

a, b)

I still couldn't figure out the expression for θ_2 and θ_3 from last assignment, i just got lost in the sauce on that one.

```python
import numpy as np
from numpy import cos, sin, arctan
# Declaring variables

L1 = 100.9 #[mm] Length of link 1.
L2 = 222.1 #[mm] Length of link 2.
L3 = 136.2 #[mm] Length of link 3.




def forward(joint_angles):
    """
    The forward kinematics function takes x sets of joint angles as input,
    and gives the corresponding cartesian coordinates for the tip of the arm
    as output, takes in joint anglesas: [theta1, theta2, theta3], returns as array: [x,y,z]
    """
    th1 = joint_angles[0]
    th2 = joint_angles[1]
    th3 = joint_angles[2]
    x = cos(th1)*( (L2*cos(th2)) + (L3*cos(th2 + th3)) )
    y = sin(th1)*( (L2*cos(th2)) + (L3*cos(th2 + th3)) )
    z = (L1*L2*sin(th2)) + (L3*sin(th2 + th3))
    return np.array([x,y,z])

def inverse(cart_cord):
    """
    The inverse kinematics function takes the cartesian position of the tip
    of the pen as input, and gives the corresponding joint configurations as
    output
    """
    x = cart_cord[0]
    y = cart_cord[1]
    z = cart_cord[2]

    th1 = arctan(x/y)
    th2 = y
    th3 = z
    return [th1, th2, th3]
```

Listing 1: Forwards and Inverse Kinematics implemented in python

c)

If i feed specific angles into the forward kinematics function, and feed the point given by that function into the inverse kinematics function, it should give me the angles i gave the forward kinematics function. Could also generate random points, then automatically assert the given angles by the inverse function, but i will keep it simple for now:

```python
point = forward([90, -30, 45])
angles = inverse(point)

print(f"Angles fed into forward function: {[90, -30, 45]}, point given by forward function: {point
    }. Which is fed into inverse function and got these angles: {angles}")
```

Listing 2: Testing of Forward and Inverse Kinematics functions

d)

For this, i would feed the point into the inverse function, then feed the angles given by that, into the forward function, and format it correctly.

Task 2 - Jacobian I

a)

To find the Jacobian matrix, the jacobian is given as:

$$J = \begin{bmatrix} J_v \\ J_w \end{bmatrix}$$

J_v is the linear velocity part of the Jacobian, while J_w is the angular velocity part. I will first find J_v. Each collumn J_{v_i} and J_{w_i} is given as (we are only dealing with revolute jointes in the robot, so i only list the revolute formulas):

$$J_{v_i} = Z_{i-1} \times (O_n - O_{i-1})$$

$$J_{w_i} = Z_{i-1}$$

The Z comes from the third column for the respective forwards transformation matrices, except Z_0, $Z_0 = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$, Z_1 comes from T_1^0, Z_2 comes from T_2^0, Z_3 comes from T_3^0, the O's come from the fourth column, which i will list below as we found them in the previous assignment:

$$T_1^0 = \begin{bmatrix} C_{\theta_1} & 0 & S_{\theta_1} & 0 \\ S_{\theta_1} & 0 & -C_{\theta_1} & 0 \\ 0 & 1 & 0 & l_1 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$T_2^0 = \begin{bmatrix} C_{\theta_1}C_{\theta_2} & -S_{\theta_2}C_{\theta_1} & S_{\theta_1} & l_2C_{\theta_1}C_{\theta_2} \\ S_{\theta_1}C_{\theta_2} & -S_{\theta_2}S_{\theta_1} & -C_{\theta_1} & l_2C_{\theta_2}S_{\theta_1} \\ S_{\theta_2} & C_{\theta_2} & 0 & l_2S_{\theta_2} + l_1 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$T_3^0 = \begin{bmatrix} C_{\theta_1} * C_{\theta_2+\theta_3} & -C_{\theta_1} * S_{\theta_2+\theta_3} & S_{\theta_1} & C_{\theta_1} * (l_2 * C_{\theta_2} + l_3 * C_{\theta_2+\theta_3}) \\ S_{\theta_1} * C_{\theta_2+\theta_3} & -S_{\theta_1} * S_{\theta_2+\theta_3} & -C_{\theta_1} & S_{\theta_1} * (l_2 * C_{\theta_2} + l_3 * C_{\theta_2+\theta_3}) \\ S_{\theta_2+\theta_3} & C_{\theta_2+\theta_3} & 0 & l_1 * l_2 * S_{\theta_2} + l_3 * S_{\theta_2+\theta_3} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

The jacobian will be:

$$J = \begin{bmatrix} Z_0 \times (O_3 - O_0) & Z_1 \times (O_3 - O_1) & Z_2 \times (O_3 - O_2) \\ Z_0 & Z_1 & Z_2 \end{bmatrix}$$

Since all joints are revolute. Then we find all Z's and all O's:

$$Z_0 = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}, Z_1 = \begin{bmatrix} S_{\theta_1} \\ -C_{\theta_1} \\ 0 \end{bmatrix}, Z_2 = \begin{bmatrix} S_{\theta_1} \\ -C_{\theta_1} \\ 0 \end{bmatrix}$$

$$O_0 = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}, O_1 = \begin{bmatrix} 0 \\ 0 \\ l_1 \end{bmatrix}, O_2 = \begin{bmatrix} l_2 C_{\theta_1} C_{\theta_2} \\ l_2 C_{\theta_2} S_{\theta_1} \\ l_2 S_{\theta_2} + l_1 \end{bmatrix}, O_3 = \begin{bmatrix} C_{\theta_1} * (l_2 * C_{\theta_2} + l_3 * C_{\theta_2+\theta_3}) \\ S_{\theta_1} * (l_2 * C_{\theta_2} + l_3 * C_{\theta_2+\theta_3}) \\ l_1 * l_2 * S_{\theta_2} + l_3 * S_{\theta_2+\theta_3} \end{bmatrix}$$

J_w is already done, so we find J_v :

$$J_{v_1} = Z_0 \times (O_3 - O_0)$$

$$(O_3 - O_0) = \begin{bmatrix} C_{\theta_1} * (l_2 * C_{\theta_2} + l_3 * C_{\theta_2+\theta_3}) \\ S_{\theta_1} * (l_2 * C_{\theta_2} + l_3 * C_{\theta_2+\theta_3}) \\ l_1 * l_2 * S_{\theta_2} + l_3 * S_{\theta_2+\theta_3} \end{bmatrix}$$

Do the cross product (a,b,c is x,y,z of $(O_3 - O_0)$):

$$J_{v_1} = \begin{bmatrix} x & y & z & x & y \\ 0 & 0 & 1 & 0 & 0 \\ a & b & c & a & b \end{bmatrix} = \begin{bmatrix} (0 * c - 1 * b) \\ (1 * a - 0 * c) \\ (0 * b - 0 * a) \end{bmatrix} = \begin{bmatrix} S_{\theta_1} * (l_2 * C_{\theta_2} + l_3 * C_{\theta_2+\theta_3}) \\ C_{\theta_1} * (l_2 * C_{\theta_2} + l_3 * C_{\theta_2+\theta_3}) \\ 0 \end{bmatrix}$$

We do the same for the others columns of J_v:

$$(O_3 - O_1) = \begin{bmatrix} C_{\theta_1} * (l_2 * C_{\theta_2} + l_3 * C_{\theta_2+\theta_3}) \\ S_{\theta_1} * (l_2 * C_{\theta_2} + l_3 * C_{\theta_2+\theta_3}) \\ l_1 * l_2 * S_{\theta_2} + l_3 * S_{\theta_2+\theta_3} - l_1 \end{bmatrix}$$

$$J_{v_2} = \begin{bmatrix} x & y & z & x & y \\ S_{\theta_1} & -C_{\theta_1} & 0 & S_{\theta_1} & -C_{\theta_1} \\ a & b & c & a & b \end{bmatrix} = \begin{bmatrix} (-C_{\theta_1} * c - 0 * b) \\ (0 * a - S_{\theta_1} * c) \\ (-C_{\theta_1} * a - S_{\theta_1} * b) \end{bmatrix}$$

$$J_{v_2} = \begin{bmatrix} -C_{\theta_1} * (l_1 * l_2 * S_{\theta_2} + l_3 * S_{\theta_2+\theta_3} - l_1) \\ -S_{\theta_1} * (l_1 * l_2 * S_{\theta_2} + l_3 * S_{\theta_2+\theta_3} - l_1) \\ -C_{\theta_1}^2 * (l_2 * C_{\theta_2} + l_3 * C_{\theta_2+\theta_3}) \end{bmatrix}$$

$$(O_3 - O_2) = \begin{bmatrix} C_{\theta_1} * (l_2 * C_{\theta_2} + l_3 * C_{\theta_2+\theta_3}) - l_2 C_{\theta_1} C_{\theta_2} \\ S_{\theta_1} * (l_2 * C_{\theta_2} + l_3 * C_{\theta_2+\theta_3}) - l_2 C_{\theta_2} S_{\theta_1} \\ l_1 * l_2 * S_{\theta_2} + l_3 * S_{\theta_2+\theta_3} - l_2 S_{\theta_2} - l_1 \end{bmatrix}$$

$$J_{v_3} = \begin{bmatrix} x & y & z & x & y \\ S_{\theta_1} & -C_{\theta_1} & 0 & S_{\theta_1} & -C_{\theta_1} \\ a & b & c & a & b \end{bmatrix} = \begin{bmatrix} (-C_{\theta_1} * c - 0 * b) \\ (0 * a - S_{\theta_1} * c) \\ (-C_{\theta_1} * a - S_{\theta_1} * b) \end{bmatrix}$$

$$J_{v_3} = \begin{bmatrix} -C_{\theta_1} * (l_1 * l_2 * S_{\theta_2} + l_3 * S_{\theta_2+\theta_3} - l_2 S_{\theta_2} - l_1) \\ -S_{\theta_1} * (l_1 * l_2 * S_{\theta_2} + l_3 * S_{\theta_2+\theta_3} - l_2 S_{\theta_2} - l_1) \\ -(l_2 * C_{\theta_2} + l_3 * C_{\theta_2+\theta_3}) + l_2 C_{\theta_2} \end{bmatrix}$$

Now that we have all the expressions, we can put together the jacobian:

$$J = \begin{bmatrix} S_{\theta_1} * (l_2 * C_{\theta_2} + l_3 * C_{\theta_2+\theta_3}) & -C_{\theta_1} * (l_1 * l_2 * S_{\theta_2} + l_3 * S_{\theta_2+\theta_3} - l_1) & -C_{\theta_1} * (l_1 * l_2 * S_{\theta_2} + l_3 * S_{\theta_2+\theta_3} - l_2 S_{\theta_2} - l_1) \\ C_{\theta_1} * (l_2 * C_{\theta_2} + l_3 * C_{\theta_2+\theta_3}) & -S_{\theta_1} * (l_1 * l_2 * S_{\theta_2} + l_3 * S_{\theta_2+\theta_3} - l_1) & -S_{\theta_1} * (l_1 * l_2 * S_{\theta_2} + l_3 * S_{\theta_2+\theta_3} - l_2 S_{\theta_2} - l_1) \\ 0 & -C_{\theta_1}^2 * (l_2 * C_{\theta_2} + l_3 * C_{\theta_2+\theta_3}) & -(l_2 * C_{\theta_2} + l_3 * C_{\theta_2+\theta_3}) + l_2 C_{\theta_2} \\ 0 & S_{\theta_1} & S_{\theta_1} \\ 0 & -C_{\theta_1} & -C_{\theta_1} \\ 1 & 0 & 0 \end{bmatrix}$$

b)

To find the singularities of the robot using the Jacobian, we can use this formula $det(J_v) = 0$, i simplified the expressions by subbing some smaller expressions with a,b,c,d ($C_{2*1} = cos(2\theta_1)$):

$$a = L_2C_2 + L_3 * C_{2+3}, b = L_1L_2S_2 + L_3S_{2+3} - L_1, c = L_2C_2, d = L_2S_2$$

$$det(J_v) = S_1 * a * [S_1 * b * (-a + c) + S_1 * (b - d) * C_1^2 * a] + C_1 * b * [C_1 * a * (-a + c)] - C_1 * (b - d) * [C_1^3 * a^2]$$

$$\Rightarrow ab(c - a)(S_1^2 + C_1^2) - C_1a^2(b - d)(C_1^2 - S_1^2)$$

$$\Rightarrow ab(c - a) - C_1a^2(b - d)C_{2*1}$$

From this we see that it equals 0 if $a = 0$:

$$L_2C_2 + L_3 * C_{2+3} = 0$$

From this we can tell that singularities rely on θ_2 and θ_3. Meaning that θ_1 doesn't do anything for the singularity, which i think makes sense according to how our joints are positioned. I couldn't find a specific angle for task 2d though.

c)

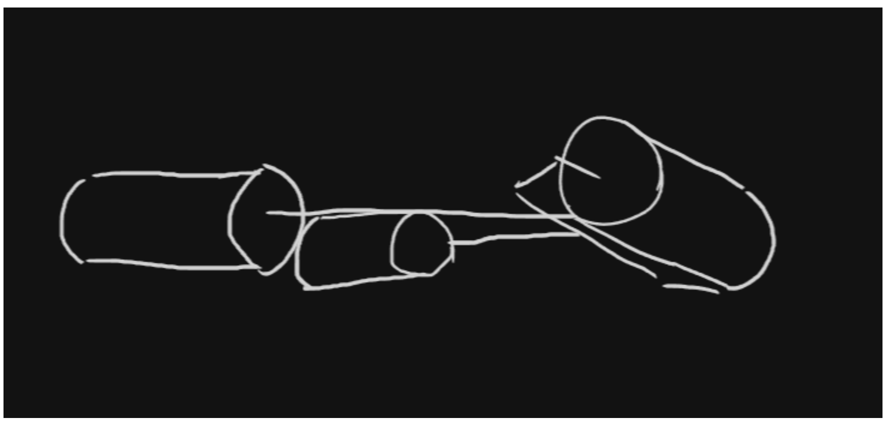
We call these configurations singularities, these are configs of the robot were it is impossible for it to move in a certain direction, and f.ex: require 'infinite' force. We get a singularity when the robot is configured so that it looses degrees of freedom, this leads to loosing control over the robot.

d)

I couldn't find a specific angle.

e)

The spherical wrist extension will encounter a singularity, when the axes of its rotary joints become parallel, or overlap, example in drawing:



f)

If we don't handle the singularities, and try to force our robot into singularity configurations, we might end up snapping or breaking our robot/joints.

Task 3

a, b)

I implemented the jacobian in python:

```python
import numpy as np
from numpy import cos, sin
# Declaring variables

L1 = 100.9 #[mm] Length of link 1.
L2 = 222.1 #[mm] Length of link 2.
L3 = 136.2 #[mm] Length of link 3.


# For this function, im assuming, we only want the linear velocities.

def jacobian(joint_angles, joint_velocities):
    """
    It takes the instant joint angles and joint velocities as input, and gives a
    3-dimensional vector of cartesian velocities of the tip of the pen as output
    """
    th1 = joint_angles[0]
    th2 = joint_angles[1]
    th3 = joint_angles[2]
    q1 = joint_velocities[0]
    q2 = joint_velocities[1]
    q3 = joint_velocities[2]

    j_v = [[sin(th1)*(L2*cos(th2) + L3*cos(th2+th3)), -cos(th1)*((L1*L2*sin(th2)) + (L3*sin(th2+th3)) - L1), -cos(th1)*((L1*L2*sin(th2)) + (L3*sin(th2+th3)) - (L2*sin(th2)) - L1)],
           [cos(th1)*(L2*cos(th2) + L3*cos(th2+th3)), -sin(th1)*((L1*L2*sin(th2)) + (L3*sin(th2+th3)) - L1), -sin(th1)*((L1*L2*sin(th2)) + (L3*sin(th2+th3)) - (L2*sin(th2)) - L1)],
           [0, -((cos(th1))**2)*((L2*cos(th2)) + (L3*cos(th2+th3))), -((L2*cos(th2)) + (L3*cos(th2+th3))) + (L2*cos(th2))]]

    vel_x = (j_v[0][0]*q1) + (j_v[0][1]*q2) + (j_v[0][2]*q3)
    vel_y = (j_v[1][0]*q1) + (j_v[1][1]*q2) + (j_v[1][2]*q3)
    vel_z = (j_v[2][0]*q1) + (j_v[2][1]*q2) + (j_v[2][2]*q3)

    return [vel_x, vel_y, vel_z]


jnt_angls = [90, -30, 45]
jnt_vels = [.1, .05, .05] # rad/s

print(jacobian(jnt_angls, jnt_vels))
```

Listing 3: Jacobian Implemented in Python

The output from this python script is: [980.4540750533117, -1965.4462952355411, 5.868241628692565]