Task 1.a Forward kinematics function

Listing 1: Python kode

```
import numpy as np
import math

L1 = 100.9 #mm
L2 = 222.1 #mm
L3 = 136.2 #mm

def forward(theta_1,theta_2,theta_3):
        theta_1 = (theta_1/180.0)*np.pi
        theta_2 = (theta_2/180.0)*np.pi
        theta_3 = (theta_3/180.0)*np.pi

        RTz_1 = [[np.cos(theta_1),-np.sin(theta_1),0,0],
                 [np.sin(theta_1),np.cos(theta_1),0,0],
                 [0,0,1,L1],
                 [0,0,0,1]]
        RTx_1 = [[1,0,0,0],
                 [0,0,-1,0],
                 [0,1,0,0],
                 [0,0,0,1]]
        RTz_2 = [[np.cos(theta_2),-np.sin(theta_2),0,0],
                 [np.sin(theta_2),np.cos(theta_2),0,0],
                 [0,0,1,0],
                 [0,0,0,1]]
        RTx_2 = [[1,0,0,L2],
                 [0,1,0,0],
                 [0,0,1,0],
                 [0,0,0,1]]
        RTz_3 = [[np.cos(theta_3),-np.sin(theta_3),0,0],
                 [np.sin(theta_3),np.cos(theta_3),0,0],
                 [0,0,1,0],
                 [0,0,0,1]]
        RTx_3 = [[1,0,0,L3],
```

```
            [0,1,0,0],
            [0,0,1,0],
            [0,0,0,1]]


H0_1 = np.dot(RTz_1,RTx_1)
H1_2 = np.dot(RTz_2,RTx_2)
H2_3 = np.dot(RTz_3,RTx_3)
cart_cord = np.dot(np.dot(H0_1,H1_2),H2_3)

return cart_cord
```

Task 1.b Inverse kinematics function

Listing 2: Python kode

```
import numpy as np
import math

L1 = 100.9 #mm
L2 = 222.1 #mm
L3 = 136.2 #mm

def inverse(X,Y,Z):

    theta_1 = np.arctan2(Y,X)

    S1 = math.sqrt(X*X + Y*Y)
    S2 = Z - L1
    S3 = math.sqrt(S1*S1 + S2*S2)
    phi_1 = math.acos((L2*L2 + S3*S3 - L3*L3)/(2*L2*S3))
    phi_2 = np.arctan2(S2,S1)
    theta_2 = (phi_2-phi_1)

    phi_3 = math.acos((L2*L2 + L3*L3 - S3*S3)/(2*L2*L3))
    theta_3 = (np.pi - phi_3)

    joint_angles = [theta_1*(180/np.pi),theta_2*(180/np.pi),theta_3*(180/np.pi)]

    return joint_angles

print(inverse(0,358,101))
```

Task 1.c Using the function

Using the functions to show how we can verify that the inverse and forward kinematics are correctly derived.

Listing 3: Python kode

```
import numpy as np
import math

L1 = 100.9 #mm
L2 = 222.1 #mm
L3 = 136.2 #mm

print(forward(-90,30,-45))
print(inverse(0,-323.9033,176.6988))
```

Task 1.d All possible sets of solutions

The TCP (Tool Center Point) is located at [x,y,z]=[0;-323.9033;176.6988] in the Base coordinate frame. All possible sets of solutions of the joint variables according to the DH setup is elbow up configuration, elbow down configuration.

Listing 4: Python kode

```
import numpy as np
import math

L1 = 100.9 #mm
L2 = 222.1 #mm
L3 = 136.2 #mm

print(forward(270,60,45))
print(forward(-90,60,45))

print(forward(270,120,-45))
print(forward(-90,120,-45))

print(inverse(0,-323.9033,176.6988))
```

Task 2.a - Deriving the Jacobian matrix for the simplified CrustCrawler robot

Jacobian can be describe as follow.

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{z} \\ \omega_x \\ \omega_y \\ \omega_z \end{bmatrix} = J. \begin{bmatrix} \dot{\theta_1} \\ \dot{\theta_2} \\ \dot{\theta_3} \end{bmatrix}$$

where J represent Jacobian matrix and q represent either prismatic or revolute according to the joint variable. In our task we have three revolute joint.

In Jacobian matrix, there has 6 rows and n column where the column is depends on the number of frame which we have based on the robot manipulator. In our problem, we have three frame, meaning Jacobian should be 6x3 matrix. We can describe the Jacobian matrix as follow.

$$J = \begin{bmatrix} J_v \\ J_\omega \end{bmatrix}$$

where J_v is linear and J_ω is the rotational part.

For simplicity, here in the table below shows how we can build Jacobian matrix.

	Prismatic joint	**Revolute joint**
Linear(J_v)	$R^0_{i-1} \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$	$R^0_{i-1} \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$ x (d^0_n)-(d^0_{i-1}) (or) Z^0_{i-1} x $(O^0_3 - O^0_{i-1})$
Rotational(J_ω)	$\begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$	$R^0_{i-1} \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$ (or) Z^0_{i-1}

Table 1: Jacobian table

where i refers to frame 1, 2, 3 and n refers to the number of we have. In our task we have 3 frame. According to the table we can have prismatic and rotational Jacobian as follow.

$$J_v = \begin{bmatrix} Z^0_0 \times (O^0_3 - O^0_0) & Z^0_1 \times (O^0_3 - O^0_1) & Z^0_2 \times (O^0_3 - O^0_2) \end{bmatrix}$$

$$J_\omega = \begin{bmatrix} Z^0_0 & Z^0_1 & Z^0_2 \end{bmatrix}$$

We have homogenous transformation matrix from Task1.

$$H_1^0 = \begin{bmatrix} C_1 & 0 & S_1 & 0 \\ S_1 & 0 & -C_1 & 0 \\ 0 & 1 & 0 & L_1 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$H_2^0 = H_1^0 H_2^1 \tag{1}$$

$$H_2^0 = \begin{bmatrix} C_1 & 0 & S_1 & 0 \\ S_1 & 0 & -C_1 & 0 \\ 0 & 1 & 0 & L_1 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} C_2 & -S_2 & 0 & C_2L_2 \\ S_2 & C_2 & 0 & S_2L_2 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$H_2^0 = \begin{bmatrix} C_1C_2 & -C_1S_2 & S_1 & C_1C_2L_2 \\ S_1C_2 & -S_1S_2 & -C_1 & S_1C_2L_2 \\ S_2 & C_2 & 0 & S_2L_2 + L_1 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$H_3^0 = \begin{bmatrix} C_1C_2C_3 - C_1S_2S_3 & -C_1C_2S_3 - C_1S_2C_3 & S_1 & C_1C_2C_3L_3 - C_1S_2S_3L_3 + C_1C_2L_2 \\ S_1C_2C3 - S_1S_2S_3 & -S_1C_2S_3 - S_1S_2C_3 & -C_1 & S_1C_2C_3L_3 - S_1S_2S_3L_3 + S_1C_2L_2 \\ S_2C3 + C_2S_3 & -S_2S_3 + C_2C_3 & 0 & S_2C_3L_3 + C_2S_3L_3 + S_2L_2 + L_1 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

As we imply this matrix in our Jacobian matrix we can have

$$Z_0^0 \times (O_3^0 - O_0^0) = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \times \begin{bmatrix} C_1C_2C_3L_3 - C_1S_2S_3L_3 + C_1C_2L_2 \\ S_1C_2C_3L_3 - S_1S_2S_3L_3 + S_1C_2L_2 \\ S_2C_3L_3 + C_2S_3L_3 + S_2L_2 + L_1 \end{bmatrix}$$

$$Z_0^0 \times (O_3^0 - O_0^0) = \begin{bmatrix} -S_1C_2C_3L_3 + S_1S_2S_3L_3 - S_1C_2L_2 \\ C_1C_2C_3L_3 - C_1S_2S_3L_3 + C_1C_2L_2 \\ 0 \end{bmatrix}$$

$$Z_1^0 \times (O_3^0 - O_1^0) = \begin{bmatrix} S_1 \\ -C_1 \\ 0 \end{bmatrix} \times \begin{bmatrix} C_1C_2C_3L_3 - C_1S_2S_3L_3 + C_1C_2L_2 \\ S_1C_2C_3L_3 - S_1S_2S_3L_3 + S_1C_2L_2 \\ S_2C_3L_3 + C_2S_3L_3 + S_2L_2 \end{bmatrix}$$

$$Z_1^0 \times (O_3^0 - O_1^0) = \begin{bmatrix} -C_1S_2C_3L_3 - C_1C_2S_3L_3 - C_1S_2L_2 \\ -S_1S_2C_3L_3 - S_1C_2S_3L_3 - S_1S_2L_2 \\ C_2C_3L_3 - S_2S_3L_3 + C_2L_2 \end{bmatrix}$$

$$Z_2^0 \times (O_3^0 - O_2^0) = \begin{bmatrix} S_1 \\ -C_1 \\ 0 \end{bmatrix} \times \begin{bmatrix} C_1C_2C_3L_3 - C_1S_2S_3L_3 \\ S_1C_2C_3L_3 - S_1S_2S_3L_3 \\ S_2C_3L_3 + C_2S_3L_3 \end{bmatrix}$$

$$Z_2^0 \times (O_3^0 - O_2^0) = \begin{bmatrix} -C_1S_2C_3L_3 - C_1C_2S_3L_3 \\ -S_1S_2C_3L_3 - S_1C_2S_3L_3 \\ C_2C_3L_3 - S_2S_3L_3 \end{bmatrix}$$

So we have J_v

$$J_v = \begin{bmatrix} -S_1C_2C_3L_3 + S_1S_2S_3L_3 - S_1C_2L_2 & -C_1S_2C_3L_3 - C_1C_2S_3L_3 - C_1S_2L_2 & -C_1S_2C_3L_3 - C_1C_2S_3L_3 \\ C_1C_2C_3L_3 - C_1S_2S_3L_3 + C_1C_2L_2 & -S_1S_2C_3L_3 - S_1C_2S_3L_3 - S_1S_2L_2 & -S_1S_2C_3L_3 - S_1C_2S_3L_3 \\ 0 & C_2C_3L_3 - S_2S_3L_3 + C_2L_2 & C_2C_3L_3 - S_2S_3L_3 \end{bmatrix}$$

$$J_\omega = \begin{bmatrix} Z_0^0 & Z_1^0 & Z_2^0 \end{bmatrix}$$

Z_0^0 represent at the Z position of identity matrix, Z_1^0 represent at the Z position of H_1^0 and Z_2^0 at Z position of H_2^0 at the rotation part respectively.

$$J_\omega = \begin{bmatrix} 0 & S_1 & S_1 \\ 0 & -C_1 & -C_1 \\ 1 & 0 & 0 \end{bmatrix}$$

We can now describe Jacobian matrix as we plug in J_v and J_ω.

$$J = \begin{bmatrix} -S_1C_2C_3L_3 + S_1S_2S_3L_3 - S_1C_2L_2 & -C_1S_2C_3L_3 - C_1C_2S_3L_3 - C_1S_2L_2 & -C_1S_2C_3L_3 - C_1C_2S_3L_3 \\ C_1C_2C_3L_3 - C_1S_2S_3L_3 + C_1C_2L_2 & -S_1S_2C_3L_3 - S_1C_2S_3L_3 - S_1S_2L_2 & -S_1S_2C_3L_3 - S_1C_2S_3L_3 \\ 0 & C_2C_3L_3 - S_2S_3L_3 + C_2L_2 & C_2C_3L_3 - S_2S_3L_3 \\ 0 & S_1 & S_1 \\ 0 & -C_1 & -C_1 \\ 1 & 0 & 0 \end{bmatrix}$$

Task 2.b

We call it singularity configuration for which rank J(q) is less than the maximum value. This is a configuration at which the Jacobian loses rank, meaning a configuration q such that rank J $\leqslant max_q$ rank J(q).

Task 2.c

Using the Jacobian matrix to find singularity configuration for the robot.

$\det(J_v) = 0$

We can do a more simplification of the above Jacobian matrix and describe J_{11} as below

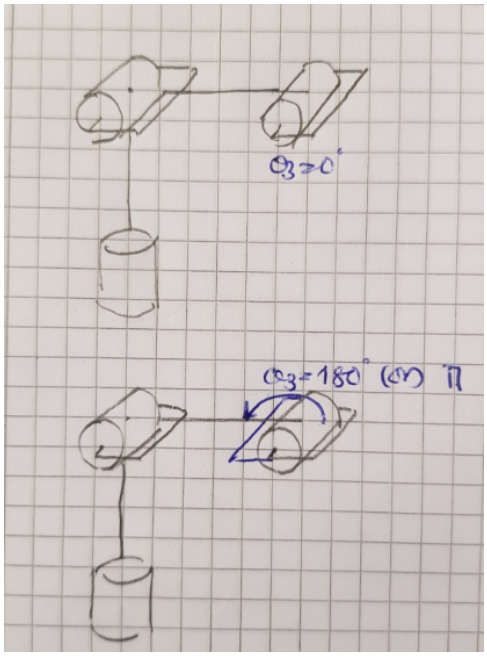
$$J_{11} = \begin{bmatrix} -L_2S_1C_2 - L_3S_1C_{23} & -L_2S_2C_1 - L_3S_{23}C_1 & -L_3C_1S_{23} \\ L_2C_1C_2 + L_3C_1C_{23} & -L_2S_1S_2 - L_3S_1S_{23} & -L_3S_1S_{23} \\ 0 & L_2C_2 + L_3C_{23} & L_3C_{23} \end{bmatrix}$$

$\det(J_{11}) = L_2L_3S_3(L_2C_2 + L_3C_{23})$.

Task2.d

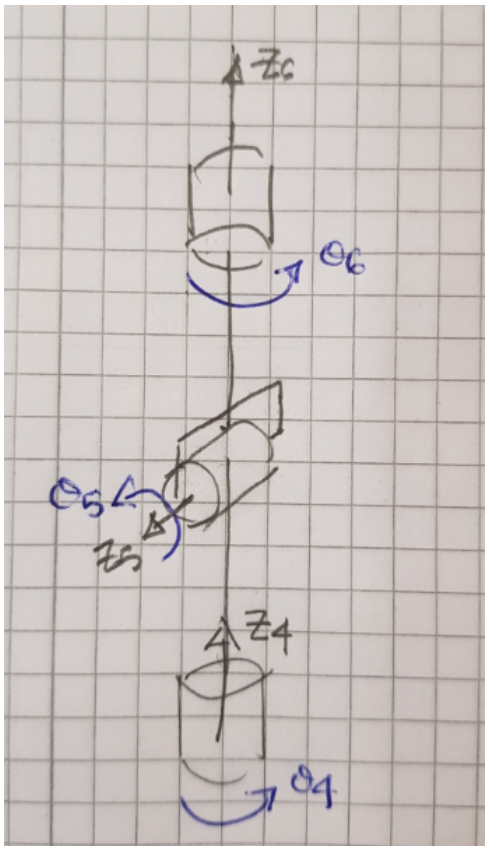
The manipulator has a singular configuration when

$S_3 = 0$, which means $\theta_3 = 0$ or π.



Task2.e

Spherical wrist singularities happen when two of the robot's wrist axes (joints 4 and 6) line up with each other. This can cause these joints to try and spin 180 degrees instantaneously.



Task2.f

Not handling a singularities in robotics meaning the axis loses one or more degree of freedom. As describe in above wrist singularities, the manipulator has lose one degree of freedom namely joint 5 and there will be infinite solution to the inverse position kinematics of the robot.

Task 3.a Jacobian function

Listing 5: Python kode

```
import numpy as np
import math
L1 = 100.9 #mm
L2 = 222.1 #mm
L3 = 136.2 #mm

def theta_velocities(t1dot, t2dot, t3dot):
    jv = np.array([t1dot,t2dot,t3dot])
    return jv

def jacobian(joint_angles, joint_velocities):

    joint_velocities = theta_velocities(2, 2, 2).reshape(3,1)
    joint_angles = forward(270,60,45)

    J11 = -np.sin(theta_1)*np.cos(theta_2)*np.cos(theta_3)*L3 + np.sin(theta_1)*np.sin(the
    J12 = -np.cos(theta_1)*np.sin(theta_2)*np.cos(theta_3)*L3 -np.cos(theta_1)*np.cos(theta
    J13 = -np.cos(theta_1)*np.sin(theta_2)*np.cos(theta_3)*L3 -np.cos(theta_1)*np.cos(theta
    J21 = np.cos(theta_1)*np.cos(theta_2)*np.cos(theta_3)*L3 -np.cos(theta_1)*np.sin(theta
    J22 = -np.sin(theta_1)*np.sin(theta_2)*np.cos(theta_3)*L3 -np.sin(theta_1)*np.cos(theta
    J23 = -np.sin(theta_1)*np.sin(theta_2)*np.cos(theta_3)*L3 -np.sin(theta_1)*np.cos(theta
    J31 = 0
    J32 = np.cos(theta_2)*np.cos(theta_3)*L3 -np.sin(theta_2)*np.sin(theta_3)*L3 + np.cos(
    J33 = np.cos(theta_2)*np.cos(theta_3)*L3 -np.sin(theta_2)*np.sin(theta_3)*L3

    J = [[J11, J12, J13],
         [J21, J22, J23],
         [J31, J32, J33]]

    cart_velocities = np.dot(J,joint_velocities.reshape(3,1))

    return cart_velocities
print(jacobian(joint_angles, joint_velocities))
```

Task 3.b Imply function

Listing 6: Python kode

```
import numpy as np
import math
L1 = 100.9 #mm
L2 = 222.1 #mm
L3 = 136.2 #mm

def theta_velocities(t1dot, t2dot, t3dot):
    jv = np.array([t1dot,t2dot,t3dot])
    return jv

def jacobian(joint_angles, joint_velocities):

    joint_velocities = theta_velocities(0.1,0.05,0.05).reshape(3,1)
    joint_angles = forward(270,60,45)

    J11 = -np.sin(theta_1)*np.cos(theta_2)*np.cos(theta_3)*L3 + np.sin(theta_1)*np.sin(the
    J12 = -np.cos(theta_1)*np.sin(theta_2)*np.cos(theta_3)*L3 -np.cos(theta_1)*np.cos(theta
    J13 = -np.cos(theta_1)*np.sin(theta_2)*np.cos(theta_3)*L3 -np.cos(theta_1)*np.cos(theta
    J21 = np.cos(theta_1)*np.cos(theta_2)*np.cos(theta_3)*L3 -np.cos(theta_1)*np.sin(theta_
    J22 = -np.sin(theta_1)*np.sin(theta_2)*np.cos(theta_3)*L3 -np.sin(theta_1)*np.cos(theta
    J23 = -np.sin(theta_1)*np.sin(theta_2)*np.cos(theta_3)*L3 -np.sin(theta_1)*np.cos(theta
    J31 = 0
    J32 = np.cos(theta_2)*np.cos(theta_3)*L3 -np.sin(theta_2)*np.sin(theta_3)*L3 + np.cos(
    J33 = np.cos(theta_2)*np.cos(theta_3)*L3 -np.sin(theta_2)*np.sin(theta_3)*L3

    J = [[J11, J12, J13],
         [J21, J22, J23],
         [J31, J32, J33]]

    cart_velocities = np.dot(J,joint_velocities.reshape(3,1))

    return cart_velocities
print(jacobian(joint_angles, joint_velocities))
```