1) Velocity FK (Geometric method)

$$\begin{pmatrix} \dot{x} \\ \dot{y} \\ \dot{z} \end{pmatrix} = \begin{pmatrix} J_v \\ J_\omega \end{pmatrix} \begin{pmatrix} \dot{q} \\ \dot{V} \end{pmatrix}$$

$\Rightarrow$ J_v

Prismatic $\rightarrow$ $J_v = \hat{z}^0_{i-1}$

Revolute $\rightarrow$ $J_v = \hat{z}_{0-1} \times (O^0_n - O^0_{i-1})$

$$O^0_n = T^0_e \rightarrow \begin{pmatrix} \cdot & \cdot & \cdot & O^0_n \\ \cdot & \cdot & \cdot & \\ \cdot & \cdot & \cdot & \\ & & & 1 \end{pmatrix}$$

$$O^0_{i-1} = T^0_{i-1} \begin{pmatrix} \cdot & \cdot & \cdot & O^0_{i-1} \\ \cdot & \cdot & \cdot & \\ \cdot & \cdot & \cdot & \\ & & & 1 \end{pmatrix}$$ $\rightarrow$ with offsets. $\rightarrow$ In our case I took joint position calculated in lab 1

$$\hat{z}_{i-1} = R^0_{i-1} \hat{z} \qquad \hat{z} = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$$

$$T^0_{i-1} = \begin{bmatrix} R^0_{i-1} & O^0_{i-1} \\ 0 & 1 \end{bmatrix}$$

$$J_v = (R^0_{i-1} \hat{z}) \times (\vec{O}^0_n - \vec{O}^0_{i-1})$$

$\Rightarrow$ J_ω

J_ω = Prismatic = 0

Revol $= \hat{z}_{i-1} = R^0_{i-1} \hat{z}$ $\qquad \hat{z} = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$

same as earlier from T^0_{i-1}

I wrote the code for geometric approach for Velocity FK. using the FK found earlier in LAB-1

From calculateFK.py I saved all T^0_{i-1} & joint positions.

```python
# This code is for Lab 2, you can ignore it ofr Lab 1
def get_axis_of_rotation(self, q):
    """
    INPUT:
    q - 1x7 vector of joint angles [q0, q1, q2, q3, q4, q5, q6]

    OUTPUTS:
    axis_of_rotation_list z_cap: - 3x7 np array of unit vectors describing the axis of rotation for each joint in the
                                   world frame

    """
    # STUDENT CODE HERE: This is a function needed by lab 2
    global z_cap
    z_cap=np.zeros((7,3))
    zth_axis= np.array([0,0,1])
    z_cap[0]=[0,0,1]
    for i in range(1,7):
        z_cap[i]= np.dot(R[i-1],zth_axis)


    return z_cap
```

using 3rd column of R^0_{i-1} finding $\hat{z}_{i-1}$

Already have $T^0_e \rightarrow$ end effector position from output of CalculateFK.py

also have joint position which already has offset of d included

$\hookrightarrow O^0_{i-1}$

So we have $\hat{z}_{i-1}$, O^0_n, O^0_{i-1} so we can find J_v & J_ω

& This Jacobian.

```python
calcJacobian.py M ×
lib > calcJacobian.py > ...
import numpy as np
from calculateFK import FK

def calcJacobian(q_in):
    """
    Calculate the full Jacobian of the end effector in a given configuration
    :param q_in: 1 x 7 configuration vector (of joint angles) [q1,q2,q3,q4,q5,q6,q7]
    :return: J - 6 x 7 matrix representing the Jacobian, where the first three
    rows correspond to the linear velocity and the last three rows correspond to
    the angular velocity, expressed in world frame coordinates
    """
    fk = FK()
    joint_positions, T0e = fk.forward(q_in)
    z_cap = fk.get_axis_of_rotation(q_in)
    J = np.zeros((6, 7))
    J_v= np.zeros((7,3))
    J_omega= np.zeros((7,3))
    O_end_0=joint_positions[7]

    for i in range(0,7):
        J_v[i]=np.cross(z_cap[i],(O_end_0-joint_positions[i]))

    J_omega=z_cap

    J_=np.hstack((J_v,J_omega))
    # print(np.round(z_cap,3))
    # print(joint_positions)


    ## STUDENT CODE GOES HERE
    J=np.transpose(J_)
    return J

if __name__ == '__main__':
    q= np.array([0, 0, 0, -np.pi/2, 0, np.pi/2, np.pi/4])
    print(np.round(calcJacobian(q),3))
```

$\vec{O}^0_{i-1}$ $\qquad$ $\hat{z}^0_{i-1} \times (\vec{O}^0_n - \vec{O}^0_{i-1})$

$J_v = \hat{z}_{i-1}$

We have Jacobian $= \begin{bmatrix} J_v \\ J_\omega \end{bmatrix}_{6 \times 7}$ matrix.

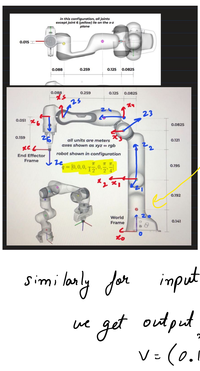
$$\underbrace{\begin{bmatrix} V \\ \omega \end{bmatrix}}_{6 \times 1} = J(q)\, \dot{q} \rightarrow \text{Velocity FK}$$

2)

```python
FK_velocity.py M ×
lib > FK_velocity.py > ...
import numpy as np
from calcJacobian import calcJacobian

def FK_velocity(q_in, dq):
    """
    :param q_in: 1 x 7 vector corresponding to the robot's current configuration.
    :param dq: 1 x 7 vector corresponding to the joint velocities.
    :return:
    velocity - 6 x 1 vector corresponding to the end effector velocities.
    """


    ## STUDENT CODE GOES HERE
    J=np.round(calcJacobian(q_in),3)
    velocity = np.zeros((6, 1))
    velocity= np.dot(J,dq)

    return velocity

q= np.array([0, 0, 0, -np.pi/2, 0, np.pi/2, np.pi/4])
dq=np.array([1,0,0,0,0,0,0])
print(np.round(FK_velocity(q,dq),3))
```

input of constant 1 velocity at joint 1

```
PROBLEMS   OUTPUT   TERMINAL   PORTS

TERMINAL
(code) tejdTejendras-MacBook-Pro meam520_robotics % /Users/tej/Applications/Anaconda/anaconda3/envs/code/bin/pyth
obotics/lib/FK_velocity.py
[0.   0.554 0.   0.   1.   ]
(code) tejdTejendras-MacBook-Pro meam520_robotics %
```



so for this orientation & $\dot{\theta}_1 = 1$ we get what is expected the end-effector velocity will only be in +x y axis & ω in z with value 1.

we can also confirm $V = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \times$ (End effector coordinates)

$= 0.554$.

similarly for input $\dot{q}_1 = [0\ 1\ 0\ 0\ 0\ 0\ 0]$

we get output

$$V = (\overset{x}{0.189}, \overset{y}{0}, \overset{z}{-0.554})$$

$$\omega = (0, 1, 0)$$

which is what we expect.

3) when calculating IK-velocity we give end-effector velocity and we get joint velocity.

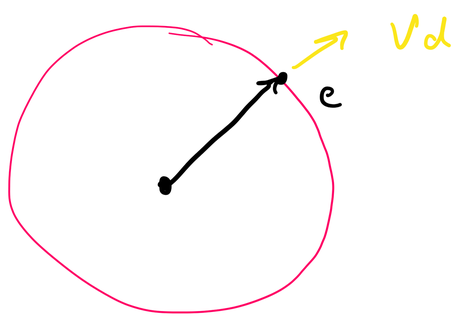
So for No sol^n →

mathematically if rank J < rank [J|ε]

as this means that velocity is not in terms of Jacobians col^n.

Analytically :

let say our robot is at end point of reachable workspace. and we give velocity of end effector to extend more. which is not



if I give Vd as shown its not possible so No sol^n.

So for Multiple / Infinite sol^n

If rank J = rank [J|ε] so sol^n exists.

but as $n > 6$ so we have multiple sol^n

Geometrically when two Z axis are collinear there will be infinite sol^n. which is case most of the time in Panda robot

Unique Sol^n :

This will not be a general case as it is 7DOF so it will mostly have multiple sol^n but if there are some external constraints like environmental constraint then the bot might have one-sol^n. but in general it will have multiple sol^n only.

4) → Find velocity FK & IK using T_{i-1}^{0} from calculateFK.py.

→ Then on hardware giving velocity to each joints & seeing if everything works fine.

→ checking for singularities & how to go with many sol^n.

→ Also as we have environmental constraints so can find joint angles limitation & develop dq such that it is not outside it so that bot would not collide with other objects like wall / table.