Import Depedencies

In [1]:

```
import cv2
import mediapipe as mp
import numpy as np
import time
import math
import numpy as np

# Dependecines for CSV
import csv
import os

# for rotation matrix
from scipy.spatial.transform import Rotation as R
```

Import CSV files

In [2]:

```
rows = ['Frame','t1_pitch','t1_yaw','t1_roll']
with open('coords_yaw.csv', mode='w', newline='') as f:
    csv_writer = csv.writer(f, delimiter=',', quotechar='"', quoting=csv.QUOTE_MINIMAL)
    csv_writer.writerow(rows)
```

The Implementation

In [3]:

```
mp_face_mesh = mp.solutions.face_mesh
face_mesh = mp_face_mesh.FaceMesh(min_detection_confidence=0.5, min_tracking_confidence=0.5

mp_drawing = mp.solutions.drawing_utils

drawing_spec = mp_drawing.DrawingSpec(thickness=1, circle_radius=1)
frame = 1

cap = cv2.VideoCapture('v1_yaw.mp4')

while cap.isOpened():
    success, image = cap.read()

    start = time.time()

    # Flip the image horizontally for a later selfie-view display
    # Also convert the color space from BGR to RGB
    image = cv2.cvtColor(cv2.flip(image, 1), cv2.COLOR_BGR2RGB)

    # To improve performance
    image.flags.writeable = False

    # Get the result
    results = face_mesh.process(image)

    # To improve performance
    image.flags.writeable = True

    # Convert the color space from RGB to BGR
    image = cv2.cvtColor(image, cv2.COLOR_RGB2BGR)

    img_h, img_w, img_c = image.shape
    face_3d = []
    face_2d = []

    if results.multi_face_landmarks:
        for face_landmarks in results.multi_face_landmarks:
            for idx, lm in enumerate(face_landmarks.landmark):
                if idx == 33 or idx == 263 or idx == 1 or idx == 61 or idx == 291 or idx ==
                    if idx == 1:
                        nose_2d = (lm.x * img_w, lm.y * img_h)
                        nose_3d = (lm.x * img_w, lm.y * img_h, lm.z * 3000)

                    x, y = int(lm.x * img_w), int(lm.y * img_h)

                    # Get the 2D Coordinates
                    face_2d.append([x, y])

                    # Get the 3D Coordinates
                    face_3d.append([x, y, lm.z])

            # Convert it to the NumPy array
            face_2d = np.array(face_2d, dtype=np.float64)

            # Convert it to the NumPy array
            face_3d = np.array(face_3d, dtype=np.float64)

            # The camera matrix
            focal_length = 1 * img_w
```

```
            cam_matrix = np.array([ [focal_length, 0, img_h / 2],
                                    [0, focal_length, img_w / 2],
                                    [0, 0, 1]])

            # The distortion parameters
            dist_matrix = np.zeros((4, 1), dtype=np.float64)

            # Solve PnP
            success, rot_vec, trans_vec = cv2.solvePnP(face_3d, face_2d, cam_matrix, dist_m

            # Get rotational matrix
            rmat, jac = cv2.Rodrigues(rot_vec)


            r = R.from_matrix(rmat)
            #r = R.from_rotvec(np.transpose(rot_vec))

            angles = r.as_euler('zyx', degrees=True)
            angles = angles*1000
            print(angles)




            # Add the text on the image
            #cv2.putText(image, text, (20, 50), cv2.FONT_HERSHEY_SIMPLEX, 2, (0, 255, 0), 2
            cv2.putText(image, "t1_p: " + str(angles[2]), (500, 50), cv2.FONT_HERSHEY_SIMPL
            cv2.putText(image, "t1_y: " + str(angles[1]), (500, 100), cv2.FONT_HERSHEY_SIMP
            cv2.putText(image, "t1_r: " + str(angles[0]), (500, 150), cv2.FONT_HERSHEY_SIMP



        end = time.time()
        totalTime = end - start

        row = [frame,angles[2],angles[1],angles[0]]
        with open('coords_yaw.csv', mode='a', newline='') as f:
            csv_writer = csv.writer(f, delimiter=',', quotechar='"', quoting=csv.QUOTE_MINI
            csv_writer.writerow(row)

        mp_drawing.draw_landmarks(
                    image=image,
                    landmark_list=face_landmarks,
                    connections=mp_face_mesh.FACEMESH_TESSELATION,
                    landmark_drawing_spec=drawing_spec,
                    connection_drawing_spec=drawing_spec)


    cv2.imshow('Head Pose Estimation', image)
    frame = frame + 1
    if cv2.waitKey(5) & 0xFF == 27:
        break


cap.release()
```

```
[ 2.47385573 10.71188651 -5.76196287]
```

```
[ 2.42471144 10.42103911 -6.93739936]
[ 2.47585334 10.34816694 -7.71182636]
[ 2.40404254  9.69416137 -7.91277923]
[ 2.35838295  9.37231558 -8.1194592 ]
[ 2.39058009  9.34571708 -8.28078823]
[ 2.36765312  9.02419611 -8.93885336]
[ 2.31328231  9.14663682 -7.80225891]
[ 2.31090248  8.59067612 -8.85926294]
[ 2.30979367  8.72160978 -8.77393057]
[ 2.33803462  9.55676836 -7.86995715]
[ 2.35466344  9.60270673 -8.3168879 ]
[ 2.33030501  9.04983096 -8.08380311]
[ 2.39056891  9.90536818 -8.17550791]
[ 2.49734745 10.50159098 -9.01554481]
[ 2.67306961 12.17936759 -8.51117571]
[ 2.88907624 14.88225594 -7.08242376]
[ 3.19703723 18.02386133 -6.85383622]
[ 3.50045557 20.9376752  -5.09340339]
```

In [ ]:

```
print(rmat)
```

In [ ]:

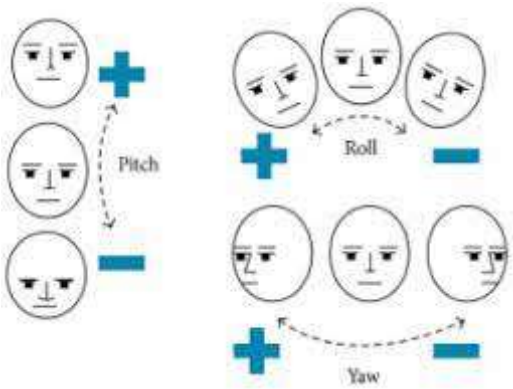
```
print(np.transpose(rot_vec))
```

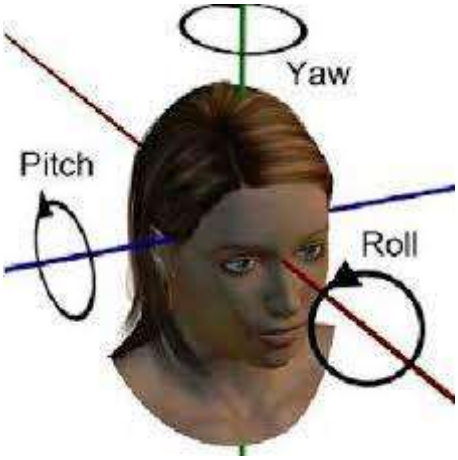
In [ ]:

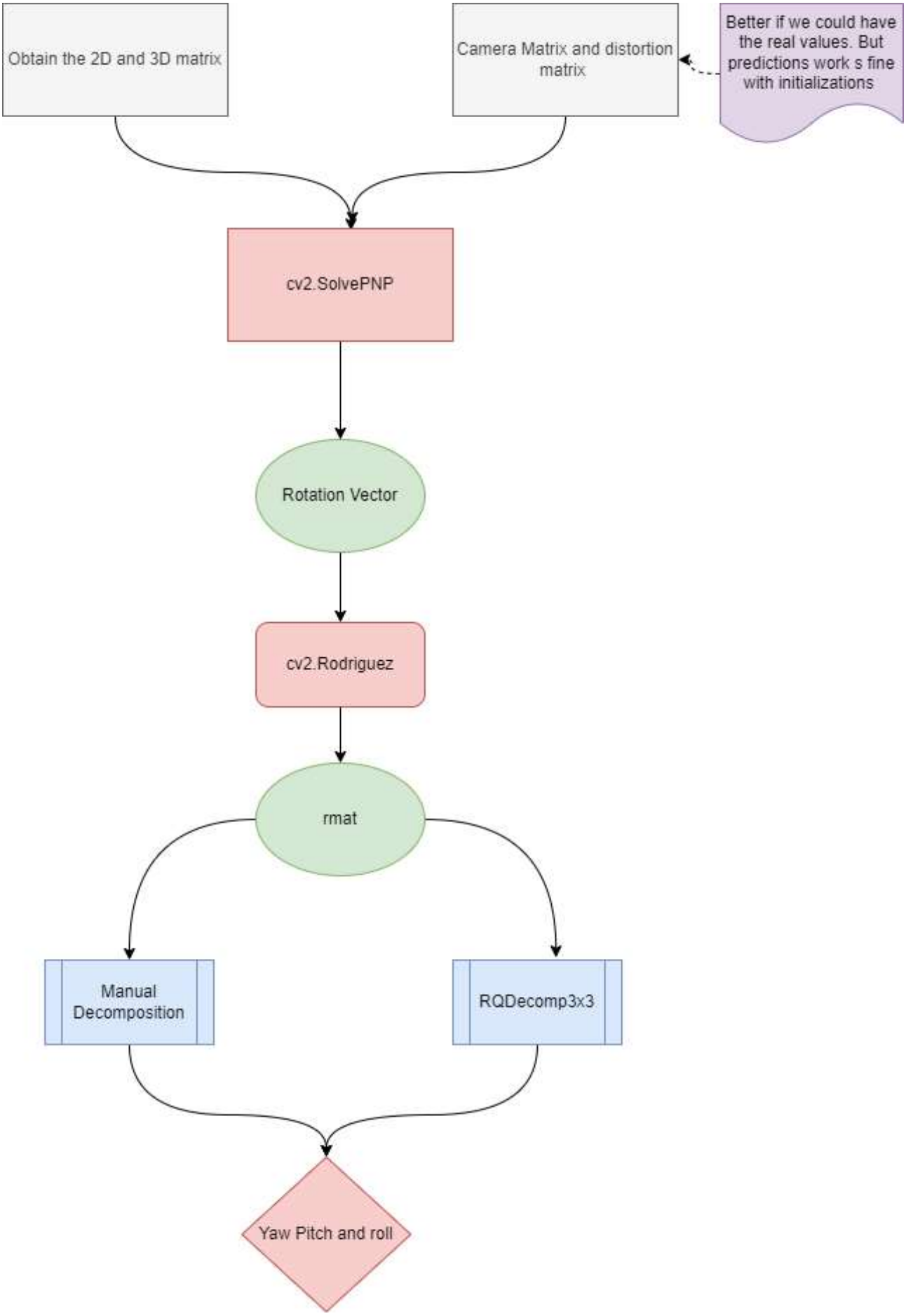
```

```

In [ ]:

```

```




Yaw
Pitch
Roll


Obtain the 2D and 3D matrix
Camera Matrix and distortion matrix
Better if we could have the real values. But predictions work s fine with initializations
cv2.SolvePNP
Rotation Vector
cv2.Rodriguez
rmat
Manual Decomposition
RQDecomp3x3
Yaw Pitch and roll

Proper Euler angles	Tait-Bryan angles
$X_1Z_2X_3 = \begin{bmatrix} c_2 & -c_3s_2 & s_2s_3 \\ c_1s_2 & c_1c_2c_3 - s_1s_3 & -c_3s_1 - c_1c_2s_3 \\ s_1s_2 & c_1s_3 + c_2c_3s_1 & c_1c_3 - c_2s_1s_3 \end{bmatrix}$	$X_1Z_2Y_3 = \begin{bmatrix} c_2c_3 & -s_2 & c_2s_3 \\ s_1s_3 + c_1c_3s_2 & c_1c_2 & c_1s_2s_3 - c_3s_1 \\ c_3s_1s_2 - c_1s_3 & c_2s_1 & c_1c_3 + s_1s_2s_3 \end{bmatrix}$
$X_1Y_2X_3 = \begin{bmatrix} c_2 & s_2s_3 & c_3s_2 \\ s_1s_2 & c_1c_3 - c_2s_1s_3 & -c_1s_3 - c_2c_3s_1 \\ -c_1s_2 & c_3s_1 + c_1c_2s_3 & c_1c_2c_3 - s_1s_3 \end{bmatrix}$	$X_1Y_2Z_3 = \begin{bmatrix} c_2c_3 & -c_2s_3 & s_2 \\ c_1s_3 + c_3s_1s_2 & c_1c_3 - s_1s_2s_3 & -c_2s_1 \\ s_1s_3 - c_1c_3s_2 & c_3s_1 + c_1s_2s_3 & c_1c_2 \end{bmatrix}$
$Y_1X_2Y_3 = \begin{bmatrix} c_1c_3 - c_2s_1s_3 & s_1s_2 & c_1s_3 + c_2c_3s_1 \\ s_2s_3 & c_2 & -c_3s_2 \\ -c_3s_1 - c_1c_2s_3 & c_1s_2 & c_1c_2c_3 - s_1s_3 \end{bmatrix}$	$Y_1X_2Z_3 = \begin{bmatrix} c_1c_3 + s_1s_2s_3 & c_3s_1s_2 - c_1s_3 & c_2s_1 \\ c_2s_3 & c_2c_3 & -s_2 \\ c_1s_2s_3 - c_3s_1 & c_1c_3s_2 + s_1s_3 & c_1c_2 \end{bmatrix}$
$Y_1Z_2Y_3 = \begin{bmatrix} c_1c_2c_3 - s_1s_3 & -c_1s_2 & c_3s_1 + c_1c_2s_3 \\ c_3s_2 & c_2 & s_2s_3 \\ -c_1s_3 - c_2c_3s_1 & s_1s_2 & c_1c_3 - c_2s_1s_3 \end{bmatrix}$	$Y_1Z_2X_3 = \begin{bmatrix} c_1c_2 & s_1s_3 - c_1c_3s_2 & c_3s_1 + c_1s_2s_3 \\ s_2 & c_2c_3 & -c_2s_3 \\ -c_2s_1 & c_1s_3 + c_3s_1s_2 & c_1c_3 - s_1s_2s_3 \end{bmatrix}$
$Z_1Y_2Z_3 = \begin{bmatrix} c_1c_2c_3 - s_1s_3 & -c_3s_1 - c_1c_2s_3 & c_1s_2 \\ c_1s_3 + c_2c_3s_1 & c_1c_3 - c_2s_1s_3 & s_1s_2 \\ -c_3s_2 & s_2s_3 & c_2 \end{bmatrix}$	$Z_1Y_2X_3 = \begin{bmatrix} c_1c_2 & c_1s_2s_3 - c_3s_1 & s_1s_3 + c_1c_3s_2 \\ c_2s_1 & c_1c_3 + s_1s_2s_3 & c_3s_1s_2 - c_1s_3 \\ -s_2 & c_2s_3 & c_2c_3 \end{bmatrix}$
$Z_1X_2Z_3 = \begin{bmatrix} c_1c_3 - c_2s_1s_3 & -c_1s_3 - c_2c_3s_1 & s_1s_2 \\ c_3s_1 + c_1c_2s_3 & c_1c_2c_3 - s_1s_3 & -c_1s_2 \\ s_2s_3 & c_3s_2 & c_2 \end{bmatrix}$	$Z_1X_2Y_3 = \begin{bmatrix} c_1c_3 - s_1s_2s_3 & -c_2s_1 & c_1s_3 + c_3s_1s_2 \\ c_3s_1 + c_1s_2s_3 & c_1c_2 & s_1s_3 - c_1c_3s_2 \\ -c_2s_3 & s_2 & c_2c_3 \end{bmatrix}$

Proper euler angles

https://upload.wikimedia.org/wikipedia/commons/thumb/9/9e/EulerProjections2.svg/300px-EulerProjections2.svg.png (https://upload.wikimedia.org/wikipedia/commons/thumb/9/9e/EulerProjections2.svg/300px-EulerProjections2.svg.png)

Proper Euler angles [edit]

Assuming a frame with unit vectors (*X*, *Y*, *Z*) given by their coordinates as in the main diagram, it can be seen that:

$$\cos(\beta) = Z_3.$$

And, since

$$\sin^2 x = 1 - \cos^2 x,$$

for $0 < x < \pi$ we have

$$\sin(\beta) = \sqrt{1 - Z_3^2}.$$

As Z_2 is the double projection of a unitary vector,

$$\cos(\alpha) \cdot \sin(\beta) = -Z_2,$$
$$\cos(\alpha) = -Z_2 / \sqrt{1 - Z_3^2}.$$

There is a similar construction for Y_3, projecting it first over the plane defined by the axis *z* and the line of nodes. As the angle between the planes is $\pi/2 - \beta$ and $\cos(\pi/2 - \beta) = \sin(\beta)$, this leads to:

$$\sin(\beta) \cdot \cos(\gamma) = Y_3,$$
$$\cos(\gamma) = Y_3 / \sqrt{1 - Z_3^2},$$

and finally, using the inverse cosine function,

$$\alpha = \arccos\left(-Z_2 / \sqrt{1 - Z_3^2}\right),$$
$$\beta = \arccos(Z_3),$$
$$\gamma = \arccos\left(Y_3 / \sqrt{1 - Z_3^2}\right).$$

Proper Euler angles Tait-Bryan angles {\displaystyle X_{1}Z_{2}X_{3}}{\displaystyle X_{1}Z_{2}X_{3}} {\displaystyle {

$$\alpha = \arctan(\frac{R_{31}}{R_{21}})$$
$$\beta = \arctan(\frac{\sqrt{1-R_{11}^2}}{R_{11}})$$
$$\gamma = \arctan(\frac{R_{13}}{-R_{12}})$$

}}{\displaystyle {

$$\alpha = \arctan(\frac{R_{31}}{R_{21}})$$
$$\beta = \arctan(\frac{\sqrt{1-R_{11}^2}}{R_{11}})$$
$$\gamma = \arctan(\frac{R_{13}}{-R_{12}})$$

}} {\displaystyle X_{1}Z_{2}Y_{3}}{\displaystyle X_{1}Z_{2}Y_{3}} {\displaystyle {

$$\alpha = \arctan(\frac{R_{32}}{R_{22}})$$
$$\beta = \arctan(\frac{-R_{12}}{\sqrt{1-R_{12}^2}})$$
$$\gamma = \arctan(\frac{R_{13}}{R_{11}})$$

}}{\displaystyle {

$$\alpha = \arctan(\frac{R_{32}}{R_{22}})$$
$$\beta = \arctan(\frac{-R_{12}}{\sqrt{1-R_{12}^2}})$$
$$\gamma = \arctan(\frac{R_{13}}{R_{11}})$$

}} {\displaystyle X_{1}Y_{2}X_{3}}{\displaystyle X_{1}Y_{2}X_{3}} {\displaystyle {

$$\alpha = \arctan(\frac{R_{21}}{-R_{31}})$$
$$\beta = \arctan(\frac{\sqrt{1-R_{11}^2}}{R_{11}})$$
$$\gamma = \arctan(\frac{R_{12}}{R_{13}})$$

}}{\displaystyle {

$$\alpha = \arctan(\frac{R_{21}}{-R_{31}})$$
$$\beta = \arctan(\frac{\sqrt{1-R_{11}^2}}{R_{11}})$$
$$\gamma = \arctan(\frac{R_{12}}{R_{13}})$$

}} {\displaystyle X_{1}Y_{2}Z_{3}}{\displaystyle X_{1}Y_{2}Z_{3}} {\displaystyle {

$$\alpha = \arctan(\frac{-R_{23}}{R_{33}})$$
$$\beta = \arctan(\frac{R_{13}}{\sqrt{1-R_{13}^2}})$$
$$\gamma = \arctan(\frac{-R_{12}}{R_{11}})$$

}}{\displaystyle {

$$\alpha = \arctan(\frac{-R_{23}}{R_{33}})$$
$$\beta = \arctan(\frac{R_{13}}{\sqrt{1-R_{13}^2}})$$
$$\gamma = \arctan(\frac{-R_{12}}{R_{11}})$$

}} {\displaystyle Y_{1}X_{2}Y_{3}}{\displaystyle Y_{1}X_{2}Y_{3}} {\displaystyle {

$$\alpha = \arctan(\frac{R_{12}}{R_{32}})$$
$$\beta = \arctan(\frac{\sqrt{1-R_{22}^2}}{R_{22}})$$
$$\gamma = \arctan(\frac{R_{21}}{-R_{23}})$$

}}{\displaystyle {

$$\alpha = \arctan(\frac{R_{12}}{R_{32}})$$
$$\beta = \arctan(\frac{\sqrt{1-R_{22}^2}}{R_{22}})$$
$$\gamma = \arctan(\frac{R_{21}}{-R_{23}})$$

}} {\displaystyle Y_{1}X_{2}Z_{3}}{\displaystyle Y_{1}X_{2}Z_{3}} {\displaystyle {

$$\alpha = \arctan(\frac{R_{13}}{R_{33}})$$
$$\beta = \arctan(\frac{-R_{23}}{\sqrt{1-R_{23}^2}})$$
$$\gamma = \arctan(\frac{R_{21}}{R_{22}})$$

}}{\displaystyle {

$$\alpha = \arctan(\frac{R_{13}}{R_{33}})$$
$$\beta = \arctan(\frac{-R_{23}}{\sqrt{1-R_{23}^2}})$$
$$\gamma = \arctan(\frac{R_{21}}{R_{22}})$$

}} {\displaystyle Y_{1}Z_{2}Y_{3}}{\displaystyle Y_{1}Z_{2}Y_{3}} {\displaystyle {

$$\alpha = \arctan(\frac{R_{32}}{-R_{12}})$$
$$\beta = \arctan(\frac{\sqrt{1-R_{22}^2}}{R_{22}})$$
$$\gamma = \arctan(\frac{R_{23}}{R_{21}})$$

}}{\displaystyle {

$$\alpha = \arctan(\frac{R_{32}}{-R_{12}})$$
$$\beta = \arctan(\frac{\sqrt{1-R_{22}^2}}{R_{22}})$$
$$\gamma = \arctan(\frac{R_{23}}{R_{21}})$$

}} {\displaystyle Y_{1}Z_{2}X_{3}}{\displaystyle Y_{1}Z_{2}X_{3}} {\displaystyle {

$$\alpha = \arctan(\frac{-R_{31}}{R_{11}})$$
$$\beta = \arctan(\frac{R_{21}}{\sqrt{1-R_{21}^2}})$$
$$\gamma = \arctan(\frac{-R_{23}}{R_{22}})$$

}}{\displaystyle {

$$\alpha = \arctan(\frac{-R_{31}}{R_{11}})$$
$$\beta = \arctan(\frac{R_{21}}{\sqrt{1-R_{21}^2}})$$
$$\gamma = \arctan(\frac{-R_{23}}{R_{22}})$$

}} {\displaystyle Z_{1}Y_{2}Z_{3}}{\displaystyle Z_{1}Y_{2}Z_{3}} {\displaystyle {

$$\alpha = \arctan(\frac{R_{23}}{R_{13}})$$
$$\beta = \arctan(\frac{\sqrt{1-R_{33}^2}}{R_{33}})$$
$$\gamma = \arctan(\frac{R_{32}}{-R_{31}})$$

}}{\displaystyle {

$$\alpha = \arctan(\frac{R_{23}}{R_{13}})$$
$$\beta = \arctan(\frac{\sqrt{1-R_{33}^2}}{R_{33}})$$
$$\gamma = \arctan(\frac{R_{32}}{-R_{31}})$$

}} {\displaystyle Z_{1}Y_{2}X_{3}}{\displaystyle Z_{1}Y_{2}X_{3}} {\displaystyle {

$$\alpha = \arctan(\frac{R_{21}}{R_{11}})$$
$$\beta = \arctan(\frac{-R_{31}}{\sqrt{1 - R_{31}^2}})$$
$$\gamma = \arctan(\frac{R_{32}}{R_{33}})$$

}}{\displaystyle {

$$\alpha = \arctan(\frac{R_{21}}{R_{11}})$$
$$\beta = \arctan(\frac{-R_{31}}{\sqrt{1 - R_{31}^2}})$$
$$\gamma = \arctan(\frac{R_{32}}{R_{33}})$$

}} {\displaystyle Z_{1}X_{2}Z_{3}}Z_{1}X_{2}Z_{3} {\displaystyle {

$$\alpha = \arctan(\frac{R_{13}}{-R_{23}})$$
$$\beta = \arctan(\frac{\sqrt{1 - R_{33}^2}}{R_{33}})$$
$$\gamma = \arctan(\frac{R_{31}}{R_{32}})$$

}}{\displaystyle {

$$\alpha = \arctan(\frac{R_{13}}{-R_{23}})$$
$$\beta = \arctan(\frac{\sqrt{1 - R_{33}^2}}{R_{33}})$$
$$\gamma = \arctan(\frac{R_{31}}{R_{32}})$$

}} {\displaystyle Z_{1}X_{2}Y_{3}}{\displaystyle Z_{1}X_{2}Y_{3}} {\displaystyle {

$$\alpha = \arctan(\frac{-R_{12}}{R_{22}})$$
$$\beta = \arctan(\frac{R_{32}}{\sqrt{1 - R_{32}^2}})$$
$$\gamma = \arctan(\frac{-R_{31}}{R_{33}})$$

}}{\displaystyle {

$$\alpha = \arctan(\frac{-R_{12}}{R_{22}})$$
$$\beta = \arctan(\frac{R_{32}}{\sqrt{1 - R_{32}^2}})$$
$$\gamma = \arctan(\frac{-R_{31}}{R_{33}})$$

}}

#include <opencv2/calib3d.hpp>

Converts a rotation matrix to a rotation vector or vice versa.

Parameters src Input rotation vector (3x1 or 1x3) or rotation matrix (3x3). dst Output rotation matrix (3x3) or rotation vector (3x1 or 1x3), respectively. jacobian Optional output Jacobian matrix, 3x9 or 9x3, which is a matrix of partial derivatives of the output array components with respect to the input array components. $\theta \leftarrow norm(r) r \leftarrow r/\theta R = \cos(\theta) I + (1 - \cos\theta) r r^T + \sin(\theta) \begin{bmatrix} 0 & -r_z & r_y \\ r_z & 0 & -r_x \\ -r_y & r_x & 0 \end{bmatrix}$ Inverse transformation can be also done easily, since

$\sin(\theta) \begin{bmatrix} 0 & -r_z & r_y \\ r_z & 0 & -r_x \\ -r_y & r_x & 0 \end{bmatrix} = R - R^T$ A rotation vector is a convenient and most compact representation of a rotation matrix (since any rotation matrix has just 3 degrees of freedom). The representation is used in the global 3D geometry optimization procedures like calibrateCamera, stereoCalibrate, or solvePnP .

Note More information about the computation of the derivative of a 3D rotation matrix with respect to its exponential coordinate can be found in: A Compact Formula for the Derivative of a 3-D Rotation in Exponential Coordinates, Guillermo Gallego, Anthony J. Yezzi [80] Useful information on SE(3) and Lie Groups can be found in: A tutorial on SE(3) transformation parameterizations and on-manifold optimization, Jose-Luis Blanco [23] Lie Groups for 2D and 3D Transformation, Ethan Eade [62] A micro Lie theory for state estimation in robotics, Joan Solà, Jérémie Deray, Dinesh Atchuthan [196]

In [ ]:

In [ ]:

In [ ]:

In [ ]: