```python
"""
CIVE 6374 – Optical Imaging Metrology
Professor: Dr. Craig Glennie
Author: Joshua Genova
Lab # 1
Description: Similarity, Affine and Projective Transformations
Deadline: February 22, 2023 10:00 AM
"""
import time
import numpy as np
import math
import matplotlib.pyplot as plt

def similarity_transform(xc, yc, xf, yf):
    """
    Similarity Transform
    Find translation (delta x, delta y)
    Find scale (lambda)
    Find rotation (theta)
    """
    print('-'*79)
    print("Similarity Transform")
    n = len(xc)
    mat_size = 2*n

    # create l-vector
    l_mat = np.zeros(shape=(mat_size,1))
    idx = 0
    for i in range(0, mat_size, 2):
        l_mat[i] = xf[idx]
        l_mat[i+1] = yf[idx]
        idx += 1

    # create A-matrix
    A_mat = np.zeros(shape=(mat_size,4))
    idx = 0
    for i in range(0, mat_size, 2):
        A_mat[i] = [xc[idx], -yc[idx], 1, 0]
        A_mat[i+1] = [yc[idx], xc[idx], 0, 1]
        idx +=1

    # calculate the unknowns x_hat
    x_hat = np.dot(np.dot(np.linalg.inv(np.dot(np.transpose(A_mat),A_mat)), np.transpose(A_mat)),
l_mat)
    A, B, Dx, Dy = float(x_hat[0]), float(x_hat[1]), float(x_hat[2]), float(x_hat[3])
    scale = math.sqrt(A**2 + B**2)
    theta = math.atan(B/A)

    # print linear parameters
    print('-'*79)
    print(f'A: {A}')
    print(f'B: {B}')

    # print non-linear parameters
    print('-'*79)
    print(f'delta X: {Dx}')
```

```
    print(f'delta Y: {Dy}')
    print(f'scale: {scale}')
    print(f'theta: {theta}')

    # create residuals vector
    v = np.dot(A_mat, x_hat) - l_mat
    v_mat = np.zeros(shape=(n,2))
    idx = 0
    x_rms = 0
    y_rms = 0

    # calculate rms
    for i in range(0, mat_size, 2):
        v_mat[idx] = [v[i], v[i+1]]
        x_rms = x_rms + v[i]**2
        y_rms = y_rms + v[i+1]**2
        idx += 1
    x_rms = math.sqrt((1/n)*x_rms)
    y_rms = math.sqrt((1/n)*y_rms)
    print('-'*79)
    print('Residuals: ')
    print(v_mat)
    print(f'x RMS {x_rms}')
    print(f'y RMS {y_rms}')
    print('-'*79)

    # quiver plot of vals and residuals
    fig, ax = plt.subplots(figsize = (5, 5))
    for i in range(n):
        ax.quiver(xc[i], yc[i], v_mat[i,0], v_mat[i,1])
    ax.set_xlabel('x(mm)')
    ax.set_ylabel('y(mm)')
    ax.set_title('Image Point Residual Plot')

    plt.show()

    return

def affine_transform(xc, yc, xf, yf):
    """
    Affine Transform
    Find translation (delta x, delta y)
    Find rotation (theta)
    Find scale (Sx, Sy)
    Find non-orthogonality of comparator axes (delta)
    """
    print('-'*79)
    print('Affine Transform')
    n = len(xc)
    mat_size = 2*n

    # create l-vector
    l_mat = np.zeros(shape=(mat_size,1))
    idx = 0
    for i in range(0, mat_size, 2):
        l_mat[i] = xf[idx]
```

```
        l_mat[i+1] = yf[idx]
        idx += 1

    # create A-matrix
    A_mat = np.zeros(shape=(mat_size,6))
    idx = 0
    for i in range(0, mat_size, 2):
        A_mat[i] = [xc[idx], yc[idx], 1, 0, 0 ,0]
        A_mat[i+1] = [0, 0, 0, xc[idx], yc[idx], 1]
        idx +=1

    # calculate the unknowns x_hat
    x_hat = np.dot(np.dot(np.linalg.inv(np.dot(np.transpose(A_mat),A_mat)), np.transpose(A_mat)),
l_mat)
    A, B, Dx, C, D, Dy = float(x_hat[0]), float(x_hat[1]), float(x_hat[2]), float(x_hat[3]),
float(x_hat[4]), float(x_hat[5])
    theta = math.atan(C/A)
    Sx = math.sqrt(A**2 + C**2)
    Sy = math.sqrt(B**2 + D**2)
    delta =math.atan((A*B + C*D)/(A*D - B*C))

    # print linear parameters
    print('-'*79)
    print(f'A: {A}')
    print(f'B: {B}')
    print(f'C: {C}')
    print(f'D: {D}')

    # print non-linear parameters
    print('-'*79)
    print(f'delta X: {Dx}')
    print(f'delta Y: {Dy}')
    print(f'theta: {theta}')
    print(f'Scale X: {Sx}')
    print(f'Scale Y: {Sy}')
    print(f'delta: {delta}')

    # create residuals vector
    v = np.dot(A_mat, x_hat) - l_mat
    v_mat = np.zeros(shape=(n,2))
    idx = 0
    x_rms = 0
    y_rms = 0

    # calculate rms
    for i in range(0, mat_size, 2):
        v_mat[idx] = [v[i], v[i+1]]
        x_rms = x_rms + v[i]**2
        y_rms = y_rms + v[i+1]**2
        idx += 1
    x_rms = math.sqrt((1/n)*x_rms)
    y_rms = math.sqrt((1/n)*y_rms)
    print('-'*79)
    print('Residuals: ')
    print(v_mat)
    print(f'x RMS {x_rms}')
    print(f'y RMS {y_rms}')
```

```
    print('-'*79)

    # quiver plot of vals and residuals
    fig, ax = plt.subplots(figsize = (5, 5))
    for i in range(n):
        ax.quiver(xc[i], yc[i], v_mat[i,0], v_mat[i,1])
    ax.set_xlabel('x(mm)')
    ax.set_ylabel('y(mm)')
    ax.set_title('Image Point Residual Plot')

    plt.show()

    return

def projective_trans(xc, yc, xf, yf):
    """
    Projective Transform
    Find translation (delta x, delta y)
    Find differential scale (Sx, Sy)
    Find non-orthogonality (delta)
    Find rotation (theta)
    Find out of plane inclination (2 parameters)
    """
    print('-'*79)
    print("Projective Transform")
    n = len(xc)
    mat_size = 2*n

    # create l-vector
    l_mat = np.zeros(shape=(mat_size,1))
    idx = 0
    for i in range(0, mat_size, 2):
        l_mat[i] = xf[idx]
        l_mat[i+1] = yf[idx]
        idx += 1

    # create A-matrix
    A_mat = np.zeros(shape=(mat_size,8))
    idx = 0
    for i in range(0, mat_size, 2):
        A_mat[i] = [xc[idx], yc[idx], 1, 0, 0 ,0, -xf[idx]*xc[idx], -xf[idx]*yc[idx]]
        A_mat[i+1] = [0, 0, 0, xc[idx], yc[idx], 1, -yf[idx]*xc[idx], -yf[idx]*yc[idx]]
        idx +=1

    # calculate the unknowns x_hat
    x_hat = np.dot(np.dot(np.linalg.inv(np.dot(np.transpose(A_mat),A_mat)), np.transpose(A_mat)),
l_mat)
    A, B, Dx, C, D, Dy, plane_incline1, plane_incline2 = float(x_hat[0]), float(x_hat[1]),
float(x_hat[2]), float(x_hat[3]), float(x_hat[4]), float(x_hat[5]), float(x_hat[6]),
float(x_hat[7])
    theta = math.atan(C/A)
    Sx = math.sqrt(A**2 + C**2)
    Sy = math.sqrt(B**2 + D**2)
    delta =math.atan((A*B + C*D)/(A*D - B*C))

    # print linear parameters
    print('-'*79)
```

```
    print(f'A: {A}')
    print(f'B: {B}')
    print(f'C: {C}')
    print(f'D: {D}')

    # print non-linear parameters
    print('-'*79)
    print(f'delta X: {Dx}')
    print(f'delta Y: {Dy}')
    print(f'theta: {theta}')
    print(f'Scale X: {Sx}')
    print(f'Scale Y: {Sy}')
    print(f'delta: {delta}')
    print(f'out of plane inclination: {plane_incline1, plane_incline2}')

    # create residuals vector
    v = np.dot(A_mat, x_hat) - l_mat
    v_mat = np.zeros(shape=(n,2))
    idx = 0
    x_rms = 0
    y_rms = 0

    # calculate rms
    for i in range(0, mat_size, 2):
        v_mat[idx] = [v[i], v[i+1]]
        x_rms = x_rms + v[i]**2
        y_rms = y_rms + v[i+1]**2
        idx += 1
    x_rms = math.sqrt((1/n)*x_rms)
    y_rms = math.sqrt((1/n)*y_rms)
    print('-'*79)
    print('Residuals: ')
    print(v_mat)
    print(f'x RMS {x_rms}')
    print(f'y RMS {y_rms}')
    print('-'*79)

    # quiver plot of vals and residuals
    fig, ax = plt.subplots(figsize = (5, 5))
    for i in range(n):
        ax.quiver(xc[i], yc[i], v_mat[i,0], v_mat[i,1])
    ax.set_xlabel('x(mm)')
    ax.set_ylabel('y(mm)')
    ax.set_title('Image Point Residual Plot')

    plt.show()
    return

if __name__=="__main__":
    # test vals
    # xc = [-113.767, -43.717, 36.361, 106.408, 107.189, 37.137, -42.919, -102.968, -112.052,
-42.005, 38.051, 108.089, 108.884, 38.846, -41.208, -111.249]
    # yc = [-107.400, -108.204, -109.132, -109.923, -39.874, -39.070, -38.158, -37.446, 42.714,
41.903, 40.985, 40.189, 110.221, 111.029, 111.961, 112.759]
    # xf = [-110, -40, 40, 110, 110, 40, -40, -100, -110, -40, 40, 110, 110, 40, -40, -110]
    # yf = [-110, -110, -110, -110, -40, -40, -40, -40, 40, 40, 40, 40, 110, 110, 110, 110]
```

```
    # assignment vals
    xc = [-105.997, 106.004, -106, 106.012, -112, 112.006, 0.005, 0.002]
    yc = [-105.995, 106.008, 106.009, -105.995, 0.007, 0.007, 112.007, -111.998]

    # image 1
    xf1 = [1346, 19162, 1347, 19164, 842, 19667, 10254, 10255]
    yf1 = [-19286, -1471, -1472, -19284, -10380, -10377, -967, -19789]

    # image 2
    xf2 = [1347, 19175, 1359, 19163, 850, 19673, 10268, 10254]
    yf2 = [-19286, -1484, -1472, -19297, -10379, -10390, -973, -19796]

    similarity_transform(xc, yc, xf1, yf1)
    similarity_transform(xc, yc, xf2, yf2)
    affine_transform(xc, yc, xf1, yf1)
    affine_transform(xc, yc, xf2, yf2)
    projective_trans(xc, yc, xf1, yf1)
    projective_trans(xc, yc, xf2, yf2)
```