Lab3\Lab3.py

```python
"""
CIVE 6374 – Optical Imaging Metrology
Professor: Dr. Craig Glennie
Author: Joshua Genova
Lab # 3
Description: Relative Orientation
Deadline: March 22, 2023 10:00 AM
"""

import numpy as np
from numpy.linalg import inv, det
import math
from math import sin, cos
import matplotlib.pyplot as plt
import preLab3 as pre_lab_3

def correct_images(xc1, yc1, xc2, yc2):
    idx = 0
    correction_left = np.zeros(shape=(len(xc1),2))
    correction_right = np.zeros(shape=(len(xc2),2))
    for i in range(len(xc1)):
        # correct images 1
        xl, yl = pre_lab_3.get_left(xc1[i], yc1[i])
        correction_left[idx][0] = round(xl, 3)
        correction_left[idx][1] = round(yl, 3)

        # correct images 2
        xr, yr = pre_lab_3.get_right(xc2[i], yc2[i])
        correction_right[idx][0] = round(xr, 3)
        correction_right[idx][1] = round(yr, 3)
        idx += 1

    # print(f'Total Correction Left: \n {correction_left}\n')
    # print(f'Total Correction Right: \n {correction_right}\n')
    return correction_left, correction_right

def transform_images(xr, yr, c, omega, phi, kappa):
    rot_mat = np.array([
        [cos(phi)*cos(kappa), cos(omega)*sin(kappa)+sin(omega)*sin(phi)*cos(kappa), sin(omega)*sin(
        [-cos(phi)*sin(kappa), cos(omega)*cos(kappa)-sin(omega)*sin(phi)*sin(kappa), sin(omega)*cos
        [sin(phi), -sin(omega)*cos(phi), cos(omega)*cos(phi)]
    ])

    xr_t = np.zeros(len(xr))
    yr_t = np.zeros(len(yr))
    zr_t = np.zeros(len(xr))
    for i in range(len(xr)):
        vr = np.array([xr[i], yr[i], -c])
        xr_t[i], yr_t[i], zr_t[i] = np.dot(rot_mat.T, vr.T)
    return xr_t, yr_t, zr_t

def find_A_elems(xl, yl, c, xr, yr, zr, bx, by, bz, omega, phi, kappa):
```

```python
    dby = np.array([[0, 1, 0], [xl, yl, -c], [xr, yr, zr]])
    # print(f'dby:\n {dby}')
    dby = det(dby)
    # print(f'dby = {dby}')

    dbz = np.array([[0, 0, 1], [xl, yl, -c], [xr, yr, zr]])
    # print(f'dbz:\n {dbz}')
    dbz = det(dbz)
    # print(f'dbz = {dbz}')

    dw = np.array([[bx, by, bz], [xl, yl, -c], [0, -zr, yr]])
    # print(f'dw:\n {dw}')
    dw = det(dw)
    # print(f'dw = {dw}')

    A = -yr*sin(omega) + zr*cos(omega)
    B = xr*sin(omega)
    C = -xr*cos(omega)
    dphi = np.array([[bx, by, bz], [xl, yl, -c], [A, B, C]])
    # print(f'dphi:\n {dphi}')
    dphi = det(dphi)
    # print(f'dphi = {dphi}')

    D = -yr*cos(omega)*cos(phi) - zr*sin(omega)*cos(phi)
    E = xr*cos(omega)*cos(phi) - zr*sin(phi)
    F = xr*sin(omega)*cos(phi) + yr*sin(phi)
    dkappa = np.array([[bx, by, bz], [xl, yl, -c], [D, E, F]])
    # print(f'dkappa:\n {dkappa}')
    dkappa = det(dkappa)
    # print(f'dkappa = {dkappa}')

    return dby, dbz, dw, dphi, dkappa

def find_misclosure(xl, yl, c, xr, yr, zr, bx, by, bz):
    w = np.array([[bx, by, bz], [xl, yl, -c], [xr, yr, zr]])
    w = det(w)

    return w

def find_delta(xl, yl, c, xr, yr, zr, bx, by, bz, omega, phi, kappa):
    A_matrix = np.zeros(shape=(len(xl),5))
    w = np.zeros(shape=(len(xl), 1))
    idx = 0
    for i in range(len(xl)):
        dby, dbz, dw, dphi, dkappa = find_A_elems(xl[i], yl[i], c, xr[i], yr[i], zr[i], bx, by, bz,
        A_matrix[idx][0] = dby
        A_matrix[idx][1] = dbz
        A_matrix[idx][2] = dw
        A_matrix[idx][3] = dphi
        A_matrix[idx][4] = dkappa

        w[idx] = find_misclosure(xl[i], yl[i], c, xr[i], yr[i], zr[i], bx, by, bz)

        idx += 1
    A_matrix_trans = np.transpose(A_matrix)
    by_e, bz_e, omega_e, phi_e, kappa_e = -np.dot(np.dot(inv(np.dot(A_matrix_trans, A_matrix)), A_m
```

```python
    err = np.array([by_e[0], bz_e[0], omega_e[0], phi_e[0], kappa_e[0]])
    by = by + by_e[0]
    bz = bz + bz_e[0]
    omega = omega + omega_e[0]
    phi = phi + phi_e[0]
    kappa = kappa + kappa_e[0]

    return by, bz, omega, phi, kappa, err

def space_intersection(xl, yl, c, xr, yr, zr, bx, by, bz):
    scale = (bx*zr - bz*xr) / (xl*zr + c*xr)
    mu = (-bx*c - bz*xl) / (xl*zr + c*xr)

    model_Xl = scale*xl
    model_Yl = scale*yl
    model_Zl = -scale*c

    model_Xr = mu*xr + bx
    model_Yr = mu*yr + by
    model_Zr = mu*zr + bz

    model_L = np.transpose(np.array([model_Xl, (model_Yl + model_Yr)/2, model_Zl]))
    model_R = np.transpose(np.array([model_Xr, (model_Yl + model_Yr)/2, model_Zr]))

    pY = model_Yr - model_Yl

    return model_L, model_R, pY, scale, mu

def plot_scale(scale_left, scale_right):
    id = np.array([100, 101, 102, 103, 104, 105])
    plt.subplot(1,2,1)
    plt.bar(id, scale_left, color='darkblue', edgecolor='black', linewidth=0.1)
    plt.xlabel("Image ID", fontdict={'family':'serif','color':'black','size':10})
    plt.ylabel('Scale Factor (λ)', fontdict={'family':'serif','color':'black','size':10})
    plt.title("Left Image Scale Factor", fontdict ={'family':'serif','color':'black','size':12})

    plt.subplot(1,2,2)
    plt.bar(id, scale_right, color='darkred', edgecolor='black', linewidth=0.1)
    plt.xlabel("Image ID", fontdict={'family':'serif','color':'black','size':10})
    plt.ylabel('Scale Factor(μ)', fontdict={'family':'serif','color':'black','size':10})
    plt.title("Right Image Scale Factor", fontdict ={'family':'serif','color':'black','size':12})

    plt.show()

def find_corr_matrix(err_mat):
    A_mat = np.zeros(shape=(len(err_mat),5))
    D_mat = np.zeros(shape=(len(err_mat),5))
    S_mat = np.zeros(shape=(5,5))

    for i in range(len(err_mat)):
        A_mat[i] = err_mat[i]
    # print(A_mat)

    dby_mean = np.mean([A_mat[:,0]])
    dbz_mean = np.mean([A_mat[:,1]])
    dw_mean = np.mean([A_mat[:,2]])
```

```
    dp_mean = np.mean([A_mat[:,3]])
    dk_mean = np.mean([A_mat[:,4]])


    idx = 0
    for i in range(len(A_mat)):
        D_mat[idx][0] = A_mat[idx][0] - dby_mean
        D_mat[idx][1] = A_mat[idx][1] - dbz_mean
        D_mat[idx][2] = A_mat[idx][2] - dw_mean
        D_mat[idx][3] = A_mat[idx][3] - dp_mean
        D_mat[idx][4] = A_mat[idx][4] - dk_mean

        idx += 1

    CSSP = np.dot(D_mat.T, D_mat)
    C = CSSP*(1/(len(err_mat)-1))
    for i in range(len(C)):
        S_mat[i][i] = math.sqrt(C[i][i])
    S_inv = inv(S_mat)
    R = np.dot(S_inv, np.dot(C, S_inv))

    return R


####################################################################################################
if __name__=="__main__":

    # Image 1
    xc1 = [9460, 17400, 10059, 19158, 11844, 17842]
    yc1 = [-2292, -1661, -10883, -10412, -17253, -18028]

    # Image 2
    xc2 = [1411, 9416, 2275, 11129, 4160, 10137]
    yc2 = [-2081, -1167, -10787, -10048, -17085, -17690]

    c = 153.358 # mm

    left_images, right_images = correct_images(xc1, yc1, xc2, yc2)
    xl = left_images[:,0]
    yl = left_images[:,1]
    xr = right_images[:,0]
    yr = right_images[:,1]

    bx = 92.000
    by = 0
    bz = 0
    omega = 0
    phi = 0
    kappa = 0

    err_mat = []
    iter = 1
    while(True):
        xr_t, yr_t, zr_t = transform_images(xr, yr, c, omega, phi, kappa)
        old_by = by
        by, bz, omega, phi, kappa, err = find_delta(xl, yl, c, xr_t, yr_t, zr_t, bx, by, bz, omega,
```

```
        err_mat.append(err)
        print(f'Number of iterations = {iter}')
        print(f'delta:\n {np.array([round(by, 3), round(bz, 3), round(math.degrees(omega), 3), rour
3)])}')
        iter += 1
        if abs(old_by - by) < 1e-3:
            break

    model_L, model_R, pY, scale_left, scale_right = space_intersection(xl, yl, c, xr_t, yr_t, zr_t,

    print(f'Model L:\n {model_L}\n')
    print(f'Model R:\n {model_R}\n')

    print(f'scale: \n{scale_left}\n')
    print(f'mu: \n{scale_right}\n')

    print(f'y-parallax values: \n{pY}\n')

    plot_scale(scale_left, scale_right)

    corr_matrix = find_corr_matrix(err_mat)
    print(f'Correlation Coefficient Matrix: \n{corr_matrix}\n')
```