Lab4.py

```python
"""
CIVE 6374 – Optical Imaging Metrology
Professor: Dr. Craig Glennie
Author: Joshua Genova
Lab # 4
Description: Absolute Orientation
Deadline: April 05, 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 preLab4 as pre_lab_4

def rot_mat(W, P, K):
    rot_mat = np.array([
        [cos(P)*cos(K), cos(W)*sin(K)+sin(W)*sin(P)*cos(K), sin(W)*sin(K)-cos(W)*sin(P)*cos(K)],
        [-cos(P)*sin(K), cos(W)*cos(K)-sin(W)*sin(P)*sin(K), sin(W)*cos(K)+cos(W)*sin(P)*sin(K)],
        [sin(P), -sin(W)*cos(P), cos(W)*cos(P)]
    ])

    return rot_mat

def approx_vals(Xm, Ym, Zm, Xg, Yg, Zg):
    Xm1 = Xm[0]
    Ym1 = Ym[0]
    Zm1 = Zm[0]
    Xm2 = Xm[1]
    Ym2 = Ym[1]
    Zm2 = Zm[1]

    Xo1 = Xg[0]
    Yo1 = Yg[0]
    Zo1 = Zg[0]
    Xo2 = Xg[1]
    Yo2 = Yg[1]
    Zo2 = Zg[1]

    X_m = np.array([Xm1, Ym1, Zm1])
    X_o = np.array([Xo1, Yo1, Zo1])

    K = math.atan2((Xo2-Xo1), (Yo2-Yo1)) - math.atan2((Xm2-Xm1), (Ym2-Ym1))
    M_0 = rot_mat(0, 0, K)

    d_o = math.sqrt((Xo2 - Xo1)**2 + (Yo2 - Yo1)**2 + (Zo2 - Zo1)**2)
    d_m = math.sqrt((Xm2 - Xm1)**2 + (Ym2 - Ym1)**2 + (Zm2 - Zm1)**2)
    scale = d_o/d_m

    t_vec = X_o - scale*np.dot(M_0, X_m)

    return K, t_vec[0], t_vec[1], t_vec[2], scale
```

```python
def partial_d(Xm, Ym, Zm, W, P, K, scale, M):
    sW = sin(W)
    cW = cos(W)
    sP = sin(P)
    cP = cos(P)
    sK = sin(K)
    cK = cos(K)

    m11 = M[0][0]
    m12 = M[0][1]
    m13 = M[0][2]
    m21 = M[1][0]
    m22 = M[1][1]
    m23 = M[1][2]
    m31 = M[2][0]
    m32 = M[2][1]
    m33 = M[2][2]

    # partial derivatives of X
    dXdW = scale*Ym*(-sW*sK + cW*sP*cK) + scale*Zm*(cW*sK + sW*sP*cK)
    dXdP = -scale*Xm*sP*cK + scale*Ym*sW*cP*cK - scale*Zm*cW*cP*cK
    dXdK = -scale*Xm*cP*sK + scale*Ym*(cW*cK - sW*sP*sK) + scale*Zm*(sW*cK + cW*sP*sK)
    dXdscale = Xm*m11 + Ym*m12 + Zm*m13
    dXdtx = 1
    dXdty = 0
    dXdtz = 0
    dX = [dXdW, dXdP, dXdK, dXdtx, dXdty, dXdtz, dXdscale]

    # partial derivatives of Y
    dYdW = scale*Ym*(-sW*cK - cW*sP*sK) + scale*Zm*(cW*cK - sW*sP*sK)
    dYdP = scale*Xm*sP*sK - scale*Ym*sW*cP*sK + scale*Zm*cW*cP*sK
    dYdK = -scale*Xm*cP*cK + scale*Ym*(-cW*sK - sW*sP*cK) + scale*Zm*(-sW*sK + cW*sP*cK)
    dYdscale = Xm*m21 + Ym*m22 + Zm*m23
    dYdtx = 0
    dYdty = 1
    dYdtz = 0
    dY = [dYdW, dYdP, dYdK, dYdtx, dYdty, dYdtz, dYdscale]

    # partial derivatives of Z
    dZdW = -scale*Ym*cW*cP - scale*Zm*sW*cP
    dZdP = scale*Xm*cP + scale*Ym*sW*sP - scale*Zm*cW*sP
    dZdK = 0
    dZdscale = Xm*m31 + Ym*m32 + Zm*m33
    dZdtx = 0
    dZdty = 0
    dZdtz = 1
    dZ = [dZdW, dZdP, dZdK, dZdtx, dZdty, dZdtz, dZdscale]

    return dX, dY, dZ

def misclosure(Xm, Ym, Zm, tx, ty, tz, scale, M, Xg, Yg, Zg):
    m11 = M[0][0]
    m12 = M[0][1]
    m13 = M[0][2]
    m21 = M[1][0]
```

```
    m22 = M[1][1]
    m23 = M[1][2]
    m31 = M[2][0]
    m32 = M[2][1]
    m33 = M[2][2]

    wX = scale*(m11*Xm + m12*Ym + m13*Zm) + tx - Xg
    wY = scale*(m21*Xm + m22*Ym + m23*Zm) + ty - Yg
    wZ = scale*(m31*Xm + m32*Ym + m33*Zm) + tz - Zg

    return wX, wY, wZ

def find_deltas(Xm, Ym, Zm, W, P, K, tx, ty, tz, scale, Xg, Yg, Zg):
    M = rot_mat(W, P, K)
    size = len(Xm)
    A_mat = np.zeros(shape=(3*size, 7))
    w = np.zeros(shape=(3*size, 1))
    idx = 0
    for i in range(size):
        A_mat[idx], A_mat[idx+1], A_mat[idx+2] = partial_d(Xm[i], Ym[i], Zm[i], W, P, K, scale, M)
        w[idx], w[idx+1], w[idx+2] = misclosure(Xm[i], Ym[i], Zm[i], tx, ty, tz, scale, M, Xg[i], Y
        idx += 3

    delta = -np.dot(np.dot(inv(np.dot(A_mat.T, A_mat)), A_mat.T), w)
    W = W + delta[0]
    P = P + delta[1]
    K = K + delta[2]
    tx = tx + delta[3]
    ty = ty + delta[4]
    tz = tz + delta[5]
    scale = scale + delta[6]

    return W[0],  P[0], K[0],  tx[0], ty[0], tz[0],  scale[0],  A_mat

def object_space(Xm, Ym, Zm, W, P, K, tx, ty, tz, scale):
    M = rot_mat(W, P, K)
    coords = np.array([[Xm], [Ym], [Zm]])
    t_vec = np.array([tx, ty, tz])
    Xo, Yo, Zo = scale*np.dot(M, coords) + t_vec

    return Xo[0], Yo[1],  Zo[2]

def resid(Xg, Yg, Zg, Xo, Yo, Zo):
    res_x = np.zeros(len(Xg))
    res_y = np.zeros(len(Yg))
    res_z = np.zeros(len(Zg))

    x_rms = 0
    y_rms = 0
    z_rms = 0

    for i in range(len(Xg)):
        res_x[i] = Xo[i] - Xg[i]
        res_y[i] = Yo[i] - Yg[i]
        res_z[i] = Zo[i] - Zg[i]
```

```python
            x_rms = x_rms + res_x[i]**2
            y_rms = y_rms + res_y[i]**2
            z_rms = z_rms + res_z[i]**2

        x_rms = math.sqrt((1/len(Xg))*x_rms)
        y_rms = math.sqrt((1/len(Xg))*y_rms)
        z_rms = math.sqrt((1/len(Xg))*z_rms)

        return res_x, res_y, res_z, x_rms, y_rms, z_rms

    def object_space_pc(B, W, P, K, tx, ty, tz, scale):
        M = rot_mat(W, P, K)
        t_hat = np.array([[tx], [ty], [tz]])

        rpcL = t_hat
        rpcR = scale*np.dot(M, B) + t_hat

        return rpcL, rpcR

    def trans_angles(w, p, k, W, P, K):
        M_m_i_R = rot_mat(w, p, k)
        M_m_i_L = rot_mat(0, 0, 0)
        M_m_o = rot_mat(W, P, K)

        M_o_i_L = np.dot(M_m_i_L, M_m_o.T)
        M_o_i_R = np.dot(M_m_i_R, M_m_o.T)

        return M_o_i_L, M_o_i_R

    def find_corr_matrix(A_mat):
        S_mat = np.zeros(shape=(7,7))
        C = inv(np.dot(A_mat.T, A_mat))
        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

    def extract_angles(M_o_i_L, M_o_i_R):

        m11_l = M_o_i_L[0][0]
        m21_l = M_o_i_L[1][0]
        m31_l = M_o_i_L[2][0]
        m32_l = M_o_i_L[2][1]
        m33_l = M_o_i_L[2][2]

        m11_r = M_o_i_R[0][0]
        m21_r = M_o_i_R[1][0]
        m31_r = M_o_i_R[2][0]
        m32_r = M_o_i_R[2][1]
        m33_r = M_o_i_R[2][2]

        w_l = math.atan2(-m32_l,m33_l)
        p_l = math.asin(m31_l)
        k_l = math.atan2(-m21_l,m11_l)
```

```
    left_image_angles = np.array([math.degrees(w_l), math.degrees(p_l), math.degrees(k_l)])

    w_r = math.atan2(-m32_r,m33_r)
    p_r = math.asin(m31_r)
    k_r = math.atan2(-m21_r,m11_r)
    right_image_angles = np.array([math.degrees(w_r), math.degrees(p_r), math.degrees(k_r)])

    return left_image_angles, right_image_angles
####################################################################################################
if __name__=="__main__":
    c = c = 153.358
    Xg= [-399.28, 475.55, 517.62]
    Yg = [-679.72, -538.18, -194.43]
    Zg = [1090.96, 1090.5, 1090.65]
    image_model_L, image_model_R = pre_lab_4.task_1()
    print(f'Model Space: \n{image_model_L}')
    Xm = image_model_L[:,0]
    Ym = image_model_L[:,1]
    Zm = image_model_L[:,2]

    # initial conditions
    K, tx, ty, tz, scale = approx_vals(Xm, Ym, Zm, Xg, Yg, Zg)
    W = 0
    P = 0
    print('Initial Conditions')
    print(f'W: {math.degrees(W)}')
    print(f'P: {math.degrees(P)}')
    print(f'K: {math.degrees(K)}')
    print(f'tx: {tx}')
    print(f'ty: {ty}')
    print(f'tz: {tz}')
    print(f'scale: {scale}\n')

    # Scale number
    S = 5000
    # deviation
    r_obs = 15e-6
    tol_coords = S*r_obs/10
    tol_til = r_obs/(10*c)
    tol_scale = 1e-5
    print(f'Tol coords: {tol_coords}')
    print(f'Tol tilt: {tol_til}')
    print(f'Tol scale: {tol_scale}')


    # Task 1
    print('-'*80)
    Task = 1
    print(f'Task # {Task}')
    iters = 10
    for i in range(iters):
        W_old = W
        P_old = P
        K_old = K
        tx_old = tx
        ty_old = ty
```

```
        tz_old = tz
        scale_old = scale
        W, P, K, tx, ty, tz, scale, A_mat = find_deltas(Xm, Ym, Zm, W, P, K, tx, ty, tz, scale, Xg,
        if abs(W-W_old) < tol_til and abs(P-P_old) < tol_til and abs(K-K_old) < tol_til and abs(tx-
tol_coords and abs(ty-ty_old) < tol_coords and abs(tz-tz_old) < tol_coords and abs(scale-scale_old)
            print(f'Converged at {i+1} iterations!')
            break

    print(f'\ntx: {tx} m')
    print(f'ty: {ty} m')
    print(f'tz {tz} m')
    print(f'omega: {math.degrees(W)} deg')
    print(f'phi: {math.degrees(P)} deg')
    print(f'kappa: {math.degrees(K)} deg')
    print(f'lambda: {scale}\n')

    # Task 2
    print('-'*80)
    Task += 1
    print(f'Task # {Task}')

    # Task 3
    print('-'*80)
    Task += 1
    print(f'Task # {Task}')

    Xo_vec=np.zeros(len(Xm))
    Yo_vec=np.zeros(len(Ym))
    Zo_vec=np.zeros(len(Zm))
    idx = 0
    for i in range(len(Xm)):
        Xo_vec[idx], Yo_vec[idx], Zo_vec[idx] = object_space(Xm[i], Ym[i], Zm[i], W, P, K, tx, ty,
        idx += 1
    print(f"Object Space X: {Xo_vec}")
    print(f"Object Space Y: {Yo_vec}")
    print(f"Object Space Z: {Zo_vec}\n")

    vX, vY, vZ, x_rms, y_rms, z_rms = resid(Xg, Yg, Zg, Xo_vec, Yo_vec, Zo_vec)
    print(f'vX: {vX}')
    print(f'vY: {vY}')
    print(f'vZ: {vZ}')
    print(f'x_rms: {x_rms}')
    print(f'y_rms: {y_rms}')
    print(f'z_rms: {z_rms}\n')

    # Task 4
    print('-'*80)
    Task += 1
    print(f'Task # {Task}')
    bx = 92.000
    by = -1.421510121593803
    bz = -1.2872527980970032
    B = np.array([[bx], [by], [bz]])
    rpcL, rpcR = object_space_pc(B, W, P, K, tx, ty, tz, scale)
    print(f'Left Image PC: \n{rpcL}')
    print(f'Right Image PC: \n{rpcR}\n')
```

```
    # Task 5
    print('-'*80)
    Task += 1
    print(f'Task # {Task}')
    print('Task 5:')
    Xg_val = [-466.39, 42.73, 321.09, 527.78]
    Yg_val = [-542.31, -412.19, -667.45, -375.72]
    Zg_val = [1091.55, 1090.82, 1083.49, 1092]
    image_model_L_val, image_model_R_val = pre_lab_4.task_5()
    print(f'Model Space (Validation):\n {image_model_L_val}')
    Xm_val = image_model_L_val[:,0]
    Ym_val = image_model_L_val[:,1]
    Zm_val = image_model_L_val[:,2]

    # calculate residuals for validation coords
    Xo_vec_val=np.zeros(len(Xm_val))
    Yo_vec_val=np.zeros(len(Ym_val))
    Zo_vec_val=np.zeros(len(Zm_val))
    idx = 0
    for i in range(len(Xm_val)):
        Xo_vec_val[idx], Yo_vec_val[idx], Zo_vec_val[idx] = object_space(Xm_val[i], Ym_val[i], Zm_v
tx, ty, tz, scale)
        idx += 1

    print(f"Validation Object Space X: {Xo_vec_val}")
    print(f"Validation Object Space Y: {Yo_vec_val}")
    print(f"Validation Object Space Z: {Zo_vec_val}\n")

    vX_val, vY_val, vZ_val, x_rms_val, y_rms_val, z_rms_val = resid(Xg_val, Yg_val, Zg_val, Xo_vec_
Zo_vec_val)
    print(f'vX_val: {vX_val}')
    print(f'vY_val: {vY_val}')
    print(f'vZ_val: {vZ_val}')
    print(f'x_rms_val: {x_rms_val}')
    print(f'y_rms_val: {y_rms_val}')
    print(f'z_rms_val: {z_rms_val}\n')

    print(f'Correlation Coefficient Matrix: \n{find_corr_matrix(A_mat)}')

    # Task 6
    print('-'*80)
    Task += 1
    print(f'Task # {Task}')
    print(f'r_x: {S*r_obs}')
    print(f'r_y: {S*r_obs}')
    print(f'r_H: {math.sqrt(2)*S*r_obs/0.6}')

    # Task 7
    print('-'*80)
    Task += 1
    print(f'Task # {Task}')
    w = -0.017060700258602l2
    p = 0.004738042393395472
    k = -0.030192790615912662
    M_o_i_L_all, M_o_i_R_all = trans_angles(w, p, k, W, P, K)
```

```
# Task 8
print('-'*80)
Task += 1
print(f'Task # {Task}')
left_image_angles, right_image_angles = extract_angles(M_o_i_L_all, M_o_i_R_all)
print(f'Extracted Angles Left: {left_image_angles}')
print(f'Extracted Angles Right: {right_image_angles}')
```