Course name

课程实习报告

学生姓名 Name

学　　院 李四光学院

班　　级 201226

学　　号 2022100XXXX

授课教师 Teacher Name

2025 年 2 月 13 日

目录

1 单像空间后方交会 1

1.1 实验原理与流程 1

1.1.1 实验原理 1

1.2 实习数据与代码部分 2

1.2.1 实习数据 2

1.2.2 实习要求 2

1.2.3 实习代码 2

1.3 实习结果与数据分析 2

1.3.1 实习结果 2

1.3.2 成果分析 2

1.4 单次实习小结 3

2 写在最后 4

2.1 发布地址 4

3 实习体会 4

附录 A 文件列表 4

附录 B 代码 4

1 单像空间后方交会

1.1 实验原理与流程

1.1.1 实验原理

此为公式与图片输入方法示意：

图 1 单像空间后方交会原理图

详细流程：基本关系式为共线条件方程式：

$$x = -f\frac{a_1(X-X_S)+b_1(Y-Y_S)+c_1(Z-Z_S)}{a_3(X-X_S)+b_3(Y-Y_S)+c_3(Z-Z_S)} = -f\frac{\overline{X}}{\overline{Z}}$$
$$y = -f\frac{a_2(X-X_S)+b_2(Y-Y_S)+c_2(Z-Z_S)}{a_3(X-X_S)+b_3(Y-Y_S)+c_3(Z-Z_S)} = -f\frac{\overline{Y}}{\overline{Z}}$$

其中，f 为焦距，$a_1, b_1, c_1, a_2, b_2, c_2, a_3, b_3, c_3$ 为外方位元素，X_S, Y_S, Z_S 为相对于像片坐标系的相对坐标，X, Y, Z 为地面点的地理坐标，x, y 为像点的坐标。通过以上方程，可以得到三个方程，通过解方程组，可以求解出外方位元素。

用矩阵形式表示：

$$V = AX - l$$

$$V = [v_x, \quad v_y]^{\mathrm{T}}$$

$$A = \begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} & a_{15} & a_{16} \\ a_{21} & a_{22} & a_{23} & a_{24} & a_{25} & a_{26} \end{bmatrix}$$

$$X = [dX_S \quad dY_S \quad dZ_S \quad d\varphi \quad d\omega \quad d\kappa]^T$$

$$l = [l_x \quad l_y]^T$$

1.2 实习数据与代码部分

1.2.1 实习数据

请输入文本

1.2.2 实习要求

XXX 请输入文本

1.2.3 实习代码

详细代码请见报告末尾附录 A 部分的文件列表示意与附录 B 的代码部分，在行文中不再加以赘述。

1.3 实习结果与数据分析

1.3.1 实习结果

XXX 请输入文本

1.3.2 成果分析

XXX 请输入文本

1.4 单次实习小结

XXX 请输入文本

2 写在最后

2.1 发布地址

代码将在本人的 Github 上发布并进行存档。

- Github: https://github.com/HMBlankcat

3 实习体会

请输入文本

附录 A 文件列表

表 1 文件列表

文件名	功能描述
单像空间后方交会.py	单像空间后方交会程序代码

附录 B 代码

单像空间后方交会.py

```
import numpy as np

#摄影机主距、内方位元素、相片比例尺
fk=153.24
m=50000.0
x0=0.0
y0=0.0
center=np.matrix([x0,y0])

#地面点坐标
mat_ground=np.matrix([[36589.41,25273.32,2195.17],
                      [37631.08,31324.51,728.69],
                      [39100.97,24934.98,2386.50],
                      [40426.54,30319.81,757.31]])
#像点坐标
```

```
mat_photo=np.matrix([[-86.15,-68.99],
                     [-53.40,82.21],
                     [-14.78,-76.63],
                     [10.46,64.43]])

#外方位元素
PhotoCenter=np.sum(mat_ground,axis=0)/4 #计算的是地面点的中心

PhotoCenter[0,2]=fk*m/1000.0 #摄影机的高度
H=PhotoCenter[0,2]

print('PhotoCenter:',PhotoCenter)

#外方位角元素phi omega kapa
phi=omega=ka=0 #初始化
Angle=np.matrix([phi,omega,ka])
#设置改正数组，Xs，Ys，Zs，phi,omega,kapa
Iteration=0 #迭代次数
while True:
    Iteration+=1
    print('这是第',Iteration,'次迭代~')
    #计算旋转矩阵
    R=np.matrix([[np.cos(phi)*np.cos(ka)-np.sin(phi)*np.sin(omega)*np.sin(ka
        )    , np.cos(omega)*np.sin(ka)    , np.sin(phi)*np.cos(ka)+np.cos(
        phi)*np.sin(omega)*np.sin(ka)],
                 [-np.cos(phi)*np.sin(ka)-np.sin(phi)*np.sin(omega)*np.cos(ka
                    )   , np.cos(omega)*np.cos(ka)    , -np.sin(phi)*np.sin(
                    ka)+np.cos(phi)*np.sin(omega)*np.cos(ka)],
                 [-np.sin(phi)*np.cos(omega)
                                                        , -np.sin(omega)
                                  , np.cos(phi)*np.cos(omega)]])

    a1, b1, c1 = R[0, 0], R[0, 1], R[0, 2]
    a2, b2, c2 = R[1, 0], R[1, 1], R[1, 2]
    a3, b3, c3 = R[2, 0], R[2, 1], R[2, 2]

    Ra=np.matrix([[a1,b1,c1],
                  [a2,b2,c2],
                  [a3,b3,c3]])
    print('Ra:', Ra)

    #常数项lx，ly
    lxy=np.zeros_like(mat_photo)
```

```
    #Zb
    Zb=np.zeros((mat_ground.shape[0],1))
    #A系数矩阵
    A=np.zeros((2*mat_ground.shape[0],6))

    #共线条件方程
    for i in range(mat_photo.shape[0]):
        lx=mat_photo[i,0]+fk*(a1 * (mat_ground[i, 0] - PhotoCenter[0, 0]) +
            b1 * (mat_ground[i, 1] - PhotoCenter[0, 1]) + c1 * (
                        mat_ground[i, 2] - PhotoCenter[0, 2])) / (
                        a3 * (mat_ground[i, 0] - PhotoCenter[0, 0]) + b3 * (
                            mat_ground[i, 1] - PhotoCenter[0, 1]) + c3 * (
                            mat_ground[i, 2] - PhotoCenter[0, 2]))
        ly=mat_photo[i,1]+fk*(a2 * (mat_ground[i, 0] - PhotoCenter[0, 0]) +
            b2 * (mat_ground[i, 1] - PhotoCenter[0, 1]) + c2 * (
                        mat_ground[i, 2] - PhotoCenter[0, 2])) / (
                         a3 * (mat_ground[i, 0] - PhotoCenter[0, 0]) + b3 *
                             (mat_ground[i, 1] - PhotoCenter[0, 1]) + c3 * (
                             mat_ground[i, 2] - PhotoCenter[0, 2]))

        lxy[i,0]=lx
        lxy[i,1]=ly
        x=mat_photo[i,0]
        y=mat_photo[i,1]

        Zb[i,0]=a3*(mat_ground[i,0]-PhotoCenter[0,0])+b3*(mat_ground[i,1]-
            PhotoCenter[0,1])+c3*(mat_ground[i,2]-PhotoCenter[0,2])
        A[2 * i, 0] = (a1 * fk + a3 * x) / Zb[i, 0]
        A[2 * i, 1] = (b1 * fk + b3 * x) / Zb[i, 0]
        A[2 * i, 2] = (c1 * fk + c3 * x) / Zb[i, 0]
        A[2 * i, 3] = y * np.sin(omega) - (x * (x * np.cos(ka) - y * np.sin(
            ka)) / fk + fk * np.cos(ka)) * np.cos(omega)
        A[2 * i, 4] = -fk * np.sin(ka) - x * (x * np.sin(ka) + y * np.cos(ka
            )) / fk
        A[2 * i, 5] = y
        A[2 * i + 1, 0] = (a2 * fk + a3 * y) / Zb[i, 0]
        A[2 * i + 1, 1] = (b2 * fk + b3 * y) / Zb[i, 0]
        A[2 * i + 1, 2] = (c2 * fk + c3 * y) / Zb[i, 0]
        A[2 * i + 1, 3] = -x * np.sin(omega) - (y * (x * np.cos(ka) - y * np
            .sin(ka) )/ fk - fk * np.sin(ka)) * np.cos(omega)
        A[2 * i + 1, 4] = -fk * np.cos(ka) - y * (x * np.sin(ka) + y * np.
            cos(ka)) / fk
        A[2 * i + 1, 5] = -x
```

```
87
88      l=np.matrix(lxy).reshape(8,1) #求转置
89      print('Zb:',Zb)
90      print('A:',A)
91      print('l:',l)
92      A=np.matrix(A)#将A转换为矩阵
93      #最小二乘法求解
94      mat1=((A.T*A).I*A.T*l)
95      delta=mat1
96      print('delta:',delta)
97      dphi, domega, dka = delta[3, 0], delta[4, 0], delta[5, 0]
98      phi += dphi
99      omega += domega
100     ka += dka
101     PhotoCenter[0, 0] += delta[0, 0]
102     PhotoCenter[0, 1] += delta[1, 0]
103     PhotoCenter[0, 2] += delta[2, 0]
104     H=PhotoCenter[0,2]
105     print('PhotoCenter:',PhotoCenter)
106     print('phi:',phi)
107     print('omega:',omega)
108     print('ka:',ka)
109     #迭代停止条件
110     if np.abs(delta).max()< 3e-5:
111         print('delta:', delta)
112         print('dphi', dphi)
113         print('domega', domega)
114         print('dka', dka)
115         break
116
117 print('phi:',phi)
118 print('omega:',omega)
119 print('ka:',ka)
120 print('PhotoCenter:',PhotoCenter)
121 # 计算残差向量v
122 v = l - A * delta
123 #残差的中误差计算
124 n = 2 * mat_ground.shape[0]  #观测方程的数量
125 u = 6  #未知数的数量
126 sigma = np.sqrt((v.T * v)[0, 0] / (n - u))  #中误差计算
127
128 print('残差向量v:',v)
129 print('中误差sigma:',sigma)
```