**单片空间后方交会实习报告**

班级： 2022级A3班

姓名： 阮思琳

学号： 2022302131290

二〇二四年九月

**目 录**

[一 实习目的 1](#_Toc178114058)

[1.1 实习的目的和意义 1](#_Toc178114059)

[二 实习内容 1](#_Toc178114060)

[2.1 题目 1](#_Toc178114061)

[2.2 基本原理 2](#_Toc178114062)

[三 实习结果与分析 5](#_Toc178114063)

[3.1 程序结果 5](#_Toc178114064)

[四 实习体会 6](#_Toc178114065)

[4.1 漏洞调试及解决 6](#_Toc178114066)

[4.2 实习体会 7](#_Toc178114067)

[五 代码附录 8](#_Toc178114068)

# 

# 一 实习目的

## 1.1 实习的目的和意义

**实践编程技能与空间几何算法的结合**：本实习旨在通过动手编写一个基于Visual C++的单片空间后方交会程序，将编程技能与摄影测量中的空间几何计算紧密结合。通过处理实验数据，实现像片外方位元素的自动计算与输出，并附带精度评估功能，以此锻炼将理论知识转化为实际编程代码的能力。

**深化空间后方交会理论认知与编程实践**：深入探索单片空间后方交会的核心原理，理解其在摄影测量中的应用价值。特别关注在存在多余观测数据的情况下，如何通过编程实现最小二乘平差方法，以优化求解影像的外方位元素。这一过程不仅加深了对理论的理解，还提升了解决实际复杂问题的能力。

**强化动手调试与实验结果分析能力**：通过实际的上机操作，调试和优化所编写的程序，解决编程过程中遇到的各种问题，从而锻炼动手能力和问题解决能力。进一步地，对实验结果进行详细分析，评估程序的准确性和效率，这一过程将极大地提升综合运用所学知识（包括编程、数学、摄影测量等）来解决实际问题的能力，为后续的专业学习和工作打下坚实的基础。

# 二 实习内容

## 2.1 题目

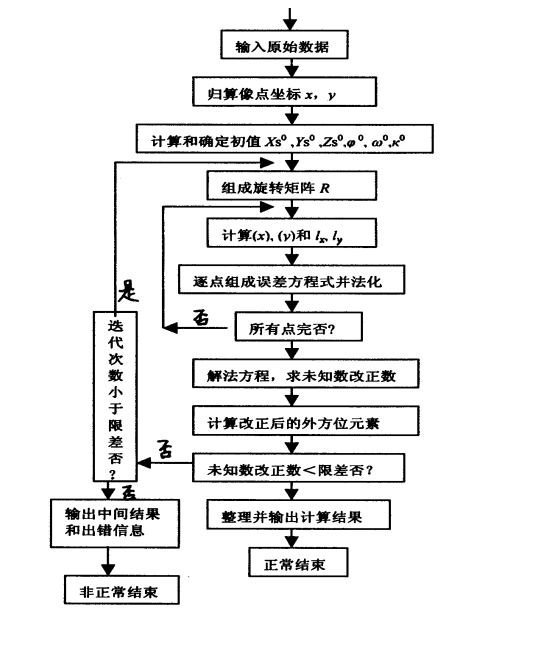


图2.1题目

已知四对点的**影像坐标和地面坐标**如上表所示，假设**内方位元素**已知：f=153.24mm，m=50000，需要我们计算近似垂直情况下，空间后方的解，即求出**像点坐标、地面坐标、单位权中误差、外方位元素及其精度**。

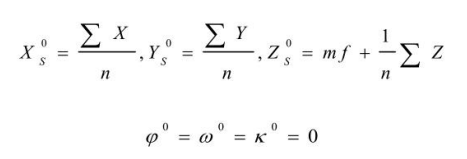
## 2.2 基本原理

利用一定数量的地面控制点，根据共线条件方程求解像片外方位元素，并评价其精度：

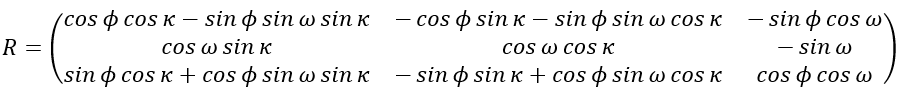


**第一步**：首先获得已知数据，包括：内方位元素 x0 y0 f、控制点的影像坐标 x y 及其对应的地面坐标 X Y Z，影像比例尺 1/m，m=50000。

**第二步**，确定未知数的初始值

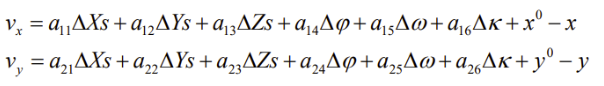


**第三步**，利用角元素近似值计算方向余弦，组成旋转矩阵，矩阵中各元素的计算公式如下：



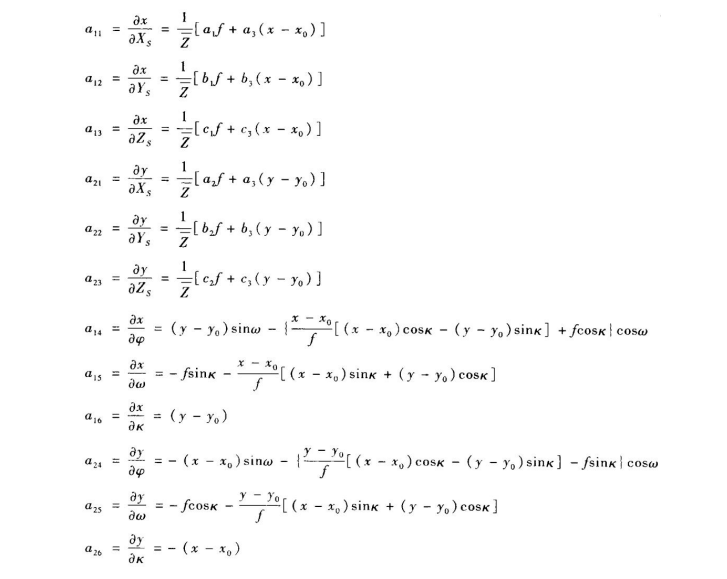
**第四步**，计算像点坐标的近似值。利用未知数的近似值和控制点的地面坐标，代入以共线方程式，得到各个控制点对应像点的像平面坐标近似值(x)、(y)：

**第五步**，计算误差方程系数矩阵和常数项，组成误差方程式。一个控制点对应的误差方程式为：



矩阵形式为：

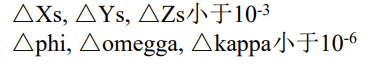
经推导后可得系数矩阵A各偏导数的值为，



**第六步**，计算间接平差模型法方程的是系数矩阵ATA和常数项ATL，组成法方程式：

**第七步**，求得外方位元素的改正数dXs，dYs，dZs，dφ，dω，dκ。用前次迭代取得的近似值，加本次迭代的改正数，计算外方位元素的新值。

**第八步**，限差检测。将求得的外方位元素改正数与规定的限差比较，若小于限差，则迭代结束。否则用新的近似值重复第三步～第七步满足要求为止。



**第九步**，精度评定，由误差方程的计算式，单位权中误差的计算式，以及平差中6个参数的协因数得到参数Xi的中误差.

# 三 实习结果与分析

## 3.1 程序结果

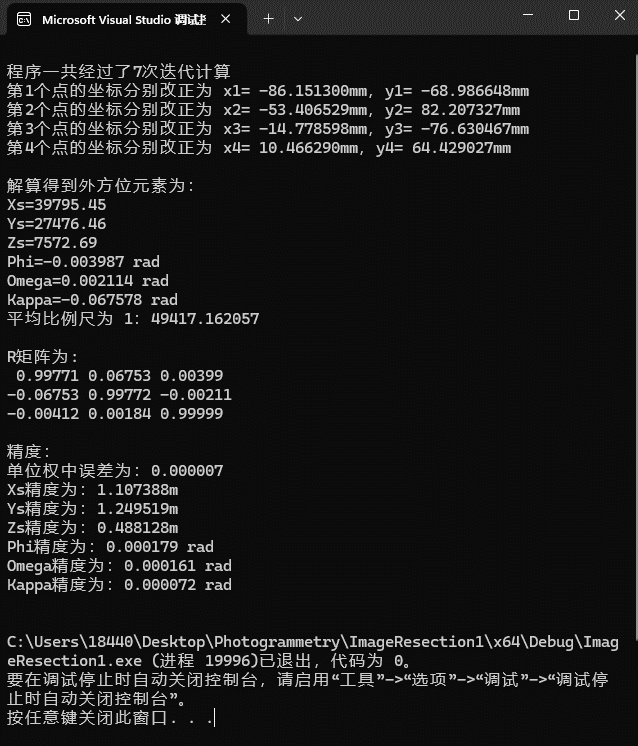


图3.1 程序结果图

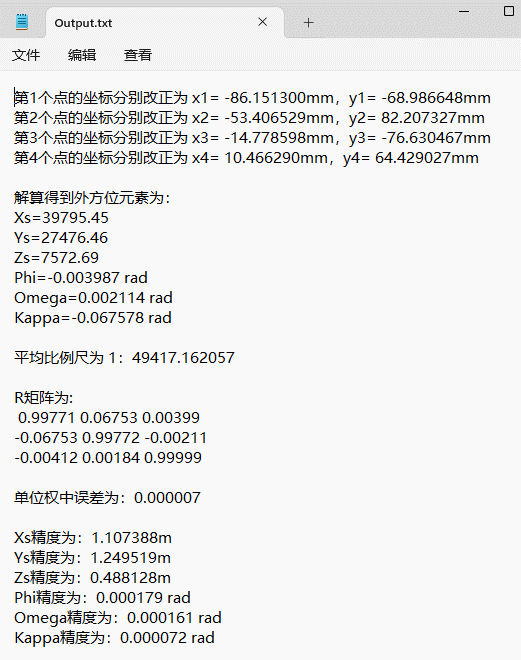


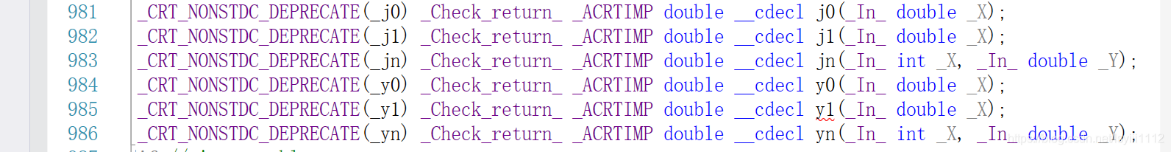
图3.1 存储txt文件

# 四 实习体会

## 4.1 漏洞调试及解决

1、“y1”重定义问题：

在程序中尽量避免定义使用y1变量，y1在math.h里定义过了，另外，不仅是y1,连j0,j1,jn,y0,yn都被定义了。



2、通过opencv的Mat构造矩阵和数组赋值：

C++本身没有关于矩阵计算的函数，但是可以通过opencv构造Mat矩阵，使用mat的成员函数实现矩阵转置(,t)、求逆（.inv）、相乘（\*）等操作

将系数矩阵所有系数存入到一个一维数组A中，之后通过设置Mat类矩阵AA的尺寸和通道数将数组变量拆分成矩阵。（注意：由数组赋值给矩阵的过程是首先将矩阵中第一个元素的所有通道依次赋值，之后再赋值下一个元素。）

Mat AA = Mat(2 \* n, 6, CV\_64FC1, A);

## 4.2 实习体会

此次单片空间后方交会实习不仅深化了我对单张影像空间后方交会原理的认知，更锻炼了我通过计算机编程实现摄影测量计算的能力，是一次将抽象概念具象化、将知识转化为技能的宝贵历程。

这一过程中，我深刻体会到理论与实践的紧密关联。原本在课堂上略显晦涩的原理与步骤，在亲手操作与调试中变得生动而具体。我不仅能够回溯并巩固每个计算公式的逻辑根基，更能在实践中灵活运用，确保每一步操作都精准无误，从而保障最终结果的可靠性与准确性，做到“知其然更知其所以然”。

面对实习中的挑战，我回归基础，深入研读教材与参考书籍，力求从源头上把握问题的本质与解决之道。这一过程不仅帮助我解决了眼前的难题，更拓宽了我的知识视野，加深了我对空间后方交会技术全面而深刻的理解。同时，我充分利用互联网资源，广泛搜集相关教程与案例，通过对比分析，汲取多方智慧，为问题的解决提供了更多元化的视角与思路。

总之，通过本次单片空间后方交会的课间实习，我不仅成功克服了实习中的种种障碍，更在此过程中锤炼了问题解决的能力，提升了编程技巧，加深了对专业知识的记忆与理解。不仅是一次知识的深化之旅，更是一次能力的飞跃与成长的见证。我将在未来的学习与工作中，继续秉持这种理论与实践相结合的精神，为将来的学术和职业生涯奠定了坚实的基础。

# 五 代码附录

#include<windows.h>

#include <iostream>

#include"opencv2/opencv.hpp"

#include <vector>

using namespace std;

#pragma comment(lib,"opencv\_world480d.lib")

using namespace cv;

//内方位元素

const double f = 153.24 / 1000;

const double x0 = 0;

const double y\_0 = 0;

const int n = 4, m = 15000;

const double ms = 1e-3, mphi = 1e-6;

//定义点

struct Points {

double x;

double y;

double X;

double Y;

double Z;

};

int main()

{

//输入原始数据

Points p[4];

p[0].x = -86.15/1000, p[0].y = -68.99/1000, p[0].X = 36589.41, p[0].Y = 25273.32, p[0].Z = 2195.17;

p[1].x = -53.40/1000, p[1].y = 82.21/1000, p[1].X = 37631.08, p[1].Y = 31324.51, p[1].Z = 728.69;

p[2].x = -14.78/1000, p[2].y = -76.63/1000, p[2].X = 39100.97, p[2].Y = 24934.98, p[2].Z = 2386.50;

p[3].x = 10.46/1000, p[3].y = 64.43/1000, p[3].X = 40426.54, p[3].Y = 30319.81, p[3].Z = 757.31;

//初值

double Xs, Ys, Zs, H;

double phi, omega, kappa;

Xs = (p[0].X + p[1].X + p[2].X + p[3].X) / n;

Ys = (p[0].Y + p[1].Y + p[2].Y + p[3].Y) / n;

Zs = m \* f;

phi = omega = kappa = 0;

//结果数据

int t = 20;//最多迭代次数

double a1, a2, a3, b1, b2, b3, c1, c2, c3;//旋转矩阵R

double zeta;//单位权中误差

Mat QQ (6, 6, CV\_64FC1);//精度矩阵

Mat vv(2\*n, 1, CV\_64FC1);//改正数矩阵

do {

//利用角元素计算旋转矩阵

a1 = cos(phi) \* cos(kappa) - sin(phi) \* sin(omega) \* sin(kappa);

a2 = -cos(phi) \* sin(kappa) - sin(phi) \* sin(omega) \* cos(kappa);

a3 = -sin(phi) \* cos(omega);

b1 = cos(omega) \* sin(kappa);

b2 = cos(omega) \* cos(kappa);

b3 = -sin(omega);

c1 = sin(phi) \* cos(kappa) + cos(phi) \* sin(omega) \* sin(kappa);

c2 = -sin(phi) \* sin(kappa) + cos(phi) \* sin(omega) \* cos(kappa);

c3 = cos(phi) \* cos(omega);

//循环计算改正数至小于限差

//计算像点坐标的近似值

double \_X, \_Y, \_Z, X, Y, Z, x, y,x\_j,y\_j;

double\* A = new double[2 \* n \* 6];

double\* L = new double[2 \* n ];

for (int i = 0; i < n; i++) {

//xy近似值

X = p[i].X;

Y = p[i].Y;

Z = p[i].Z;

x = p[i].x ;

y = p[i].y ;

//\_X,\_Y,\_Z!!!xyo与xy0

\_X = a1 \* (X - Xs) + b1 \* (Y - Ys) + c1 \* (Z - Zs);

\_Y = a2 \* (X - Xs) + b2 \* (Y - Ys) + c2 \* (Z - Zs);

\_Z = a3 \* (X - Xs) + b3 \* (Y - Ys) + c3 \* (Z - Zs);

//共线方程算xy近似值xj yj

x\_j = x0 - f \* \_X / \_Z;

y\_j= y\_0 - f \* \_Y / \_Z;

//系数矩阵元素数组A

A[0 + i \* 12] = (a1 \* f + a3 \* (x - x0)) / \_Z;

A[1 + i \* 12] = (b1 \* f + b3 \* (x - x0)) / \_Z;

A[2 + i \* 12] = (c1 \* f + c3 \* (x - x0)) / \_Z;

A[3 + i \* 12] = (y - y\_0) \* sin(omega) - ((x - x0) / f \* ((x - x0) \* cos(kappa) - (y - y\_0) \* sin(kappa)) + f \* cos(kappa)) \* cos(omega);

A[4 + i \* 12] = -f \* sin(kappa) - (x - x0) / f \* ((x - x0) \* sin(kappa) + (y - y\_0) \* cos(kappa));

A[5 + i \* 12] = y - y\_0;

A[6 + i \* 12] = (a2 \* f + a3 \* (y - y\_0)) / \_Z;

A[7 + i \* 12] = (b2 \* f + b3 \* (y - y\_0)) / \_Z;

A[8 + i \* 12] = (c2 \* f + c3 \* (y-y\_0)) / \_Z;

A[9 + i \* 12] = -(x - x0) \* sin(omega) - ((y-y\_0) / f \* ((x - x0) \* cos(kappa) - (y-y\_0) \* sin(kappa)) - f \* sin(kappa)) \* cos(omega);

A[10 + i \* 12] = -f \* cos(kappa) - (y-y\_0) / f \* ((x - x0) \* sin(kappa) + (y-y\_0) \* cos(kappa));

A[11 + i \* 12] = -(x - x0);

//常数项矩阵元素数组L

L[0 + 2 \* i] = x - x\_j;

L[1 + 2 \* i] = y - y\_j;

}

//系数矩阵AA 常数项矩阵L

Mat AA = Mat(2 \* n, 6, CV\_64FC1, A);

Mat LL = Mat(2 \* n, 1, CV\_64FC1, L);

//法方程式计算

Mat At = AA.t();

Mat AtA = At\*AA;

Mat AtA\_1 = AtA.inv();

QQ = AtA\_1.clone();

Mat AtL = At\*LL;

Mat xx = AtA\_1\*AtL;

vv = AA\*xx-LL;

Mat vtv = (vv.t())\*vv ;

//计算外方位元素的新值与精度计算

Xs += xx.at<double>(0, 0);

Ys += xx.at<double>(1, 0);

Zs += xx.at<double>(2, 0);

phi += xx.at<double>(3, 0);

omega += xx.at<double>(4, 0);

kappa += xx.at<double>(5, 0);

zeta = vtv.at<double>(0, 0);

zeta = sqrt(zeta / (2 \* n - 6));

//限差检查

if ((xx.at<double>(0, 0) < ms) && (xx.at<double>(1, 0) < ms) && (xx.at<double>(2, 0) < ms) && (xx.at<double>(3, 0) < mphi) && (xx.at<double>(4, 0) < mphi) && (xx.at<double>(5, 0) < mphi))

break;

//最多循环次数

t--;

if (t == 0)

printf("结果不收敛！\n\n\n");

//释放

AA.release(); LL.release(); At.release(); AtA.release();

AtA\_1.release(); AtL.release(); xx.release(); vtv.release();

delete[] A, L;

} while (t > 0);

//输出结果

if (t > 0) {

FILE\* fp;//建立一个文件操作指针

fopen\_s(&fp, "Output.txt", "w");

//"C:\Users\18440\Desktop\Photogrammetry\ImageResection1\Output.txt"

printf("\n程序一共经过了%d次迭代计算\n", 20 - t);

for (int i = 0; i < n; i++) {

printf("第%d个点的坐标分别改正为 x1= %lfmm，y1= %lfmm\n", i + 1, (vv.at<double>(i \* 2, 0) + p[i].x) \* 1000, (vv.at<double>(i \* 2 + 1, 0) + p[i].y) \* 1000);

fprintf(fp, "第%d个点的坐标分别改正为 x1= %lfmm，y1= %lfmm\n", i + 1, vv.at<double>(i \* 2, 0) + p[i].x, vv.at<double>(i \* 2 + 1, 0) + p[i].y);

}

printf("\n解算得到外方位元素为：\nXs=%.2lf\nYs=%.2lf\nZs=%.2lf\n", Xs, Ys, Zs);

fprintf(fp, "\n解算得到外方位元素为：\nXs=%.2lf\nYs=%.2lf\nZs=%.2lf\n", Xs, Ys, Zs);

printf("Phi=%lf rad\nOmega=%lf rad\nKappa=%lf rad\n", phi, omega, kappa);

printf("平均比例尺为 1：%f\n\n", Zs / f);

fprintf(fp, "Phi=%lf rad\nOmega=%lf rad\nKappa=%lf rad\n", phi, omega, kappa);

fprintf(fp, "\n平均比例尺为 1：%f\n\n", Zs / f);

printf("R矩阵为:\n %.5lf %.5lf %.5lf\n%.5lf %.5lf %.5lf\n%.5lf %.5lf %.5lf\n\n", a1, a2, a3, b1, b2, b3, c1, c2, c3);

fprintf(fp, "R矩阵为:\n %.5lf %.5lf %.5lf\n%.5lf %.5lf %.5lf\n%.5lf %.5lf %.5lf\n\n", a1, a2, a3, b1, b2, b3, c1, c2, c3);

printf("精度：\n单位权中误差为：%lf\n", zeta);

printf("Xs精度为：%lfm\n", zeta \* sqrt(QQ.at<double>(0, 0)));

printf("Ys精度为：%lfm\n", zeta \* sqrt(QQ.at<double>(1, 1)));

printf("Zs精度为：%lfm\n", zeta \* sqrt(QQ.at<double>(2, 2)));

printf("Phi精度为：%lf rad\n", zeta \* sqrt(QQ.at<double>(3, 3)));

printf("Omega精度为：%lf rad\n", zeta \* sqrt(QQ.at<double>(4, 4)));

printf("Kappa精度为：%lf rad\n\n", zeta \* sqrt(QQ.at<double>(5, 5)));

fprintf(fp, "单位权中误差为：%lf\n\n", zeta);

fprintf(fp, "Xs精度为：%lfm\n", zeta \* sqrt(QQ.at<double>(0, 0)));

fprintf(fp, "Ys精度为：%lfm\n", zeta \* sqrt(QQ.at<double>(1, 1)));

fprintf(fp, "Zs精度为：%lfm\n", zeta \* sqrt(QQ.at<double>(2, 2)));

fprintf(fp, "Phi精度为：%lf rad\n", zeta \* sqrt(QQ.at<double>(3, 3)));

fprintf(fp, "Omega精度为：%lf rad\n", zeta \* sqrt(QQ.at<double>(4, 4)));

fprintf(fp, "Kappa精度为：%lf rad\n\n", zeta \* sqrt(QQ.at<double>(5, 5)));

fclose(fp);

}

QQ.release(); vv.release();

return 0;

}