基于 Qt 框架、OpenCV 库的相关系数和最小二乘匹配

2022302131030 贾羽佳

一、实习目的

通过相关系数和最小二乘匹配算法的编程实现，加深对这两种匹配方法在图像匹配中应用的理解，同时提升编程技能和解决实际问题的能力。

二、实习内容

1. 相关系数匹配
2. 基于相关系数的最小二乘匹配
3. 影像文件的读入、写入，同名点文件的获取
4. 可视化界面的设计

三、设计思路

在查阅相关资料后，决定先用 Harris 特征点提取方法缩小图像匹配备选点个数来提高匹配速率，之后采用相关系数窗口匹配两张影像已提取出的备选点；以匹配好的点对作为初值，考虑影像几何变形和辐射畸变，对每对匹配点对进行最小二乘匹配精化匹配结果，编写函数实现用 txt 文件保存输出同名点文件，最后完成 Qt 可视化界面的设计和交互。

四、算法原理

1. Harris 特征点提取（Harris.h 文件 HarrisCornerDetect 函数）

（1）计算梯度的平方和乘积：输入一张灰度图像，计算其水平梯度的平方 g_{xx}、垂直梯度的平方 g_{yy} 和梯度的乘积 g_{xy}。

```
Mat gx = Mat::zeros(image.size(), CV_32F);
Mat gy = Mat::zeros(image.size(), CV_32F);
Mat gxx = Mat::zeros(gx.size(), CV_32F);
Mat gyy = Mat::zeros(gy.size(), CV_32F);
Mat gxy = Mat::zeros(gx.size(), CV_32F);
for (int i = 1; i < image.rows - 1; i++) {
    for (int j = 1; j < image.cols - 1; j++) {
        float gradX = image.at<uchar>(i, j - 1) - image.at<uchar>(i, j + 1);
        gx.at<float>(i, j) = abs(gradX);
        float gradY = image.at<uchar>(i + 1, j) - image.at<uchar>(i - 1, j);
        gy.at<float>(i, j) = abs(gradY);
        gxx.at<float>(i, j) = gx.at<float>(i, j) * gx.at<float>(i, j);
        gyy.at<float>(i, j) = gy.at<float>(i, j) * gy.at<float>(i, j);
        gxy.at<float>(i, j) = gx.at<float>(i, j) * gy.at<float>(i, j);
    }
}
```

（2）高斯滤波：创建并计算一个高斯滤波卷积核用于平滑图像的梯度；使用高斯滤波卷积核对 g_{xx}、g_{yy} 和 g_{xy} 进行滤波，得到滤波后的梯度平方和乘积。

```
Mat mask = Mat::zeros(size, size, CV_32F);
int rh = (size - 1) / 2;
int rw = (size - 1) / 2;
float add_up = 0.0;
float x, y;
for (int i = 0; i < size; i++) {
    y = pow(i - rh, 2);
    for (int j = 0; j < size; j++) {
        x = pow(j - rw, 2);
        // 因为最后都要归一化的，常数部分可以不计算，也减少了运算量
        float g = exp(-(x + y) / (2 * sigma * sigma));
        mask.at<float>(i, j) = g;
        add_up += g;
    }
}
mask = mask / add_up;

Mat gxx_Gaussian = Mat::zeros(gxx.size(), CV_32F);
Mat gyy_Gaussian = Mat::zeros(gyy.size(), CV_32F);
Mat gxy_Gaussian = Mat::zeros(gxy.size(), CV_32F);

// 根据mask卷积核高斯滤波
for (int i = rh; i < image.rows - rh; i++) {
    for (int j = rw; j < image.cols - rw; j++) {
        float sum[3] = { 0 };
        for (int r = -rh; r <= rh; r++) {
            for (int c = -rw; c <= rw; c++) {
                sum[0] = sum[0] + gxx.at <float>(i + r, j + c) * mask.at<float>(r + rh, c + rw);
                sum[1] = sum[1] + gyy.at <float>(i + r, j + c) * mask.at<float>(r + rh, c + rw);
                sum[2] = sum[2] + gxy.at <float>(i + r, j + c) * mask.at<float>(r + rh, c + rw);
            }
        }
        gxx_Gaussian.at<float>(i - rh, j - rw) = static_cast<float>(sum[0]);
        gyy_Gaussian.at<float>(i - rh, j - rw) = static_cast<float>(sum[1]);
        gxy_Gaussian.at<float>(i - rh, j - rw) = static_cast<float>(sum[2]);
    }
}
```

（3）计算响应矩阵：对于图像中的每个像素点，使用滤波后的梯度平方和乘积计算 Harris 响应值，计算公式如下。

$$R(i,j) = \det(M) - k \cdot \left(\text{trace}(M)\right)^2$$

其中，M 是由滤波后的梯度平方和乘积构成的矩阵，det(M)是矩阵的行列式，trace(M)是矩阵的迹，k 是一个经验常数。

```
// 计算响应矩阵
Mat response_matrix = Mat::zeros(gxx.size(), CV_32F);
for (int i = 0; i < response_matrix.rows; i++) {
    for (int j = 0; j < response_matrix.cols; j++) {
        float a = gxx_Gaussian.at<float>(i, j);
        float b = gyy_Gaussian.at<float>(i, j);
        float c = gxy_Gaussian.at<float>(i, j);
        response_matrix.at<float>(i, j) = a * b - c * c - k * (a + b) * (a + b);
    }
}
```

（4）非极大值抑制：对于每个响应值，检查其周围邻域内的响应值。如果

当前点的响应值是邻域内的最大值，则保留该点；否则将其抑制。这一步的目的是确保最终的角点是局部最大的响应值，从而提高角点检测的准确性。同时对非极大值抑制后的响应矩阵应用一个阈值（t），只有当响应值大于这个阈值时，该点才被认为是角点。

```
vector<Point2d> respond_points;
int r = (range - 1) / 2;
// 非极大值抑制
for (int i = r; i < response_matrix.rows - r; i++) {
    for (int j = r; j < response_matrix.cols - r; j++) {
        int flag = 1;
        for (int x = -r; x <= r; x++) {
            for (int y = -r; y <= r; y++) {
                if (response_matrix.at<float>(i, j) < response_matrix.at<float>(i + x, j + y)) {
                    flag = 0;
                }
            }
        }
        if (flag == 1 && (int)response_matrix.at<float>(i, j) > t) {
            respond_points.push_back(Point2d(j, i));
        }
    }
}
```

2. 相关系数整像素级匹配

（1）初始化和准备：定义一个结构体 Pair（左像点、右像点、测度）存储匹配点对及其相似度，如右图。

```
struct Pair {
    Point2d lp;
    Point2d rp;
    double similarity;
};
```

（2）计算相似度：通过计算两个窗口的灰度值乘积的总和，然后减去窗口内像素灰度值平均数的乘积，最后除以两个窗口灰度值平方和的平方根来计算。

```
// 计算两个窗口的相关系数
inline double SimilarityCalculate(Mat l_window, Mat r_window) {
    double g1 = 0;
    double g2 = 0;
    double g3 = 0;
    double g4 = 0;
    double g5 = 0;
    double similarity = 0;
    for (int i = 0; i < l_window.rows; i++) {
        for (int j = 0; j < r_window.rows; j++) {
            g1 += l_window.at<uchar>(i, j) * r_window.at<uchar>(i, j);
            g2 += l_window.at<uchar>(i, j) * l_window.at<uchar>(i, j);
            g3 += r_window.at<uchar>(i, j) * r_window.at<uchar>(i, j);
            g4 += l_window.at<uchar>(i, j);
            g5 += r_window.at<uchar>(i, j);
        }
    }
    similarity = (g1 - g4 * g5 / l_window.rows / l_window.cols)
        / sqrt((g2 - g4 * g4 / l_window.rows / l_window.cols) *
            (g3 - g5 * g5 / l_window.rows / l_window.cols));
    return similarity;
}
```

（3）边界检查：检查一个点是否在图像的边界内，如果点太靠近图像边缘（点到边缘距离小于窗口半径），则不进行匹配，因为完整的窗口无法提取。

```
// 判断当前点是否在边界上
inline bool ToCheckPoint(Mat image, Point2d point, int window_size) {
    if (point.x < window_size * 0.5 || point.y < window_size * 0.5 ||
        point.x >= image.cols - window_size * 0.5 || point.y >= image.rows - window_size * 0.5) {
        return false;
    }
    else {
        return true;
    }
}
```

（4）相关系数粗匹配：对于每个图像中的特征点，如果该点不在边界上，则提取以该点为中心的窗口；对于每个窗口，计算其与另一个图像中所有特征点的窗口的相关系数；记录下相关系数最高的匹配，并检查该系数是否大于阈值 t；如果相关系数大于阈值，则将匹配点对和相似度存储在 Pair 结构体中，并将其添加到匹配点对列表中；最后得到一个包含所有匹配点对的 vector<Pair>。

```
vector<Pair> pairs;
if (points_l.size() <= points_r.size()) {
    for (int i = 0; i < points_l.size(); i++) {
        if (ToCheckPoint(img_1, points_l[i], window_size)) {
            Mat window_l = img_1(Rect(points_l[i].x - r, points_l[i].y - r, window_size, window_size));
            double similarity = 0.0;
            double max_similarity = 0.0;
            int max_index = 0;
            for (int j = 0; j < points_r.size(); j++) {
                if (ToCheckPoint(img_2, points_r[j], window_size)) {
                    Mat window_r = img_2(Rect(points_r[j].x - r, points_r[j].y - r, window_size, window_size));
                    similarity = SimilarityCalculate(window_l, window_r);
                    if (similarity > max_similarity) {
                        max_similarity = similarity;
                        max_index = j;
                    }
                }
            }
            // 判断最大的相关系数是否满足设定阈值
            if (max_similarity > t) {
                Pair match;
                match.lp = points_l[i];
                match.rp = points_r[max_index];
                match.similarity = max_similarity;
                pairs.push_back(match);
            }
        }
    }
}
```

（5）绘制匹配结果到图像中：定义 Draw 函数将匹配点对绘制在一个新的图像上，以便于可视化匹配结果。

```
// 绘制提取到的匹配像点
inline Mat Draw(Mat img_1, Mat img_2, vector<Pair> pairs) {
    Mat image;
    image.create(Size(img_1.cols + img_2.cols, max(img_1.rows, img_2.rows)), CV_8UC3);
    img_1.copyTo(image(Rect(0, 0, img_1.cols, img_1.rows)));
    img_2.copyTo(image(Rect(img_1.cols, 0, img_2.cols, img_2.rows)));
    static default_random_engine random_color;
    static uniform_int_distribution<int> range(0, 255);
    for (int i = 0; i < pairs.size(); i++) {
        pairs[i].rp.x = pairs[i].rp.x + img_1.cols;
        Scalar color(range(random_color), range(random_color), range(random_color));
        circle(image, pairs[i].lp, 5, color, 2);
        circle(image, pairs[i].rp, 5, color, 2);
        line(image, pairs[i].lp, pairs[i].rp, color, 2);
    }
    return image;
}
```

3. 基于相关系数的最小二乘精细匹配

（1）初始化参数：遍历粗匹配结果中的每个匹配点对，从匹配点对中提取左右图像的坐标；调整窗口大小，确保窗口大小为奇数；从左图像中提取以粗匹配点为中心的窗口 window_l，创建一个空的 window_r 用于存储右图像中对应窗口的变换后的像素值；设置几何畸变和灰度畸变的初始参数。

```
double y1 = pairs[index].lp.x;
double x1 = pairs[index].lp.y;
double y2 = pairs[index].rp.x;
double x2 = pairs[index].rp.y;
if (window_size % 2 == 0) {
    window_size += 1;
}
int r = window_size / 2;

// 两个待匹配的窗口
Mat window_l = img_1(Rect(y1 - r, x1 - r, window_size, window_size));
Mat window_r;
window_r.create(Size(window_size, window_size), CV_8UC1);

// 几何畸变、灰度畸变初值
double a0 = x2 - x1;
double a1 = 1;
double a2 = 0;
double b0 = y2 - y1;
double b1 = 0;
double b2 = 1;
double h0 = 0;
double h1 = 1;
double x_t = 0.0, y_t = 0.0;
double similarity, max_similarity = 0.0;
Point2d best_point;
```

（2）迭代求解：设定最大迭代次数为 50 次；对于每次迭代，初始化矩阵 A 和向量 L，用于构建线性方程组，遍历左图像窗口中的每个像素点，计算其在右图像中的预测位置，并进行双线性插值来获取灰度值，根据预测位置和灰度值，构建误差方程并填充矩阵 A 和向量 L；如果有效像素点少于 8 个，则跳出迭代，因为无法求解方程组；计算当前窗口的相关系数，求解线性方程组，更新畸变参数，计算最佳匹配点的坐标；如果当前迭代的相关系数大于之前的最大相关系数，则更新最佳匹配点和最大相关系数；如果相关系数大于设定的阈值 t，则将当前匹配点对添加到最终匹配结果中，并跳出迭代。

```
// 开始迭代求解，设定最大迭代次数50次，只匹配得到相关系数大于阈值的点
for (int iter = 0; iter < 50; iter++) {
    Mat A = Mat::zeros(window_size * window_size, 8, CV_32F);
    Mat L = Mat::zeros(window_size * window_size, 1, CV_32F);
    Mat x;
    int num = 0;
    double x_numerator = 0.0, y_numerator = 0.0, x_denominator = 0.0, y_denominator = 0.0;
    for (int i = x1 - r; i <= x1 + r; i++) {
        for (int j = y1 - r; j <= y1 + r; j++) {
            // 几何变形改正
            double m = a0 + a1 * i + a2 * j;
            double n = b0 + b1 * i + b2 * j;

            int m_f = floor(m);
            int n_f = floor(n);

            // 如果点在图像的边界不能进行计算舍弃判断
            if (m_f < 1 || m_f > img_2.rows || n_f < 1 || n_f > img_2.cols)
                continue;

            // 双线性内插重采样
            double gray_scale = (n_f + 1 - n) * ((m_f + 1 - m) * img_2.at<uchar>(m_f, n_f) +
                (m - m_f) * img_2.at<uchar>(m_f + 1, n_f)) + (n - n_f) * ((m_f + 1 - m) * img_2.at<uchar>(m_f, n_f + 1) +
                    (m - m_f) * img_2.at<uchar>(m_f + 1, n_f + 1));

            // 辐射畸变改正
            gray_scale = h0 + h1 * gray_scale;
            window_r.at<uchar>(i - x1 + r, j - y1 + r) = gray_scale;

            // 构建误差方程
            double x_r = 0.5 * (img_2.at<uchar>(m_f + 1, n_f) - img_2.at<uchar>(m_f - 1, n_f));
            double y_r = 0.5 * (img_2.at<uchar>(m_f, n_f + 1) - img_2.at<uchar>(m_f, n_f - 1));
            A.at<float>(num, 0) = 1;
            A.at<float>(num, 1) = gray_scale;
            A.at<float>(num, 2) = x_r;
            A.at<float>(num, 3) = m * x_r;
            A.at<float>(num, 4) = n * x_r;
            A.at<float>(num, 5) = y_r;
            A.at<float>(num, 6) = m * y_r;
            A.at<float>(num, 7) = n * y_r;
            L.at<float>(num, 0) = img_1.at<uchar>(i, j) - gray_scale;

            // 计算最佳匹配点位
            double x_l = 0.5 * (img_1.at<uchar>(i + 1, j) - img_1.at<uchar>(i - 1, j));
            double y_l = 0.5 * (img_1.at<uchar>(i, j + 1) - img_1.at<uchar>(i, j - 1));

            x_numerator += i * x_l * x_l;
            x_denominator += x_l * x_l;
            y_numerator += j * y_l * y_l;
            y_denominator += y_l * y_l;
            num++;
        }
    }
    if (num < 8) // 无法求解法方程
        break;

    similarity = SimilarityCalculate(window_l, window_r);

    // 计算变形参数
    solve(A, L, x, DECOMP_SVD);
```

```
double a0_1 = a0;
double a1_1 = a1;
double a2_1 = a2;
double b0_1 = b0;
double b1_1 = b1;
double b2_1 = b2;
double h0_1 = h0;
double h1_1 = h1;

a0 = a0_1 + x.at<float>(2, 0) + a0_1 * x.at<float>(3, 0) + b0_1 * x.at<float>(4, 0);
a1 = a1_1 + a1_1 * x.at<float>(3, 0) + b1_1 * x.at<float>(4, 0);
a2 = a2_1 + a2_1 * x.at<float>(3, 0) + b2_1 * x.at<float>(4, 0);
b0 = b0_1 + x.at<float>(5, 0) + a0_1 * x.at<float>(6, 0) + b0_1 * x.at<float>(7, 0);
b1 = b1_1 + a1_1 * x.at<float>(6, 0) + b1_1 * x.at<float>(7, 0);
b2 = b2_1 + a2_1 * x.at<float>(6, 0) + b2_1 * x.at<float>(7, 0);
h0 = h0_1 + x.at<float>(0, 0) + h0_1 * x.at<float>(1, 0);
h1 = h1_1 + h1_1 * x.at<float>(1, 0);

// 计算最佳匹配点位
double x_iter = x_numerator / x_denominator;
double y_iter = y_numerator / y_denominator;

x_t = a0 + a1 * x_iter + a2 * y_iter;
y_t = b0 + b1 * x_iter + b2 * y_iter;

if (similarity > max_similarity) {
    best_point.x = y_t;
    best_point.y = x_t;
    max_similarity = similarity;
}
if (max_similarity > t) {
    Pair pair;
    pair.lp = pairs[index].lp;
    pair.rp = best_point;
    pair.similarity = max_similarity;
    ls_pairs.push_back(pair);
    break;
}
```

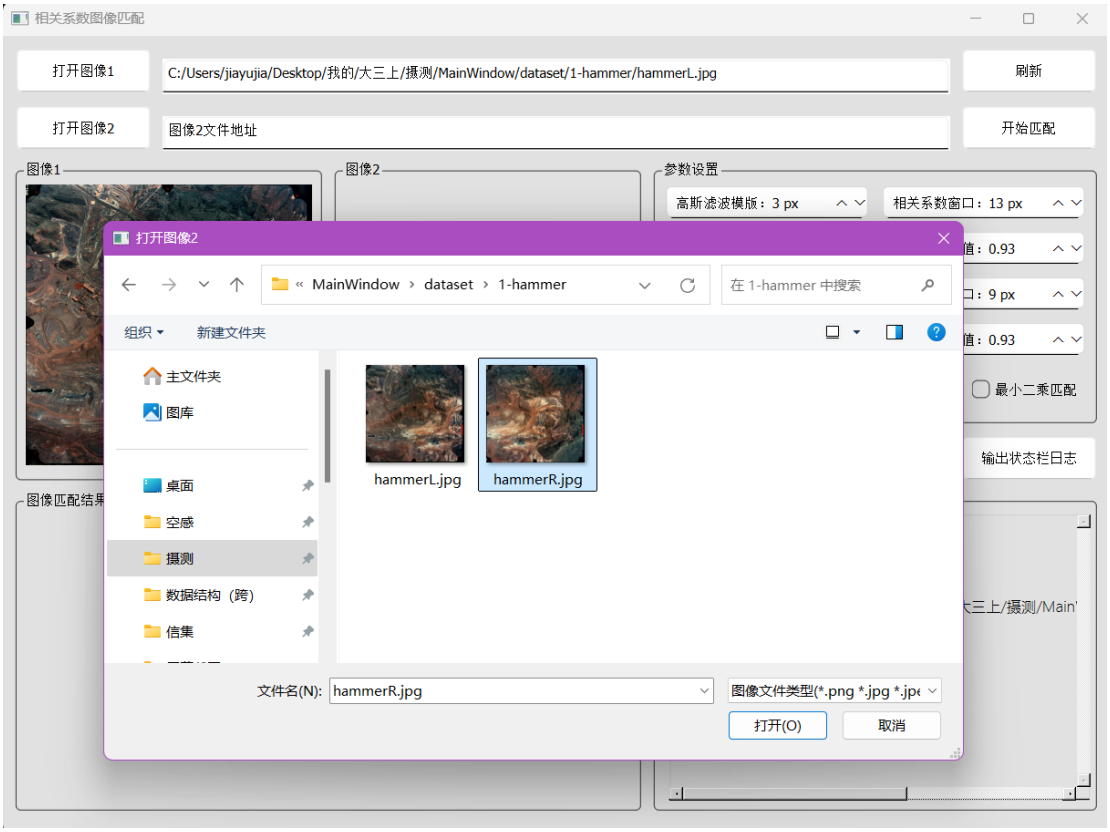
五、界面 GUI 设计

如图，图形界面共分为 5 大区域：影像文件控制区①、影像匹配展示区②、匹

配参数设置区③、匹配控制区④及命令执行状态显示区⑤。

1. 影像文件控制区

选择打开图像后，会弹出选择文件框，这个自动返回所选图像文件的路径，在代码层级会读入该图像的彩色及灰度影像，并在程序中对彩色影像进行显示。



成功读取文件后，会在上方文本框显示图像文件路径，在命令执行状态显示区显示图像的基础信息；只有在读入两张图像后，才能进行后续的匹配、输出文件等操作。

2. 匹配参数设置区

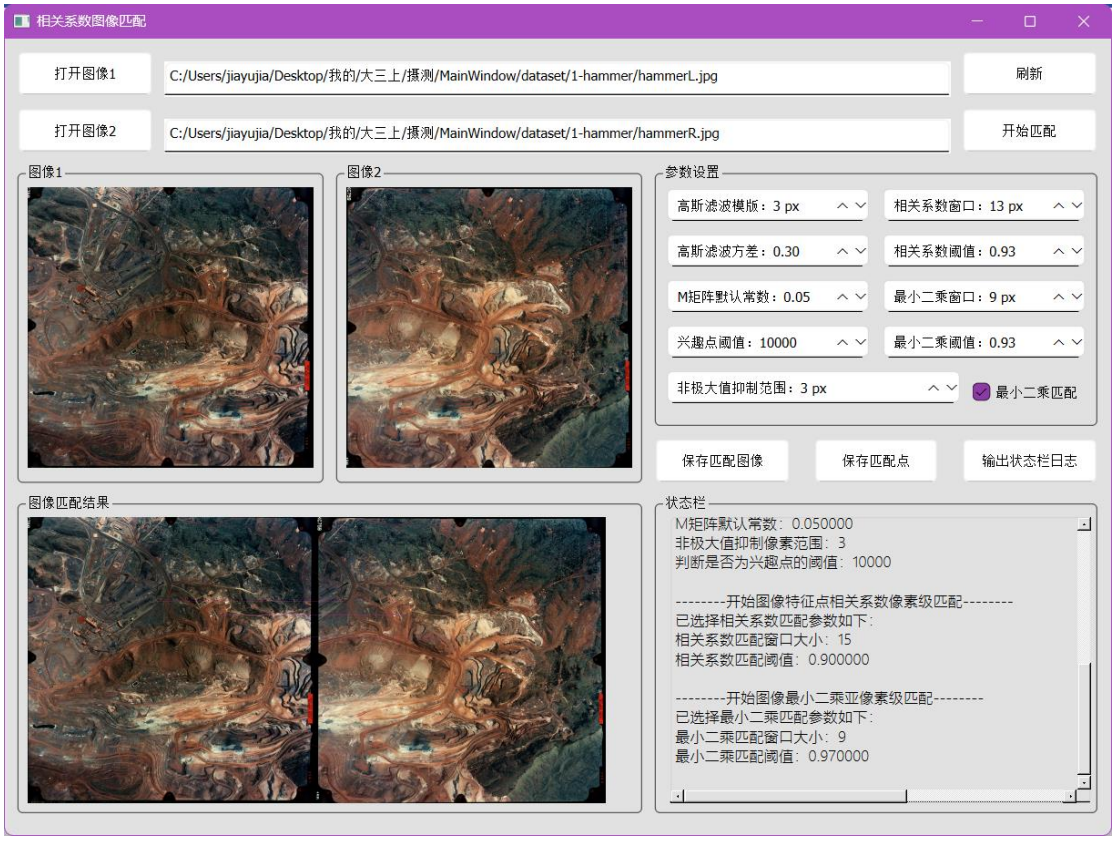
在参数设置区，可以通过改变输入框的值改变匹配相关参数。这些匹配参数设置了最大值和最小值、改变的步长等，每一次参数的改正都会在命令执行状态显示区显示参数的变化以提示使用。

3. 匹配控制区

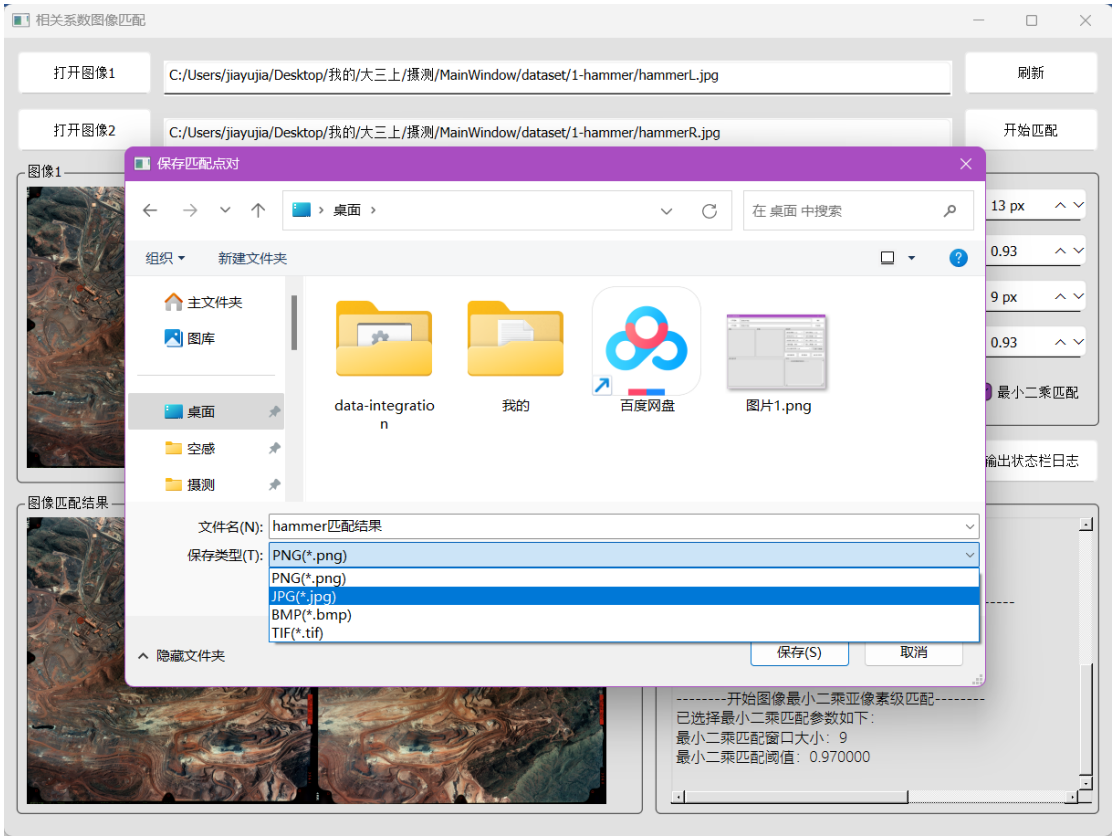
这一部分是一系列控制匹配的开始、刷新、保存匹配图像、保存匹配点、输出状态栏日志等开始执行功能的按钮。

“开始匹配”按钮会根据当前两张图像及设置的参数进行图像匹配，如果两张图像有一张为打开，在命令执行状态显示区显示提示信息；匹配完成后会

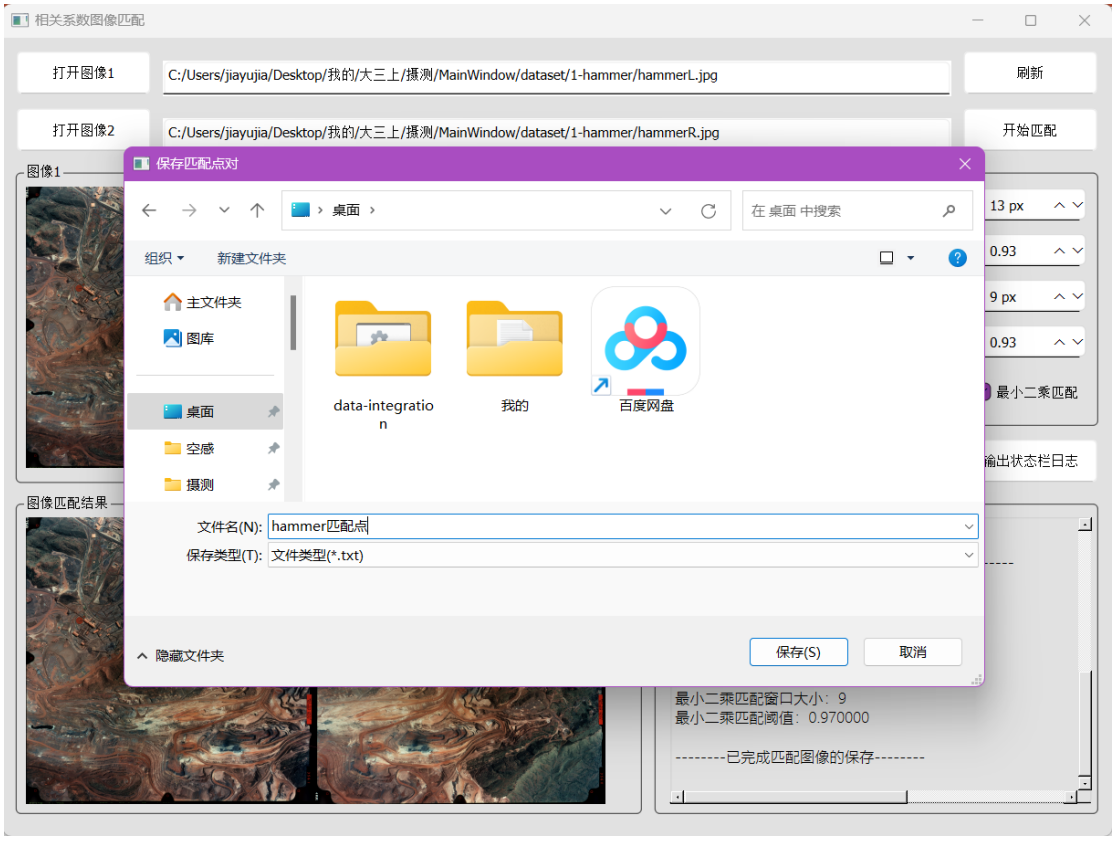
在影像匹配展示区显示匹配结果，同名点用随机颜色的线条相连接。

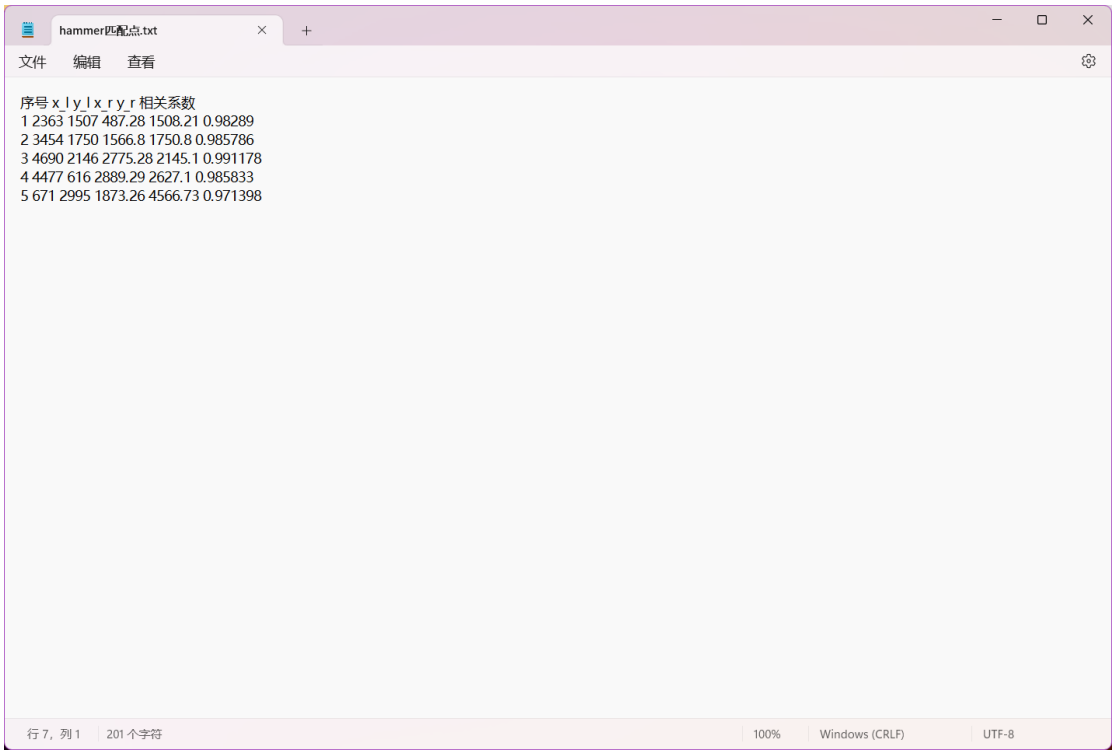


“保存匹配图像”按钮可以选择匹配好的图像保存的路径、格式和名称。

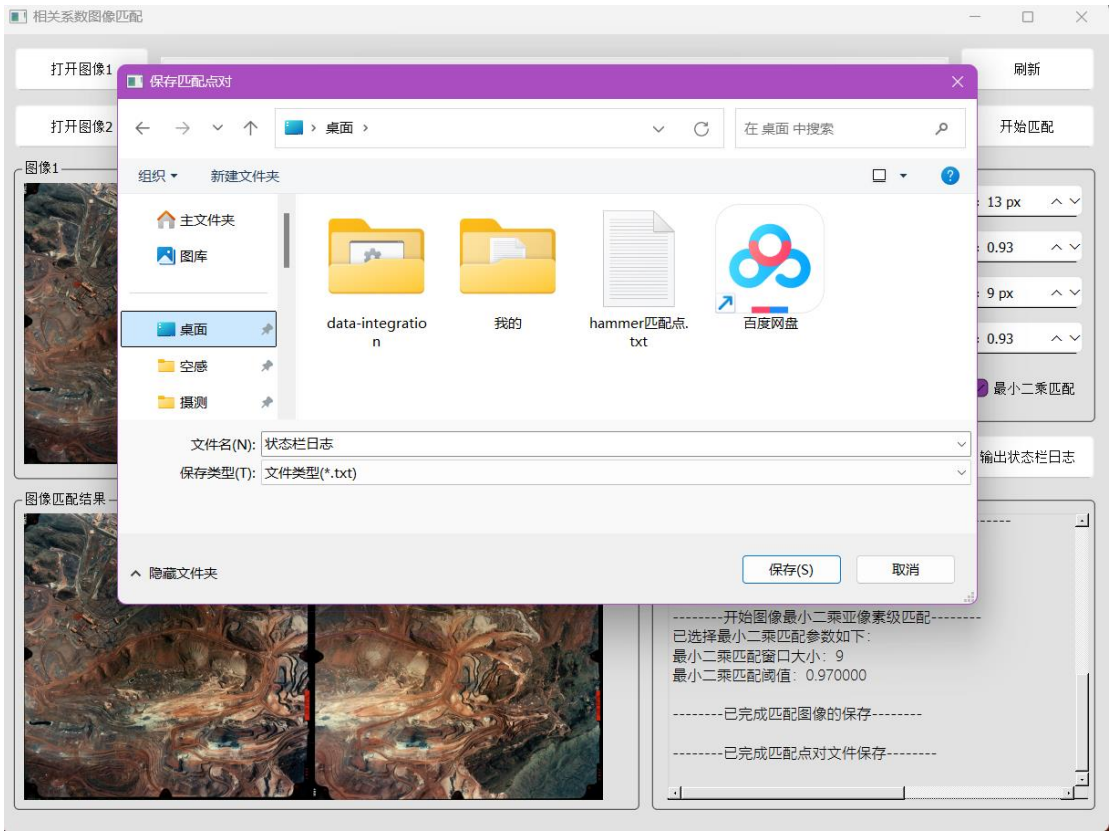


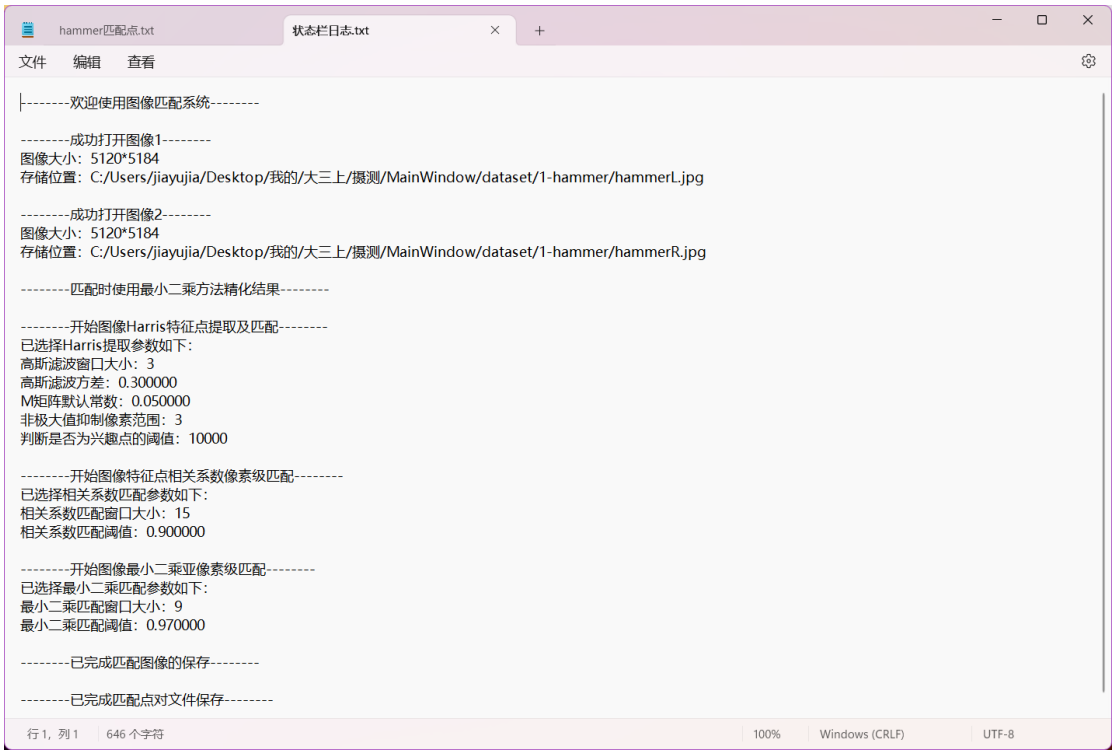
“保存匹配点”按钮命名文件名称和路径，将匹配点按 txt 文件格式输出。





“输出状态栏日志”可以将当前命令执行状态显示区的所有提示信息输出到 txt 文档中，与“保存匹配点”的文件命名、路径选择方式相同。



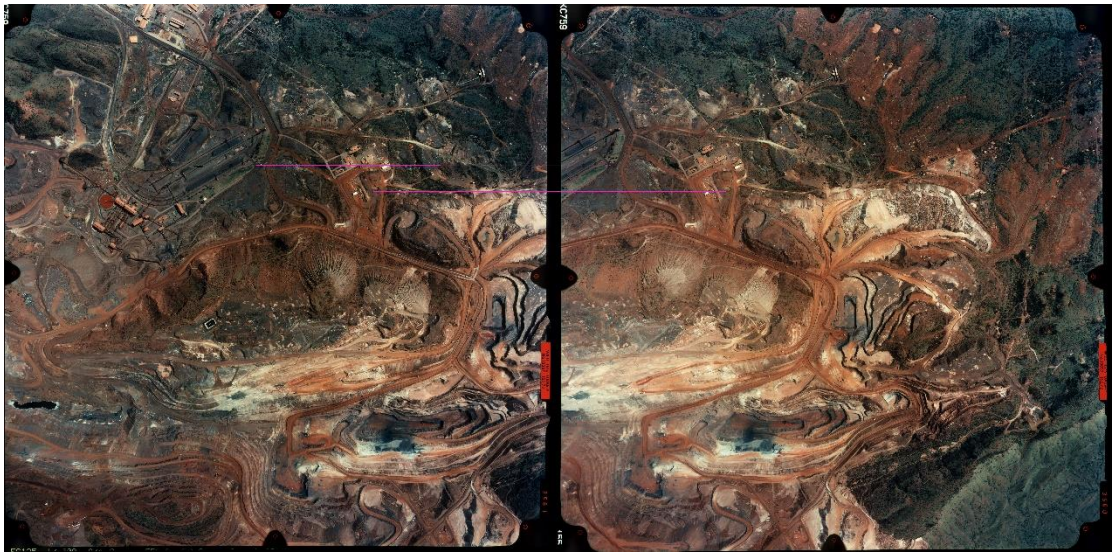


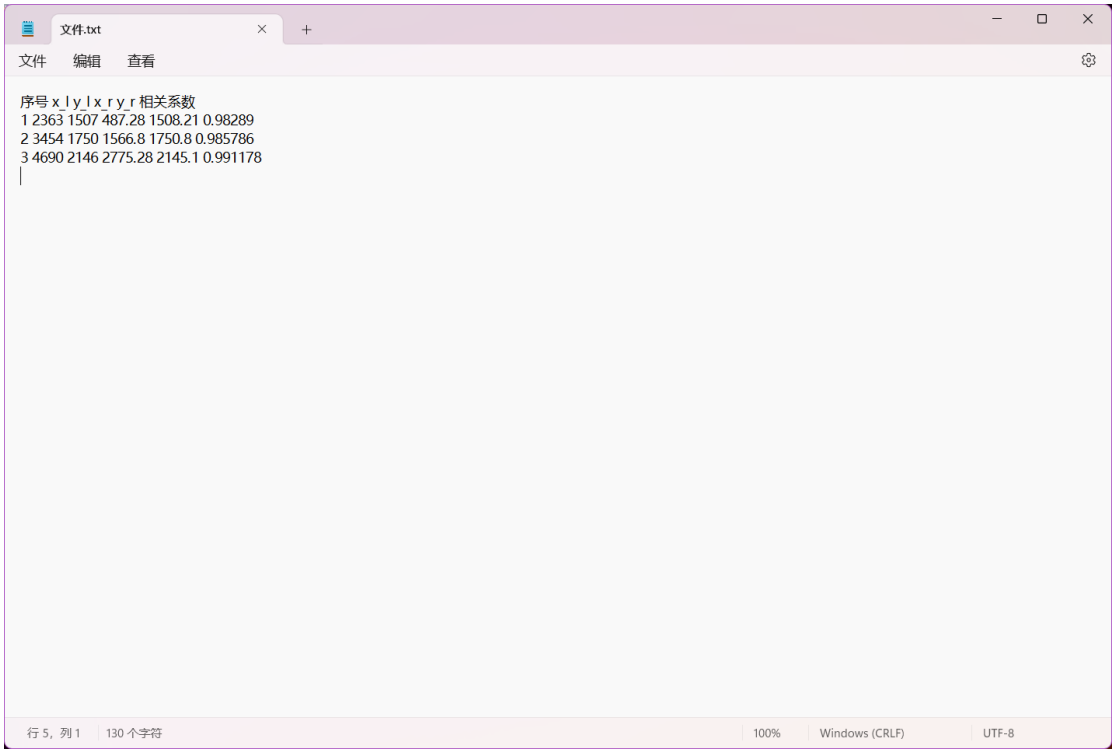
“刷新”按钮会将程序重置为程序刚打开的状态，即重置图像路径清除，已打开或匹配的图像，将参数设置为初始状态，清空命令执行状态显示区。

六、成果展示

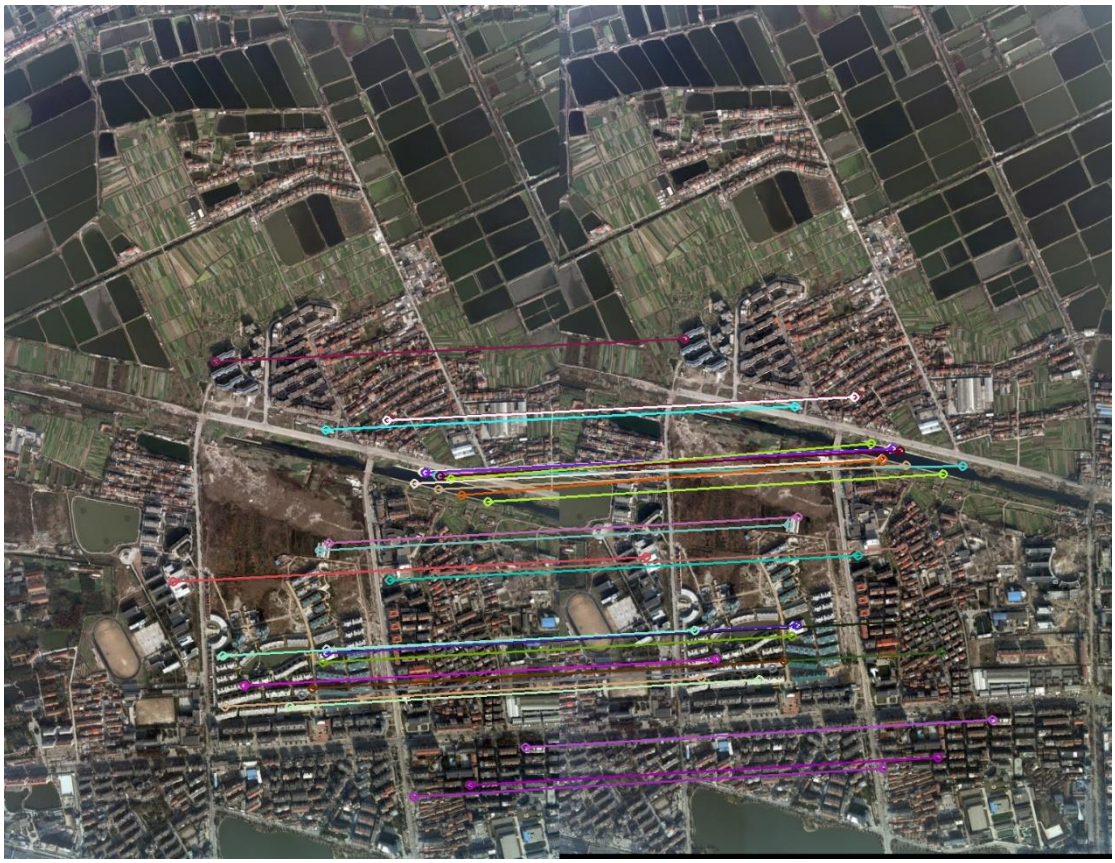
给的两对影像数据的匹配结果如下。

1. hammer





2. town



town.txt

文件 编辑 查看

序号 x_l y_l x_r y_r 相关系数
1 291 490 172.214 458.615 0.983081
2 532 570 410.909 537.195 0.984121
3 527 571 406.159 538.522 0.978654
4 443 584 323.559 551.628 0.984231
5 576 642 455.815 609.548 0.975102
6 580 643 458.592 610.195 0.970166
7 596 647 554.907 634.159 0.975257
8 600 648 468.583 612.916 0.971027
9 615 652 428.663 602.952 0.976283
10 565 658 443.984 624.413 0.984142
11 598 666 477.02 632.571 0.981647
12 630 674 445.015 624.941 0.975263
13 666 683 528.456 645.204 0.977188
14 447 740 328.373 703.907 0.973846
15 434 750 314.524 715.341 0.978746
16 594 782 472.55 747.231 0.977096
17 527 791 405.866 757.153 0.988667
18 531 791 410.794 757.35 0.979569
19 233 793 118.679 759.392 0.985311
20 629 881 507.986 845.628 0.974284
21 444 886 322.334 853.812 0.991221
22 443 892 326.561 853.215 0.981419
23 301 895 186.231 859.711 0.980301
24 439 904 320.449 867.54 0.991246
25 645 926 524.758 889.972 0.974268
26 331 936 215.2 899.878 0.980535
27 424 940 308.575 902.432 0.970599
28 410 949 292.52 911.784 0.988254
29 304 965 195.194 925.809 0.971949
30 393 965 275.231 927.624 0.976184
31 718 1021 595.818 982.607 0.970142
32 642 1073 521.254 1035.29 0.972096
33 564 1089 445.189 1049.5 0.984799

行 1，列 1 1,198 个字符 100% Windows (CRLF) UTF-8

七、实习总结

在本次实习项目中，我的主要任务是开发一个能够实现相关系数匹配和最小

二乘匹配的程序。这个项目不仅让我更深入地理解了这两种统计方法，而且也锻炼了我的C/C++编程技能。

首先，我实现了相关系数匹配功能，通过计算两个图像的相关性来确定它们是否匹配。接着，我在此基础上进一步开发了最小二乘匹配算法，以优化匹配结果。这个过程涉及到构建一个线性模型，并通过最小化误差的平方和来找到最佳拟合，这显著提高了匹配的精确度。

为了提高程序的可用性，我还设计了一个用户界面，允许用户选择文件路径和设置匹配参数。这不仅提高了用户体验，也使得程序更加灵活和易于操作。开发程序能够输出包含匹配结果的同名点文件，这样用户可以方便地查看和分析匹配结果。

在项目过程中，我遇到了一些挑战。算法理解上的困难通过阅读大量相关文献和观看在线教程得到了解决。编程挑战，如内存管理和算法优化，通过不断实践和向有经验的学长请教而逐渐克服。用户界面设计对我来说是一个新领域，但通过学习 GUI 设计原则和使用 C++的 Qt 框架，我成功地创建了一个直观且功能齐全的用户界面。调试和测试过程中遇到的 bug 通过编写单元测试和使用调试工具得到了解决；时间管理上的挑战则通过制定详细的时间表，并严格按照计划执行来克服，确保了作业的按时完成。