

特征点提取及相关系数法匹配编程 实习报告报告

班级： 2022 级 A3 班

姓名： 阮思琳

学号： 2022302131290

二〇二四年十月

目 录

一 概述.....	1
1.1 实习的目的和意义	1
1.2 实习数据	1
1.3 编程环境	1
二 特征点提取算法原理及算法	2
2.1 MORAVEC 特征点提取	2
2.2 FORSTNER 特征点提取	2
三 相关系数法影像匹配原理及实用算法	3
3.1 算法概述	3
3.2 算法步骤	4
四 编程思路及流程图	4
4.1 编程思路	4
4.1.1 基于 Moravec 算法特征点提取、存储、显示	5
4.1.2 基于 Forstner 算法特征点提取、存储、显示	6
4.1.3 图像金字塔构建	8
4.1.4 基于相关系数法影像匹配、显示、存储	9
4.2 编程流程图	12
4.3 结果展示	13
五 试验结果分析	15
5.1 特征点提取算法	15
5.1.1 阈值设定分析	15
5.1.2 随机分布与均匀分布提取特征点的实施分析	16
5.2 相关系数法影像匹配算法	17
5.2.1 算法分析	17
5.2.2 窗口大小对匹配的影响	17
六 心得体会	19

一 概述

1.1 实习的目的和意义

实践编程技能与特征匹配算法的结合: 本次实践聚焦于通过编写一套基于特定编程环境的图像处理程序,将编程技巧与计算机视觉中的特征提取与匹配算法深度融合。核心目标是掌握影像的基本读取与显示操作,这是进行后续复杂图像分析的前提。在此基础上,深入学习并应用多种常见的特征点提取算法,不仅能有效识别图像中的关键点,还能直观地展示这些特征点,从而加深对特征点提取技术的理解。

深化特征点提取与匹配算法的理论认知与编程实践: 深入探讨兴趣值窗口与抑制窗口大小如何影响特征点提取的精度和效率,通过编程实验,直观展示不同参数设置下的效果差异,这不仅深化了对特征点检测原理的认识,也锻炼了编程实现与优化算法的能力。进一步地,掌握基于相关系数法的图像匹配方法,理解其工作原理,并通过可视化手段直观展示匹配结果,同时深入探究窗口大小及相关系数阈值对匹配精度和鲁棒性的影响。

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。

1.2 实习数据

本次实习的影像为航空摄影立体影像对,影像的格式有 TIF 文件、BMP 文件、.RAW 文件,实习选用其中一种格式影像文件即可,本报告选用的 TIF 文件格式。左影像名称为“u0369_panLeft”,尺寸大小为 1023×942 ;右影像名称为“u0367_panRight”,尺寸大小为 805×887 。如下图所示:

1.3 编程环境

本次实习的编程环境为 Visual Studio 2022 C++,同时配置了 opencv 库及其扩展模块,以便于更轻松地进行图像操作和矩阵计算。

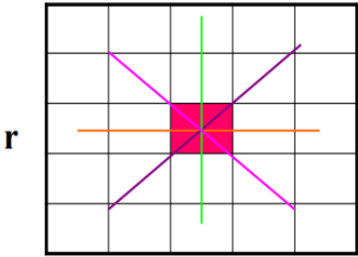
二 特征点提取算法原理及算法

2.1 Moravec 特征点提取

概述：一种基于灰度方差的角点检测算子，该算子计算图像中每个像素点沿着水平、垂直、对角线及反对角线的四个方向的灰度方差，其中的最小值选作该像素点的兴趣值 IV，再通过局部非极大值抑制来检测其是否为特征点（角点）。

算法步骤：

- 1> 计算各像元的兴趣值 IV：以像素（c,r）为中心的 $w * w$ 影像窗口中，计算四个方向（垂直、水平、正对角线、反对角线）相邻像素灰度差的平方和，去四个方向的最小值为（c，r）像点的 IV

$$\begin{aligned} V1 &= \sum_{i=-k}^{k-1} (g_{c+i,r} - g_{c+i+1,r})^2 \\ V2 &= \sum_{i=-k}^{k-1} (g_{c+i,r+i} - g_{c+i+1,r+i+1})^2 \\ V3 &= \sum_{i=-k}^{k-1} (g_{c,r+i} - g_{c,r+i+1})^2 \\ V4 &= \sum_{i=-k}^{k-1} (g_{c+i,r-i} - g_{c+i+1,r-i-1})^2 \\ IV_{c,r} &= \min\{V1, V2, V3, V4\} \end{aligned}$$


CSDN @gorgeous秋

图 2.1 Moravec 提取兴趣值 IV

- 2> 得到候选点：给定经验阈值，将兴趣值大于阈值的点作为候选点（阈值的选择应该以候选点中包括所需要的特征点而又不含过多非特征点为原则）。
- 3> 非极大值抑制：在一定的抑制窗口内，将候选点中兴趣值最大的点保留作为特征点，其他点去掉。

2.2 Forstner 特征点提取

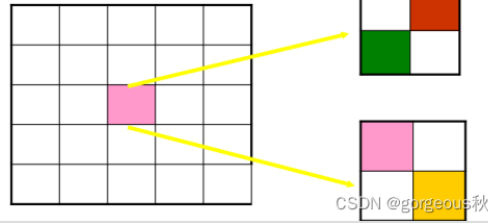
概述：通过计算各像素的 Robert's 梯度和像素（c，r）为中心的窗口内的灰度协方差矩阵，在影像中寻找具有尽可能小而接近圆的误差椭圆的点作为特征点。其特点是速度快、精度较高。

算法步骤：

- 1> 计算个像素的 Robert's 梯度

$$g_u = \frac{\partial g}{\partial u} = g_{i+1,j+1} - g_{i,j}$$

$$g_v = \frac{\partial g}{\partial v} = g_{i,j+1} - g_{i+1,j}$$



2> 计算 $I \times I$ 窗口中的灰度协方差矩阵:

$$\begin{aligned} \sum g_u^2 &= \sum_{i=c-k}^{c+k-1} \sum_{j=r-k}^{r+k-1} (g_{i+1,j+1} - g_{i,j})^2 \\ \sum g_v^2 &= \sum_{i=c-k}^{c+k-1} \sum_{j=r-k}^{r+k-1} (g_{i,j+1} - g_{i+1,j})^2 \\ \sum g_u g_v &= \sum_{i=c-k}^{c+k-1} \sum_{j=r-k}^{r+k-1} (g_{i+1,j+1} - g_{i,j})(g_{i,j+1} - g_{i+1,j}) \\ N &= \begin{bmatrix} \sum g_u^2 & \sum g_u g_v \\ \sum g_u g_v & \sum g_v^2 \end{bmatrix} \end{aligned}$$

3> 计算兴趣值 q 、 w :

$$\omega = \frac{1}{\text{tr}Q} = \frac{\text{Det}N}{\text{tr}N}$$

$$q = \frac{4\text{Det}N}{(\text{tr}N)^2}$$

4> 确定候选点: 其中 \bar{w} 为权平均值, w_c 为权中值

$$T_q = 0.5 \sim 0.75$$

$$T_w = \begin{cases} f\bar{w} & (f = 0.5 \sim 1.5) \\ cw_c & \end{cases}$$

5> 非极大值抑制: 在一个窗口内, 选取权值 w 最大的候选点作为特征点。

三 相关系数法影像匹配原理及实用算法

3.1 算法概述

相关系数是最早由统计学家卡尔·皮尔逊设计的统计指标, 是研究变量之间线性相关程度的量, 一般用字母 r 表示。由于研究对象的不同, 相关系数有多种定义方式, 较为常用的是皮尔逊相关系数。相关关系是一种非确定性的关系, 相关系数研究变量之间线性相关程度的量, 简单相关系数定义方式如下:

$$r(X,Y) = \frac{Cov(X,Y)}{\sqrt{Var[X]Var[Y]}}$$

其中 $Cov(X,Y)$ 为 X 与 Y 的协方差, $Var[X]$ 为 X 的方差, $Var[Y]$ 为 Y 的方差。

3.2 算法步骤

- 1> 读取左右影像
- 2> 确定目标点位置：使用 Moravec 算子或 Forstner 算子对左影像进行特征点提取，将提取的特征点作为目标点（具体提取步骤见上）；
- 3> 预测右影像的搜索范围：通过图像金字塔逐层搜索对应点的位置，设置搜索范围和窗口大小的参数实现精确定位；
- 4> 逐窗口计算相关系数并保存：计算窗口内左影像与右影像灰度的相关系数，并且保存起来，相关系数大于阈值时记录为匹配点对。公式如下：

$$\rho(c,r) = \frac{\sum_{i=1}^m \sum_{j=1}^n (g_{i,j} - \bar{g})(g_{i+r,j+c}' - \bar{g}')}{\sqrt{\sum_{i=1}^m \sum_{j=1}^n (g_{i,j} - \bar{g})^2 \cdot \sum_{i=1}^m \sum_{j=1}^n (g_{i+r,j+c}' - \bar{g}')^2}}$$

$$\bar{g} = \frac{1}{m \cdot n} \sum_{i=1}^m \sum_{j=1}^n g_{i,j}$$

$$\bar{g}' = \frac{1}{m \cdot n} \sum_{i=1}^m \sum_{j=1}^n g_{i+r,j+c}'$$

核心思路为：利用与目标窗口大小相等的匹配窗口在搜索影像的规定搜索区域内连续滑动，并计算目标窗口与搜索窗口的相关系数，相关系数最大的窗口所对应的中心像素点即为与目标窗口相匹配的同名点。

四 编程思路及流程图

4.1 编程思路

使用 Moravec 和 Forstner 两种角点检测方法从图像中提取特征点，然后通过多层图像金字塔和归一化互相关（NCC）在左右图像间进行特征匹配，最后在匹配图像上显示对应的关键点和连接线。

4.1.1 基于 Moravec 算法特征点提取、存储、显示

Moravec 算法通过计算各方向窗口差异最小值作为兴趣值，并通过极值点抑制得到角点。使用 `drawKeypoints` 在左图上标记角点。

以下是按您的描述和要求编写的 Moravec 算子特征点提取算法的思路，并嵌入了您代码中的关键代码部分。这段代码会依次执行参数设置、遍历图像、计算兴趣值、应用阈值筛选、抑制窗口操作，最后提取并标注局部极值特征点。

1. 设置参数：通过指定参数 `'window_size'`、`'threshold'` 和 `'win_size2'`，分别定义兴趣值窗口大小、阈值和抑制窗口大小。这些参数可以根据实验或经验值调整，以优化特征点提取效果。

2. 遍历图像：使用双层 `for` 循环逐个遍历图像像素，并在每个位置定义一个窗口 `window`（大小为 $2*n+1$ ），提取该窗口用于后续兴趣值计算。

3. 兴趣值计算：对于每个窗口，从垂直、水平、对角、反对角四个方向计算灰度差的平方和，将其中的最小值作为该像素的兴趣值。此最小值能够表征该位置是否有角点特征。

```
float V[4] = { 0 };
for (int i = -k; i < k; i++) {
    V[0] += pow(image.at<float>(h, w + i) - image.at<float>(h, w + i + 1), 2);
    V[1] += pow(image.at<float>(h + i, w + i) - image.at<float>(h + i + 1, w + i + 1), 2);
    V[2] += pow(image.at<float>(h + i, w) - image.at<float>(h + i + 1, w), 2);
    V[3] += pow(image.at<float>(h - i, w + i) - image.at<float>(h - i - 1, w + i + 1), 2);
}
float iv = *min_element(V, V + 4); // 取最小值
corners.at<float>(h, w) = iv;
```

4. 阈值筛选：依据设定的阈值 `'threshold'`，筛选出兴趣值高于该阈值的特征点位置，记录符合条件的位置并统计符合要求的候选特征点数目。也会保留未筛选的兴趣值用于后续分析。

```
if (corners.at<float>(r, c) > threshold) {
    countchoosen++;
    continue;
}
else
    corners.at<float>(r, c) = 0;
```

5. 抑制窗口控制：利用大小为`win_size2`的抑制窗口进行密集特征点的抑制，确保特征点的均匀分布。在每个抑制窗口内选择局部最大非零值作为特征点，并在图像上标记特征点的位置，以便在可视化中识别它们。

```
Mat windowCorners;
Point maxIdx;
double maxVal;
for (int h = win_size2 / 2; h < height - win_size2 / 2; h += win_size2) {
    for (int w = win_size2 / 2; w < width - win_size2 / 2; w += win_size2) {
        windowCorners = corners(Rect(w - win_size2 / 2, h - win_size2 / 2,
win_size2, win_size2));
        minMaxLoc(windowCorners, NULL, &maxVal, NULL, &maxIdx);
        if (maxVal > 0) {
            Point imgLoc(maxIdx.x + (w - win_size2 / 2), maxIdx.y + (h - win_size2
/ 2));
            keypoints.push_back(KeyPoint(imgLoc.x, imgLoc.y, 1));
        }
    }
}
```

6. 输出与可视化：输出提取到的特征点数量，通过`drawKeypoints`函数绘制特征点，便于评估算法效果。

```
cout << "Number of keypoints in left image: " << keypointsL.size() << endl;
Mat Image_Kp;
imgL.convertTo(Image_Kp, CV_8UC1, 255);
drawKeypoints(Image_Kp, keypointsL, Image_Kp, Scalar(255, 0, 0),
DrawMatchesFlags::DEFAULT);
namedWindow("imageWithKeypoints", WINDOW_NORMAL);
imshow("imageWithKeypoints", Image_Kp);
waitKey(0);
```

4.1.2 基于 Forstner 算法特征点提取、存储、显示

Forstner 算法通过窗口协方差矩阵的权值和圆度检测角点，同样进行极值点抑制。使用`drawKeypoints`在左图上标记角点。

以下是 Forstner 算子特征点提取的编程思路，包含了代码中的关键代码部分。这段代码包含参数设置、梯度计算、协方差矩阵计算、加权特征值判断、抑制窗口操作和最终特征点的提取与标注。

1. 设置参数：设置兴趣窗口大小、极值点抑制窗口、阈值等参数，定义 Forstner 算法中需要的特征点提取窗口大小和阈值`Tw`和`Tq`，用于筛选符合特

征条件的候选点。

2. 梯度计算：使用 Robert 算子计算图像的 x 方向和 y 方向梯度。为此，构建`kernelX`和`kernelY`两个 2x2 的卷积核，并通过`filter2D`函数对图像进行卷积操作，得到图像的水平（`Ix`）和垂直梯度（`Iy`）。

```
Mat kernelX = (Mat_<float>(2, 2) << 1, 0, 0, -1);
Mat kernelY = (Mat_<float>(2, 2) << 0, 1, -1, 0);

filter2D(image, Ix, CV_32F, kernelX);
filter2D(image, Iy, CV_32F, kernelY);
```

3. 协方差矩阵计算：遍历图像每个像素，并根据兴趣值窗口大小`2*n+1`定义区域窗口`window`，在该窗口内计算梯度的平方和，形成协方差矩阵的元素`Ixx`、`Iyy`和`Ixy`。基于这些矩阵元素，计算矩阵的行列式（`det`）和迹（`trace`）用于后续判断。

```
Mat windowIx = Ix(Rect(w - window_size / 2, h - window_size / 2, window_size,
window_size));
Mat windowIy = Iy(Rect(w - window_size / 2, h - window_size / 2, window_size,
window_size));

Mat Ixx, Ixy, Iyy;
multiply(windowIx, windowIx, Ixx);
multiply(windowIy, windowIy, Iyy);
multiply(windowIx, windowIy, Ixy);

double sumIxx = sum(Ixx)[0];
double sumIyy = sum(Iyy)[0];
double sumIxy = sum(Ixy)[0];

double det = (sumIxx * sumIyy) - (sumIxy * sumIxy);
double trace = sumIxx + sumIyy;
```

4. 计算加权特征值判断：根据协方差矩阵的`det`和`trace`，计算两个特征值指标：`ww`表示加权特征值，`q`表示矩形度指标。以`ww`反映特征强度、`q`反映角点结构。对于`trace`为零的情况跳过该点，并将计算结果存入候选点矩阵`corners_w`和`corners_q`中。

```
double ww = det / trace;
double q = 4 * det / (trace * trace);

corners_w.at<float>(h, w) = static_cast<float>(ww);
corners_q.at<float>(h, w) = static_cast<float>(q);
```

5. 特征点筛选：设置`Tw`和`Tq`两个阈值，将`corners_w`和`corners_q`中满足`ww > Tw`且`q > Tq`的点标记为候选特征点。在筛选过程中统计候选点数量。

```
if (corners_w.at<float>(r, c) > Tw && corners_q.at<float>(r, c) > Tq) {  
    countchoosen++;  
    continue;  
} else  
    corners_w.at<float>(r, c) = 0;
```

6. 抑制窗口操作：采用大小为`win_size2`的抑制窗口，从候选点中选择局部最大值以确保均匀分布。遍历候选特征点矩阵，并在抑制窗口中选择局部极值，最后将这些极值点存储为特征点，绘制在图像上。

```
Mat windowCorners;  
Point maxIdx;  
double maxVal;  
for (int h = win_size2 / 2; h < height - win_size2 / 2; h = h + win_size2) {  
    for (int w = win_size2 / 2; w < width - win_size2 / 2; w = w + win_size2) {  
        windowCorners = corners_w(Rect(w - win_size2 / 2, h - win_size2 / 2,  
win_size2, win_size2));  
        minMaxLoc(windowCorners, NULL, &maxVal, NULL, &maxIdx);  
        if (maxVal > 0) {  
            Point imgLoc(maxIdx.x + (w - win_size2 / 2), maxIdx.y + (h - win_size2  
/ 2));  
            keypoints.push_back(KeyPoint(imgLoc.x, imgLoc.y, 1));  
        }  
    }  
}
```

7. 输出与可视化：输出总特征点数量，便于对结果的直观分析，具体同Moravec 特征点的显示。

4.1.3 图像金字塔构建

使用多层图像金字塔（金字塔缩放图像）减少计算量并提高匹配的鲁棒性。从最低分辨率层开始匹配，逐层向上匹配到更高分辨率。

```
vector<Mat> pyramidL, pyramidR;  
Mat currentL = imgL, currentR = imgR;  
  
for (int i = 0; i < 5; i++) {  
    pyramidL.push_back(currentL);  
    pyramidR.push_back(currentR);  
    pyrDown(currentL, currentL);  
    pyrDown(currentR, currentR);  
}
```

```
pyrDown(currentR, currentR);  
}
```

4.1.4 基于相关系数法影像匹配、显示、存储

定义基于归一化相关系数（NCC）的相似性测度 **calculateNCC 函数**来评估图像块之间的相似度。

```
double calculateNCC(const Mat& blockA, const Mat& blockB) {  
    // 确保两个块大小一致  
    if (blockA.size() != blockB.size()) {  
        return -1;  
    }  
    // 均值计算  
    Scalar meanA = mean(blockA);  
    Scalar meanB = mean(blockB);  
    // 去均值  
    Mat A = blockA - meanA;  
    Mat B = blockB - meanB;  
    // 计算分子 (A 与 B 的点积)  
    double numerator = sum(A.mul(B))[0];  
    // 计算分母 (A 和 B 的范数乘积)  
    double denominator = sqrt(sum(A.mul(A))[0]) * sqrt(sum(B.mul(B))[0]);  
    // 防止除以零  
    if (denominator < 1e-10) {  
        return 0;  
    }  
    return numerator / denominator;  
}
```

findBestMatch 函数在给定搜索范围内寻找最佳匹配点。

1. 设置参数: 设置图像块大小和搜索半径。计算左图特征点的图像块, 并准备存储最佳匹配点的变量。使用`Rect`定义左图的窗口, 并确保其在图像边界内。

```
Rect windowL(ptL.x - windowSize.width / 2, ptL.y - windowSize.height / 2,  
windowSize.width, windowSize.height);  
windowL &= Rect(0, 0, imgL.cols, imgL.rows); // 限制窗口在图像范围内  
Mat blockL = imgL(windowL);
```

2. 遍历搜索范围: 在右图中通过外层和内层循环遍历指定的搜索半径范围, 遍历所有可能的点, 通过逐个移动计算当前匹配点的 NCC 值。

3. 计算 NCC: 使用`calculateNCC`函数计算左图和右图当前图像块的归一化互相关系数, 并根据 NCC 值更新最佳匹配点和最佳 NCC。记录最佳匹配点和其

NCC 值。

```
for (int dx = -searchRadius; dx <= searchRadius; ++dx) {
    for (int dy = -searchRadius; dy <= searchRadius; ++dy) {
        Point2f ptR = initialPtR + Point2f(dx, dy);
        // 提取右图的图像块
        Rect windowR(ptR.x - windowSize.width / 2, ptR.y - windowSize.height / 2,
            windowSize.width, windowSize.height);
        windowR &= Rect(0, 0, imgR.cols, imgR.rows); // 限制窗口在图像范围内

        // 如果窗口超出图像范围, 跳过
        if (windowR.width != windowSize.width || windowR.height !=
            windowSize.height) {
            continue;
        }
        Mat blockR = imgR(windowR);
        // 计算相关系数
        double ncc = calculateNCC(blockL, blockR);
        // 寻找最大相关系数的匹配点
        if (ncc > bestNCC) {
            bestNCC = ncc;
            bestMatch = ptR;
        }
    }
}
```

4. 返回最佳匹配: 返回最佳匹配点的坐标和其对应的 NCC 值, 方便后续处理或显示。

程序从**金字塔**最低分辨率开始逐个匹配特征点, 然后逐层调整匹配点, 以获取更高分辨率下的精确坐标。

```
for (size_t i = 0; i < keypointsL.size(); i++) {
    Point2f ptL = keypointsL[i].pt; // 左图特征点
    Point2f ptR = findBestMatch(pyramidL.back(), pyramidR.back(), ptL / (1 <<
        (pyramidL.size() - 1)), ptL / (1 << (pyramidL.size() - 1)), windowSize,
        searchRadius, ncc);

    float scaleFactor = 1.0f / (1 << (pyramidL.size() - 1));
    // 逐层搜索
    for (int level = pyramidL.size() - 2; level >= 0; --level) {
        // 缩放特征点坐标
        scaleFactor *= 2.0f; // 放大一倍, ptL 逐步缩小分辨率差距
        Point2f scaledPtL = ptL * scaleFactor; // *1/16 *1/8 *1/4
        Point2f scaledPtR = ptR * 2; // 放大一倍
        // 在当前层进行匹配, 找到更精确的坐标
    }
}
```

```

        ptR = findBestMatch(pyramidL[level], pyramidR[level], scaledPtL,
scaledPtR, windowSize, searchRadius / 2, ncc);
    }
    // 匹配点位置
    if (ncc > 0.96) {
        KeyPoint kpR(ptR, 1.0f);
        keypointsR.push_back(kpR);
        matches1to2.push_back(DMatch(i, static_cast<int>(keypointsR.size()) -
1, 0));
    }
}

```

左右图上绘制匹配的特征点，使用 **drawMatches** 在匹配点之间绘制连线，以显示匹配效果。

```

drawMatches(imgL_8U, keypointsL, imgR_8U, keypointsR, matches1to2, imgMatches,
Scalar(0, 255, 0), Scalar(0, 255, 0),
std::vector<char>(), DrawMatchesFlags::NOT_DRAW_SINGLE_POINTS);

```

此时要注意，使用 **drawMatches** 在匹配点之间绘制连线，要先将图片格式转化为 CV_8UC1。

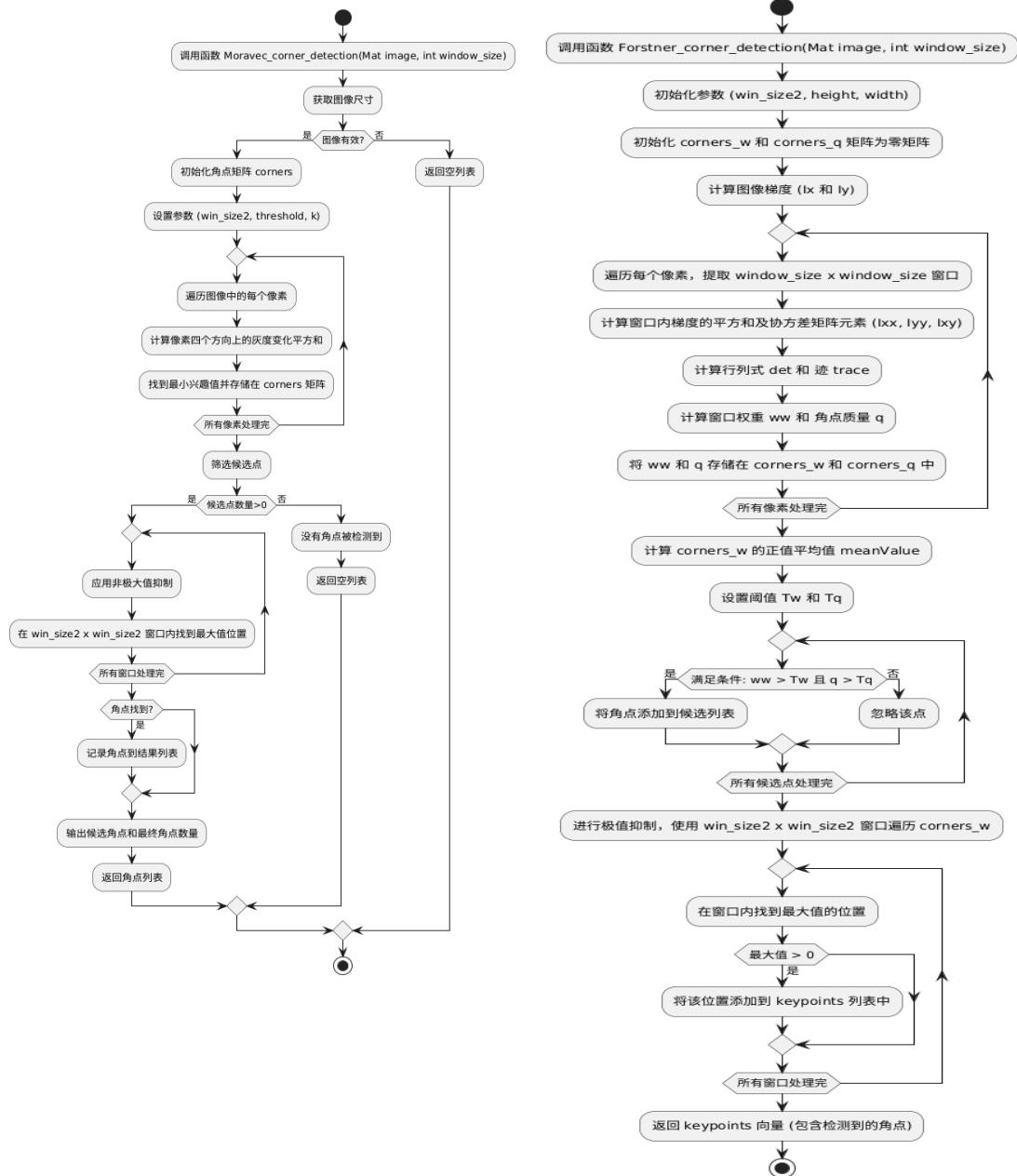
```

Mat imgL_8U, imgR_8U;
imgL.convertTo(imgL_8U, CV_8UC1, 255); // 乘以 255 是为了将浮点数缩放到 0-255 范围
imgR.convertTo(imgR_8U, CV_8UC1, 255);

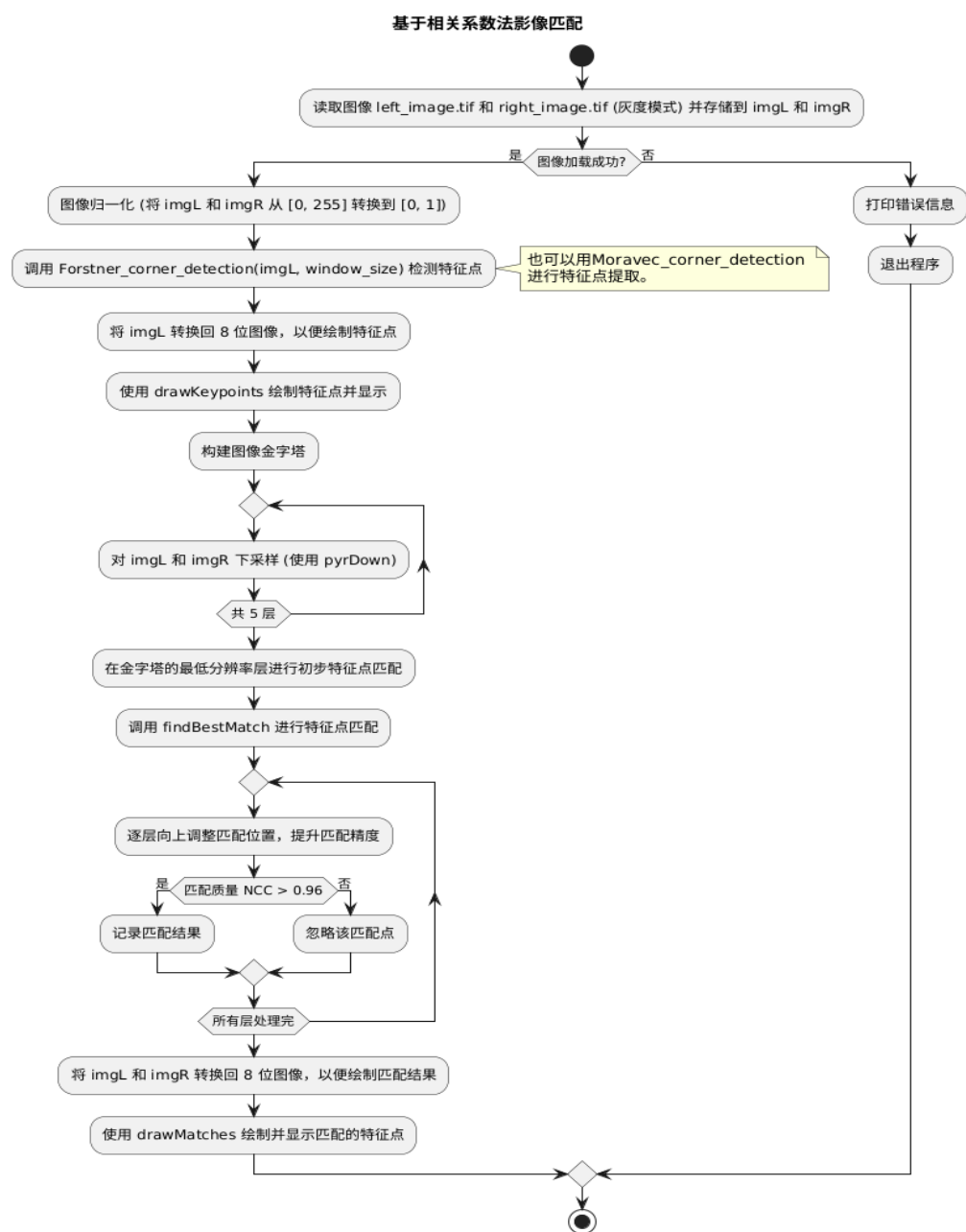
```

4.2 编程流程图

两个特征点点检测算法：



基于相关系数法影像匹配：



4.3 结果展示

Moravec 特征点检测：

兴趣值窗口大小：5×5；

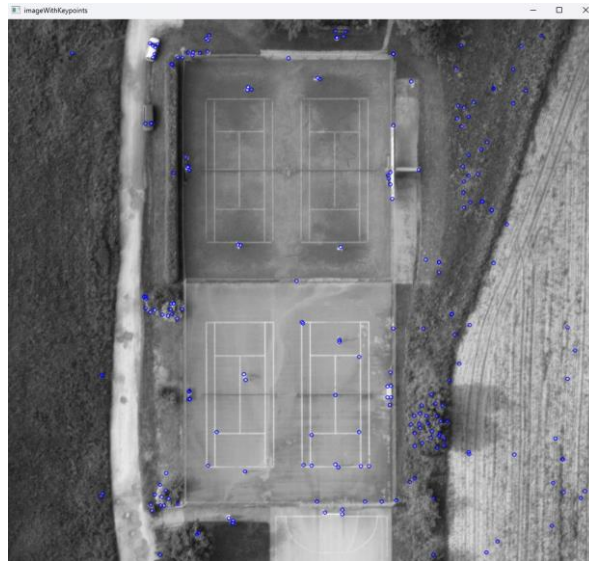
抑制窗口大小：15×15；

阈值：3.0/200.0（由于图像在 convertTo 中经过归一化处理，像素值变为 0~1）；

待选点个数：1507

极值点个数：212

Number of Chosen keypoints: 1507
Number of Result keypoints: 212



Forstner 特征点检测:

兴趣值窗口大小: 5×5 ;

抑制窗口大小: 7×7 ;

阈值: (由于图像在 `convertTo` 中经过归一化处理, 处理像素值变为 $0 \sim 1$);

$T_q = 0.9$;

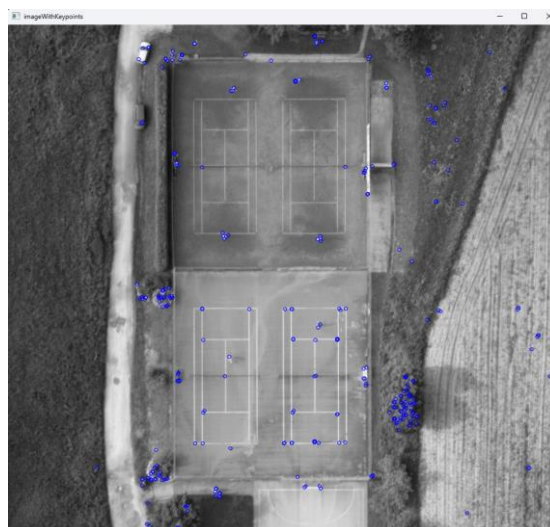
$T_w = 8.0 * \text{meanValue}(w_c)$;

w_c 的值: 0.00736

待选点个数: 1559

极值点个数: 300

Mean Value of corners_w: 0.00735776
Number of Chosen keypoints: 1559
Number of Result keypoints: 300



基于相关系数法特征点匹配（通过 Forstner 提取特征点）：

金字塔层数：5

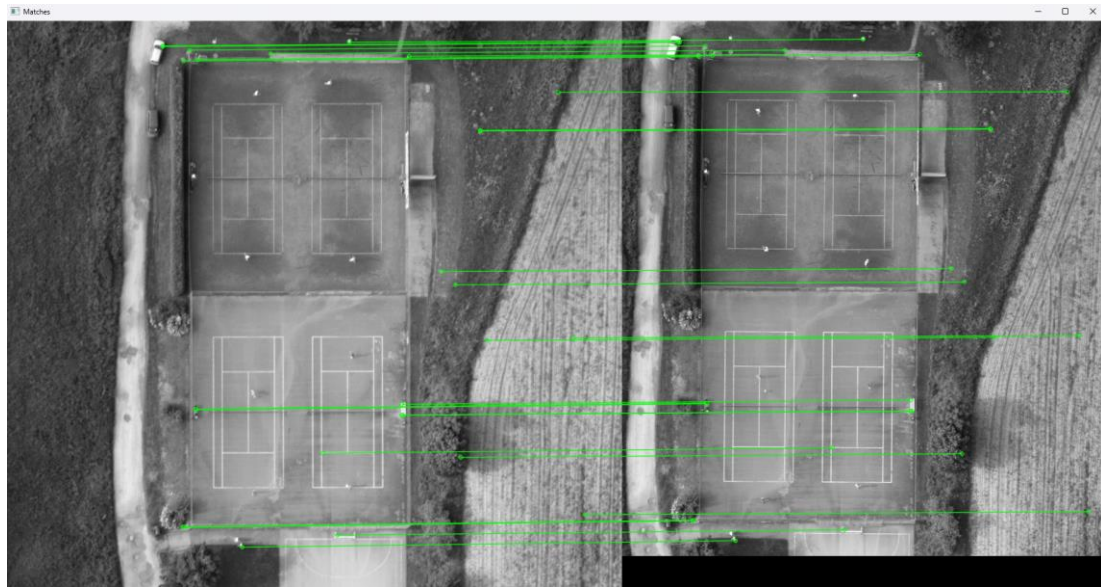
匹配窗口大小： 7×7 ；

搜索范围大小：20；

相关系数阈值： $ncc > 0.96$ ；

左图输入特征点数：300

匹配点个数：27



五 试验结果分析

5.1 特征点提取算法

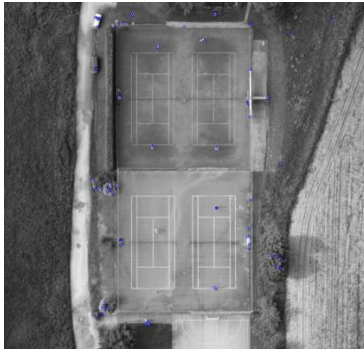
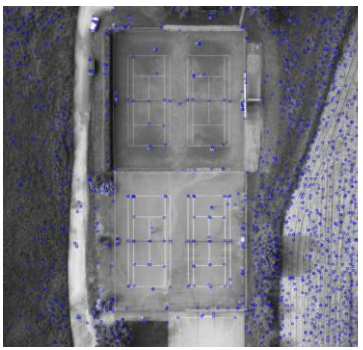
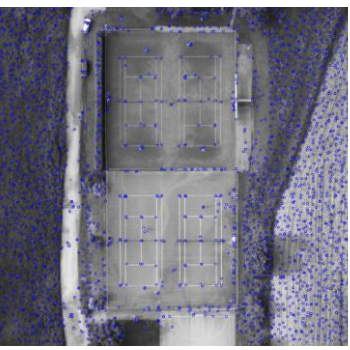
以 Moravec 为例，Forstner 算法类似。

通过修改 Moravec 算法的输入参数、相关系数匹配影像匹配算法的窗口大小与阈值，可得到该对影像特征点提取、影像匹配的不同试验结果，现对这些结果进行分析如下，主要有特征点提取算法阈值设定分析；随机分布与均匀分布提取特征点的实施分析；相关系数法影像匹配算法分析；窗口大小对匹配的影响。

5.1.1 阈值设定分析

以左影像为例，固定兴趣点窗口为 5×5 、抑制窗口为 15×15 ，改变阈值大小，

统计特征点个数，如下表所示：

阈值：3.0/100.0	阈值：3.0/500.0	阈值：3.0/1000.0
		
特征点个数：69	特征点个数：1377	特征点个数：2599

阈值如果太小，则出现大量强度不够的特征点，影像后续匹配精度。例如设置兴趣点窗口为 5*5、抑制窗口为 15*15、阈值为 3.0/1000.0、所检测到的特征点数量为 2599，数量过多。

阈值如果太大，则出现强度中等特征点被忽略，例如阈值为 3.0/100.0 时所检测到的特征点数量为 69，影像后续匹配输入点对不够也会影像精度。最终确定最佳阈值为 3.0/200.0。

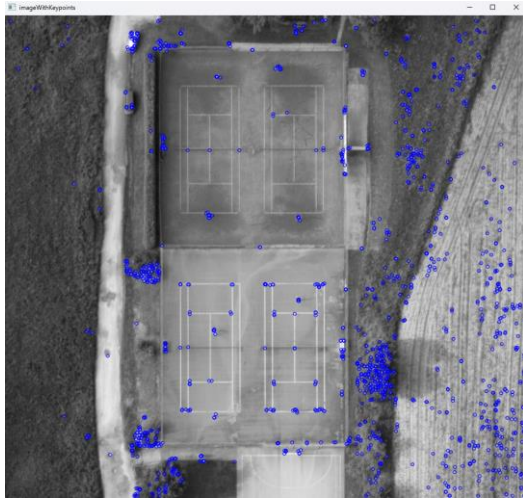
5.1.2 随机分布与均匀分布提取特征点的实施分析

以左影像为例，固定阈值大小为 3.0/200.0，在兴趣点窗口为 3*3、5*5、7*7、9*9、11*11、13*13，改变抑制窗口，统计特征点个数。

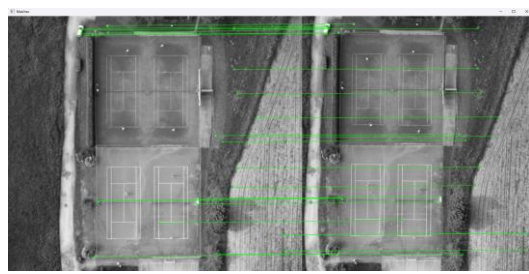
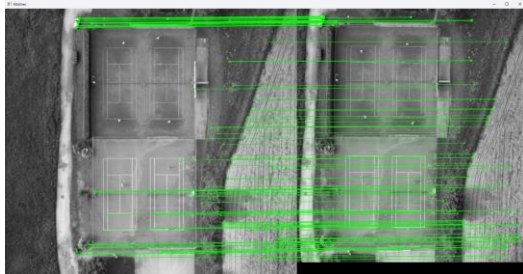
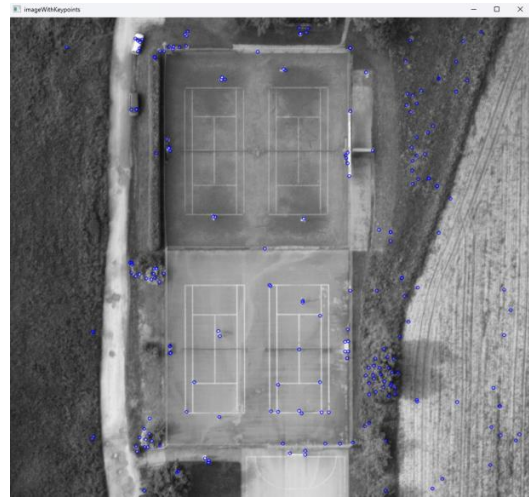
随机分布：个人认为可以通过适当调小抑制窗口进行实现，但是随机分布提取到特征点密集，不利于后续影像匹配，如下图所示（兴趣点窗口为 5*5、抑制窗口为 9*9），精度很低，检测特征点个数为 1350，匹配成功率在 20%。

均匀分布：个人认为可以通过适当增大抑制窗口进行实现，均匀分布提取到特征点则十分利用后续影像匹配，如下图所示（兴趣点窗口为 5*5、抑制窗口为 15*15），均分分布在图像各处，检测特征点个数为 212，匹配效果也很好。

随机分布



均匀分布



5.2 相关系数法影像匹配算法

5.2.1 算法分析

主要从相关系数的时间复杂度进行分析，改变搜索窗口与匹配窗口大小，统计这一段程序执行的时间。可以发现，增大搜索窗口与匹配窗口都会增加程序的执行时间，增大搜索窗口多耗费的时间多于匹配窗口耗费的时间。

5.2.2 窗口大小对匹配的影响

主要从匹配的成功率进行分析，给定阈值 0.96，改变搜索窗口与匹配窗口大小，统计这一段程序执行的时间，统计匹配成功率。

匹配窗口: 3×3

搜索范围: 15

匹配个数: 80

准确率: 20%

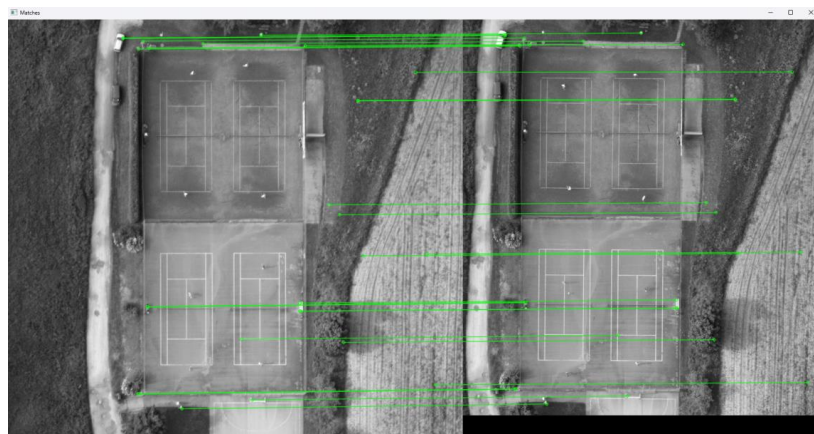


匹配窗口: 7×7

搜索范围: 20

匹配结果: 27

准确率: 100%

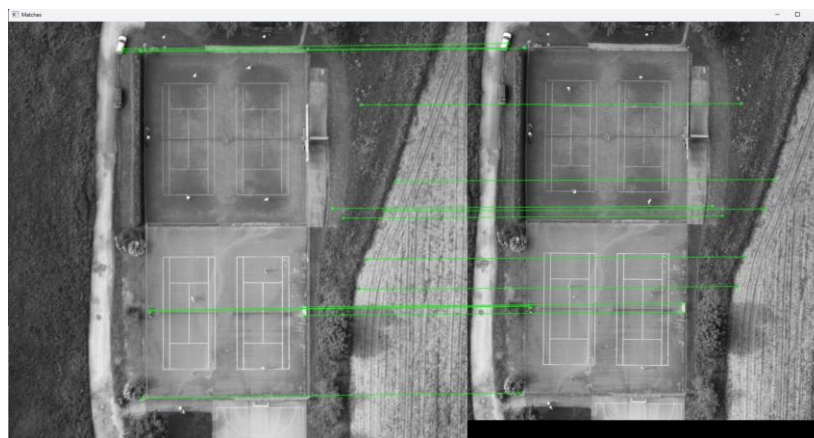


匹配窗口: 9×9

搜索范围: 25

匹配结果: 15

准确率: 100%



可以发现, 增大搜索窗口或减小匹配窗口都会增加匹配的成功率, 因此在用于匹配的过程, 考虑时间复杂度前提下, 最小化匹配窗口 (7×7) 并增大搜索窗口范围为 15, 可以有效提升相关系数法匹配的成功率。

六 心得体会

此次实习不仅深化了我对特征点提取及相关系数法匹配原理的认知，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，加深了我对特征点提取及相关系数法匹配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，通过本次特征点提取及相关系数法匹配的课间实习，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。