摄影测量学

——点特征提取及相关系数法匹配

学 院：遥感信息工程学院

班 级：2006 班（20F10）

姓 名：马文卓

学 号：2020302131249

目录

一、ReadMe

1.实习任务

本次实习的任务主要包含两大模块：点特征提取、相关系数法匹配。
• 点特征提取：利用 Moravec 或 Forstner 算法提取影像的特征点，并在影像上显示出来。同时研究兴趣阈值、抑制窗口对结果的影响，以及随机提取和均匀提取的实施分析。
• 相关系数法匹配：利用相关系数匹配算法，对两张影像进行匹配，得到同名点，并进行可视化。同时研究匹配窗口、相关系数阈值等对结果的影响。

2.实验数据

本次实习当中所用到的数据均存放于 data 文件夹中。包含三张图片：
• panLeft.bmp：灰度影像，942x1023
• panRight.bmp：灰度影像，887x805
• chess.bmp：彩色影像，467x638，主要用于检测点特征提取算法的效果

3.编程环境

本次实习当中编程环境如下：
• 编程语言：Python
• 环境：Python 3.9.7
• 第三方库：opencv、numpy、math（其余 Moravec、Forstner、匹配算法均为自己编写）

4.文件组织结构

• 点特征提取及相关系数匹配：项目总文件夹
    • data：实验数据文件夹
    • PointFeatureExtraction：点特征提取文件夹
        • Moravec(py)：Moravec 算法
        • Forstner(py)：Forstner 算法
        • main(py)：点特征提取主程序
    • CorrelationCoefficientMatching：相关系数匹配文件夹
        • Moravec(py)：匹配算法中所用到的 Moravec 特征提取算法
        • main(py)：相关系数法匹配的主程序
    • Result：实验结果文件夹
        • PointFeatureExtraction：点特征提取结果文件夹
            • Moravec：Moravec 算法结果文件夹

- chess：棋盘数据结果文件夹
- tenniscourt：网球场数据结果文件夹

- Forstner：Forstner 算法结果文件夹
  - chess：棋盘数据结果文件夹
  - tenniscourt：网球场数据结果文件夹
- CorrelationCoefficientMatching：相关系数匹配结果文件夹

- 实验报告(docx)：实验报告
- ReadMe(txt)：项目概述
- illustration：流程图文件夹

二、原理及算法

1.点特征提取

点特征提取是指运用某种算法从影像中提取我们感兴趣的或者有利用价值的点的过程。其中特征点主要指明显点、具有某种特征的点（如灰度特征），提取特征点所用的算子则称为兴趣算子或有利算子。主要的算子有 Moravec、Forstner、Harris 算子等，具体介绍见下文。

1.1Moravec 算法

（1）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）

（2）算法步骤：

1>计算各像元的兴趣值 IV：以像素（c,r）为中心的 wxw 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

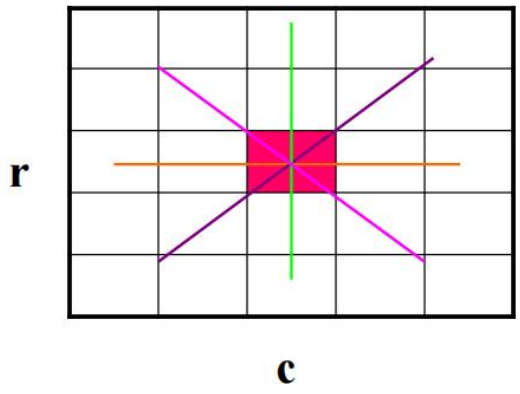
$$V1=\sum_{i=-k}^{k-1}(g_{c+i,r}-g_{c+i+1,r})^2$$

$$V2=\sum_{i=-k}^{k-1}(g_{c+i,r+i}-g_{c+i+1,r+i+1})^2$$

$$V3=\sum_{i=-k}^{k-1}(g_{c,r+i}-g_{c,r+i+1})^2$$

$$V4=\sum_{i=-k}^{k-1}(g_{c+i,r-i}-g_{c+i+1,r-i-1})^2$$

$$IV_{c,r}=\min\{V1,V2,V3,V4\}$$



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

（3）算法特点：

1>简单快速

2>对强边缘比较敏感，这是由于响应值是自相关的最小值而不是差值

3>不是旋转不变的，当图像发生旋转时，检测出的特征点具有较低的重复率

4>对噪声也比较敏感，可以考虑用比较平滑的滤波来消除噪声

1.2Forstner 算法

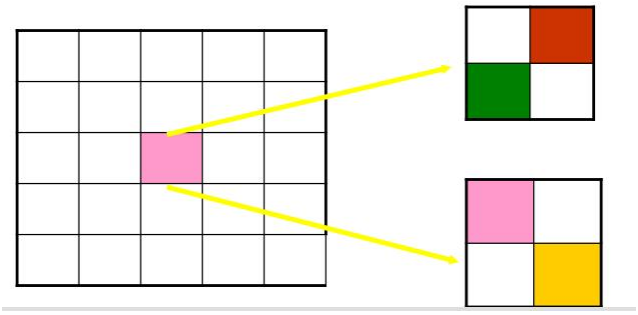
（1）概述：通过计算各像素的 Robert’s 梯度和像素（c，r）为中心的窗口内的灰度协方差矩阵，在影像中寻找具有尽可能小而接近圆的误差椭圆的点作为特征点

（2）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 lxl 窗口中的灰度协方差矩阵

$$\sum g_u^2 = \sum_{i=c-k}^{c+k-1} \sum_{j=r-k}^{r+k-1} (g_{i+1,j+1} - g_{i,j})^2$$

$$\sum g_v^2 = \sum_{i=c-k}^{c+k-1} \sum_{j=r-k}^{r+k-1} (g_{i,j+1} - g_{i+1,j})^2$$

$$\sum g_u g_v = \sum_{i=c-k}^{c+k-1} \sum_{j=r-k}^{r+k-1} (g_{i+1,j+1} - g_{i,j})(g_{i,j+1} - g_{i+1,j})$$

$$N = \begin{bmatrix} \sum g_u^2 & \sum g_u g_v \\ \sum g_u g_v & \sum g_v^2 \end{bmatrix}$$

3>计算兴趣值 q、w

$$\omega = \frac{1}{\mathrm{trQ}} = \frac{DetN}{trN}$$

$$q = \frac{4DetN}{(trN)^2}$$

4>确定候选点：当 q>Tq 同时 w>Tw 时，该像元为候选点

$$T_q = 0.5 \sim 0.75$$

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

其中$\overline{w}$为权平均值，w_c为权中值

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

（3）算法特点：

1>精度高

2>计算量大

2. 相关系数法匹配

（1）概述：基于相关系数来判断搜寻两张影像上的同名点

（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：使用 ps 等工具得到左右影像的大致视差，然后根据搜索窗口的大小确定搜索范围(搜索窗口不能太小,否则可能找不到匹配同名点)

4>逐窗口计算相关系数并保存：计算窗口内左影像个右影像灰度的相关系数，并且保存起来。公式如下：

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

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

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

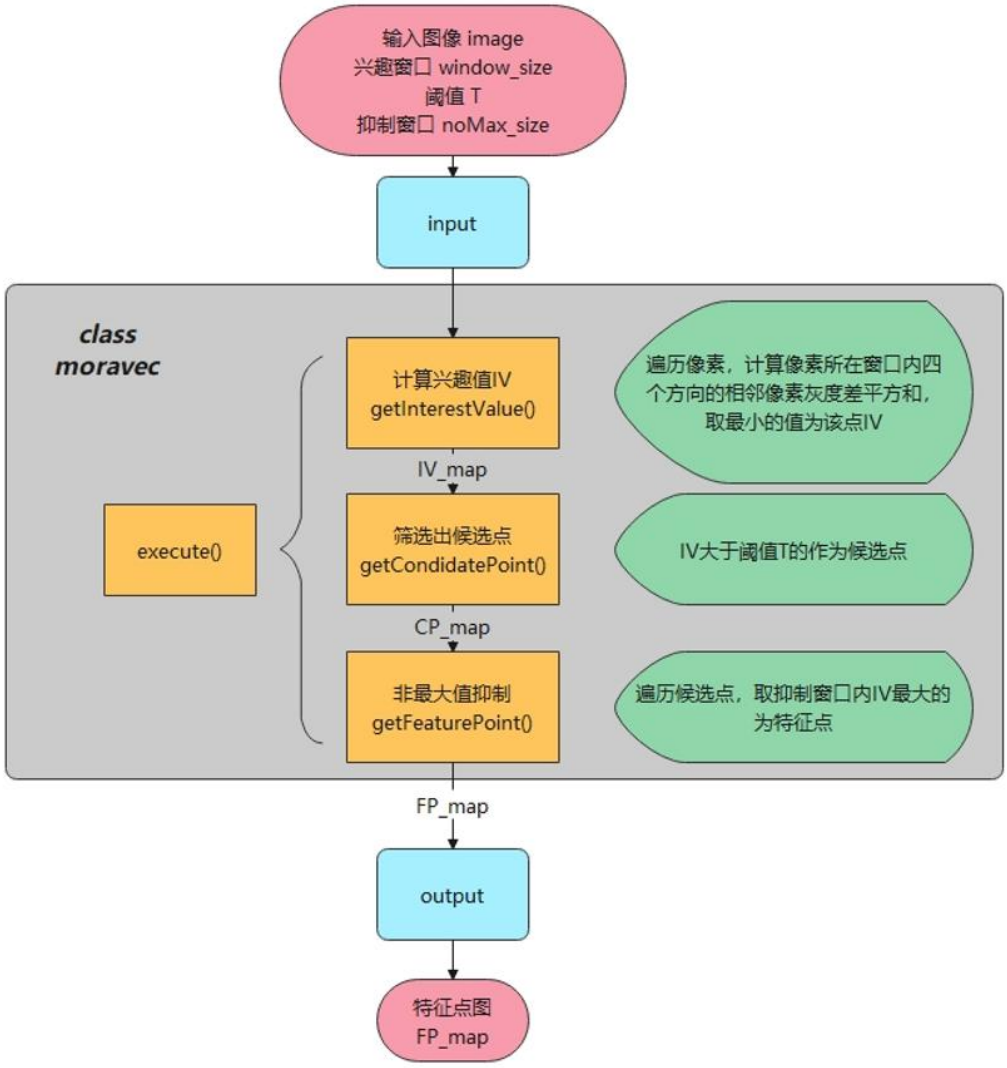
5>比较一个搜索窗口内的相关系数，如果最大值大于相关系数阈值，则该最大值对应的像点则为对应目标点在右影像上的同名点

三、编程思路及流程图

1.点特征提取

1.1Moravec 算法

（1）流程图



（2）编程思路

总体编程架构为：编写 moravec 类，其包含了 getInterestValue、getCondidatePoint、getFeaturePoint、execute 等方法来实现特征点的提取。

1>init

初始化函数。

主要初始化参数为：输入图像（彩色）、兴趣窗口大小、兴趣阈值、抑制窗口大小。详细信息见下图：

```
class moravec:
    '''
        Moravec点特征提取算法
    '''
    def __init__(self,img,window_size=5,T=-1,noMax_size=5):
        '''
            Parameters:
                img:输入彩色图像\n
                window_size:计算兴趣值窗口大小(default:5)\n
                T:确定候选点的兴趣值阈值(default:-1,即取IV非零均值为阈值)\n
                noMax_size:非极大值抑制窗口大小(default:19)\n
        '''
        self.img=cv2.cvtColor(img,cv2.COLOR_BGR2GRAY).astype(int)
        self.h=self.img.shape[0]
        self.w=self.img.shape[1]
        self.win_size=window_size
        self.T=T
        self.noMax_size=noMax_size
        self.IV_map=np.zeros([self.h, self.w], np.int32)#兴趣值图
        self.CP_map=np.zeros([self.h, self.w], np.int32)#候选点图
        self.FP_map=np.zeros([self.h, self.w], np.int32)#特征点图
```

2>getInterestValue

计算兴趣值函数。

遍历像素（边界像素可忽略），计算垂直、水平、正反对角线对应相邻像素灰度差平方和，取最小值为该点 IV，并记录在 IV_map 中，最后输出 IV_map 图像。

```
def getInterestValue(self):
    '''
        计算每个像素点的兴趣值
    '''
    k=int(self.win_size/2)
    for r in range(k,self.h-k):#遍历像素
        for c in range(k,self.w-k):#去除外围像素
            V1=V2=V3=V4=0
            for i in range(-k,k):#遍历窗口
                #计算四个方向相邻像素灰度差的平方和
                V1=V1+(self.img[r+i,c]-self.img[r+i+1,c])**2#纵向
                V2=V2+(self.img[r+i,c+i]-self.img[r+i+1,c+i+1])**2#正对角线
                V3=V3+(self.img[r,c+i]-self.img[r,c+i+1])**2#横向
                V4=V4+(self.img[r+i,c-i]-self.img[r+i+1,c-i-1])**2#反对角线
            IV=min(V1,V2,V3,V4)#取最小值为该点的兴趣值
            self.IV_map[r,c]=IV#记录各点的兴趣值
    cv2.imwrite('Result\\PointFeatureExtraction\\Moravec\\tenniscourt\\IV_map.jpg',self.IV_map)
```

3>getCondidatePoint

筛选出候选点。

根据之前提供的阈值，将 IV 大于阈值 T 的点作为特征点候选点，记录在 CP_map 中，并输出 CP_map 图像。

如果 T 为默认值-1，则阈值为图像 IV 非零均值。

```
def getCondidatePoint(self):
    '''
        根据阈值筛选得到候选点,如果T=-1则默认将非零均值作为阈值
    '''
    T=self.T
    if self.T==-1:
        T=np.sum(self.IV_map)/np.count_nonzero(self.IV_map)
    self.CP_map=np.where(self.IV_map<T,0,self.IV_map)#兴趣值大于阈值的点作为候选点
    cv2.imwrite('Result\\PointFeatureExtraction\\Moravec\\tenniscourt\\CP_map.jpg',self.CP_map)
```

4>getFeaturePoint

非极大值抑制，得到特征点。

以抑制窗口遍历图像，每个抑制窗口内选取 IV 最大的候选点作为特征点，记录在 FP_map 中，并输出 FP_map 图像。值得注意的是，在此步骤中一定要防止抑制窗口越界。

```
def getFeaturePoint(self):
    '''
        在候选点中选取极值点作为特征点
    '''
    k=int(self.noMax_size/2)
    for r in range(k,self.h-k,self.noMax_size):#按抑制窗口覆盖图片进行筛选
        for c in range(k,self.w-k,self.noMax_size):
            a1=r-k if r-k>0 else 0
            a2=r+k if r+k<self.h else self.h
            b1=c-k if c-k>0 else 0
            b2=c+k if c+k<self.w else self.w
            area=self.CP_map[a1:a2,b1:b2]#抑制区域
            fp_positions=np.where(area==np.max(area))#每个区域内最大值为特征点
            for i in range(len(fp_positions[0])):
                if self.CP_map[fp_positions[0][i]+a1,fp_positions[1][i]+b1]!=0:
                    self.FP_map[fp_positions[0][i]+a1,fp_positions[1][i]+b1]=255#在特征图中标注特征点
    cv2.imwrite('Result\\PointFeatureExtraction\\Moravec\\tenniscourt\\FP_map.jpg',self.FP_map)
```

5>Execute

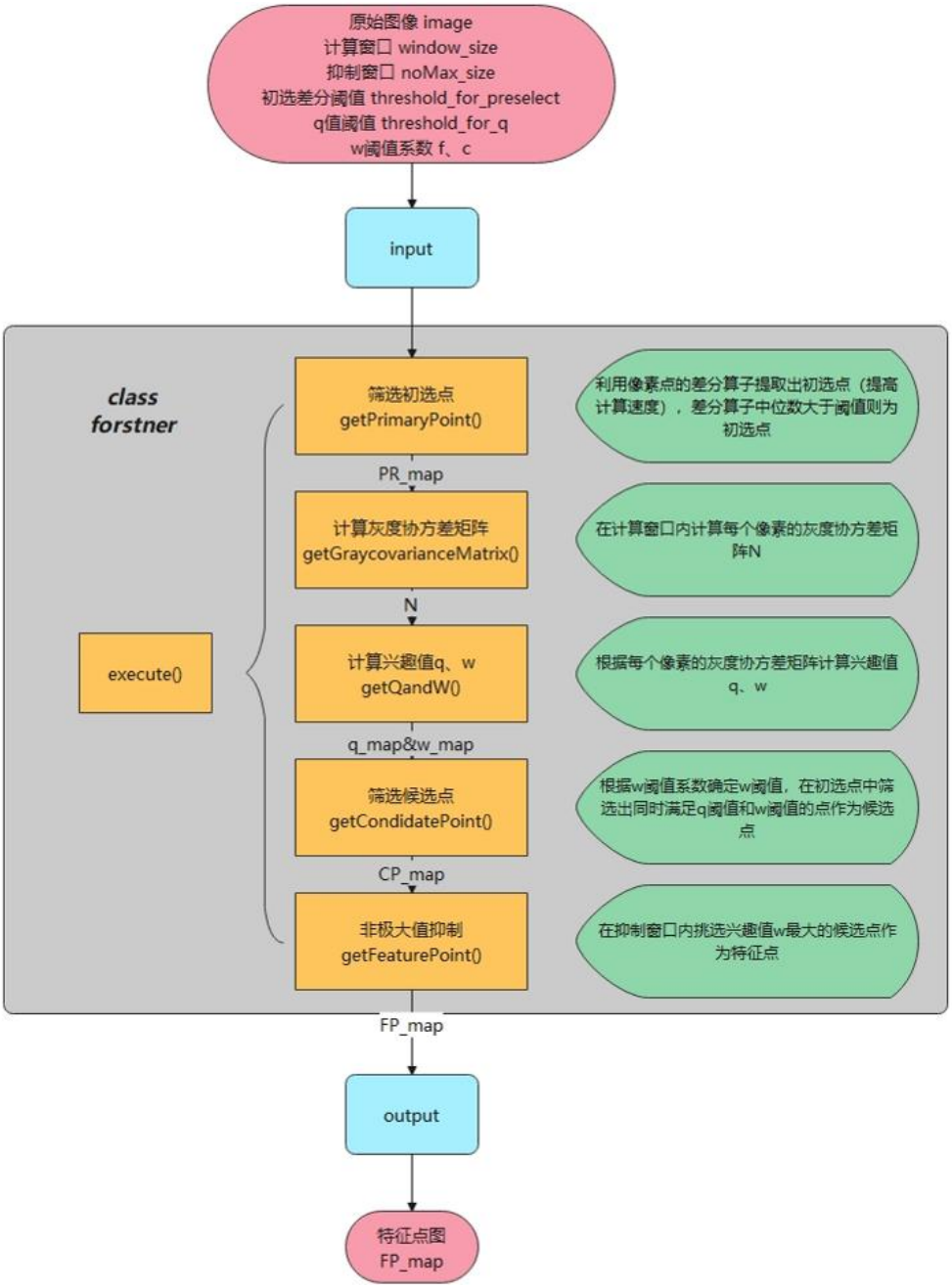
运行函数，执行 Moravec 算法。

使用上述方法，依次进行兴趣值计算、阈值筛选、非极大值抑制，返回特征点图 FP_map。

```
def execute(self):
    '''
        执行Moravec算法,返回标注图像
    '''
    self.getInterestValue()#计算兴趣值
    self.getCondidatePoint()#得到候选点
    self.getFeaturePoint()#得到特征点
    return self.FP_map
```

1.2Forstner 算法

（1）流程图



（2）编程思路

总体编程架构为：编写 forstner 类，其包含了 getPrimaryPoint、getGraycovarianceMatrix、getQandW、getCondidatePoint、getFeaturePoint、execute 等方法来实现特征点的提取。

1>init

初始化。

主要初始化参数为：输入图像（彩色）、计算窗口大小、抑制窗口大小、初选差分阈值、q 值阈值、w 阈值系数等。

具体见下图：

```
def __init__(self,img,window_size=5,noMax_size=7,threshold_for_preselect=20,threshold_for_q=0.5,f=0.5,c=5):
    '''
        Parameters:
            img:输入彩色图像\n
            window_size:计算协方差矩阵窗口大小(default:5)\n
            noMax_size:非极大值抑制窗口大小(default:19)\n
            threshold_for_preselect:进行最初筛选的差分阈值\n
            threshold_for_q:q值阈值(default:0.5)\n
            f、c:确定w权值阈值的系数(default:f=0.5,c=5)\n
    '''
    self.img=cv2.cvtColor(img,cv2.COLOR_BGR2GRAY).astype(int)
    self.h=self.img.shape[0]
    self.w=self.img.shape[1]
    self.win_size=window_size
    self.noMax_size=noMax_size
    self.Tpre=threshold_for_preselect
    self.Tq=threshold_for_q
    self.f=f
    self.c=c
    self.PR_map=np.zeros([self.h, self.w], np.int32)#初选点标志图
    self.q_map=np.zeros([self.h, self.w], np.float)#q值图
    self.w_map=np.zeros([self.h, self.w], np.float)#w值图
    self.CP_map=np.zeros([self.h, self.w], np.int32)#候选点标志图
    self.FP_map=np.zeros([self.h, self.w], np.int32)#特征点标志图
```

2>getPrimaryPoint

筛选初选点函数。

为了减少不必要的计算，加快算法速度，通过计算每个像素的差分算子，如果差分算子的中位数大于初选阈值，则该点标记为初选点，记录在 PR_map 中。（这样一些没有价值的点就被排除，不需要经历后续负责的运算，节约时间成本）。

```
def getPrimaryPoint(self):
    '''
        利用像素点的差分算子提取出初选点（提高计算速度）
        差分算子中位数大于阈值则为初选点
    '''
    k=int(self.win_size/2)
    for r in range(k,self.h-k):#遍历像素
        for c in range(k,self.w-k):#去除外围像素
            #差分算子
            dg1=dg2=dg3=dg4=0
            dg1=abs(self.img[r,c]-self.img[r+1,c])
            dg2=abs(self.img[r,c]-self.img[r,c+1])
            dg3=abs(self.img[r,c]-self.img[r-1,c])
            dg4=abs(self.img[r,c]-self.img[r,c-1])
            a=np.median([dg1,dg2,dg3,dg4])
            if np.median([dg1,dg2,dg3,dg4])>=self.Tpre:#标记初选点
                self.PR_map[r,c]=255
```

3>getGraycovarianceMatrix

计算灰度协方差矩阵。

根据输入的范围以及上述公式，计算出该点的灰度协方差矩阵 N。

```
def getGraycovarianceMatrix(self,r,c,k):
    '''
        计算灰度协方差矩阵\n
        Parameters:
            r,c:像素点行列号\n
            k:窗口半径\n
    '''
    gu2=guv=gv2=0
    N=np.zeros((2,2),dtype=np.float)
    for i in range(-k,k):#遍历窗口
        for j in range(-k,k):
            #计算灰度协方差矩阵
            gu2+=(self.img[r+i+1,c+j+1]-self.img[r+i,c+j])**2
            gv2+=(self.img[r+i+1,c+j]-self.img[r+i,c+j+1])**2
            guv+=(self.img[r+i+1,c+j+1]-self.img[r+i,c+j])*(self.img[r+i+1,c+j]-self.img[r+i,c+j+1])
    N[0,0]=gu2
    N[0,1]=guv
    N[1,0]=guv
    N[1,1]=gv2
    return N
```

4>getQandW

计算兴趣值 q、w。

根据灰度协方差矩阵和上述公式计算该点兴趣值 q、w，并存储在 q_map 和 w_map 中。

```
def getQandW(self):
    '''
        根据每个像素窗口内的灰度协方差矩阵计算兴趣值q、w
    '''
    k=int(self.win_size/2)
    for r in range(k,self.h-k):#遍历像素
        for c in range(k,self.w-k):#去除外围像素
            N=self.getGraycovarianceMatrix(r,c,k)#获取灰度协方差矩阵
            #计算q和w
            q=w=0
            if np.trace(N)!=0:
                q = 4*np.linalg.det(N)/((np.trace(N))**2)
                w = np.linalg.det(N)/np.trace(N)
            self.q_map[r,c]=q
            self.w_map[r,c]=w
```

5>getCondidatePoint

筛选出候选点。

根据 w 阈值系数确定 w 阈值，将初选点中同时满足 q 阈值和 w 阈值的点作为特征点。

```
def getCondidatePoint(self):
    '''
        根据q、w和阈值确定待选点
    '''
    k=int(self.win_size/2)
    for r in range(k,self.h-k):#遍历像素
        for c in range(k,self.w-k):#去除外围像素
            if self.PR_map[r,c]!=0:#在初选点中寻找
                if self.q_map[r,c]>=self.Tq and \
                self.w_map[r,c]>=self.f*np.mean(self.w_map) and \
                self.w_map[r,c]>=self.c*np.median(self.w_map):
                    self.CP_map[r,c]=255#标志候选点
```

6>getFeaturePoint

非极大值抑制，得到特征点。

取每个抑制窗口中 w 值最大的候选点作为特征点。

```
def getFeaturePoint(self):
    '''
        将候选点经过非最大值抑制之后得到特征点
    '''
    k=int(self.noMax_size/2)
    for r in range(k,self.h-k,self.noMax_size):#按抑制窗口覆盖图片进行筛选
        for c in range(k,self.w-k,self.noMax_size):
            a1=r-k if r-k>0 else 0
            a2=r+k if r+k<self.h else self.h
            b1=c-k if c-k>0 else 0
            b2=c+k if c+k<self.w else self.w
            area=self.w_map[a1:a2,b1:b2]#抑制区域
            fp_positions=np.where(area==np.max(area))#每个区域内最大值为特征点
            for i in range(len(fp_positions[0])):
                if self.CP_map[fp_positions[0][i]+a1,fp_positions[1][i]+b1]!=0:
                    self.FP_map[fp_positions[0][i]+a1,fp_positions[1][i]+b1]=255#在特征图中标注特征点
```

7>Execute

运行函数，执行 forstner 算法。

使用上述方法，依次进行初选、兴趣值计算、阈值筛选、非极大值抑制，返回特征点图 FP_map。

```
def execute(self):
    '''
        执行Forstner算法,返回标注图像
    '''
    self.getPrimaryPoint()#筛选初选点
    self.getQandW()#计算兴趣值
    self.getCondidatePoint()#得到候选点
    self.getFeaturePoint()#得到特征点
    return self.FP_map
```

1.3 点特征提取主程序

本程序主要是进行点特征提取算法主框架编写，进行图片读取、算法选择、特征点标注、结果保存等工作。

1>图片读取

使用 opencv 库的 imread 函数进行图片读取并展示。

```
#图片读取
def readImage(imgPath):
    '''
        读取图片,并返回图像数组\n
        Parameters:
            imgPath:图片路径
    '''
    img=cv2.imread(imgPath)
    cv2.imshow('original image',img)
    cv2.waitKey(0)
    return img
```

2>特征点标注

使用 opencv 库的 circle 函数进行特征点的标注，值得注意的是这个函数中的坐标参数采用的是（列号，行号）。

```
def drawFeaturePoint(img,FP_map):
    '''
        根据特征点在原图上标注
        Parameters:
            img:输入图像
            FP_map:特征点图
    '''
    h=img.shape[0]
    w=img.shape[1]
    for r in range(h):
        for c in range(w):
            if FP_map[r,c]==255:#如果是特征点
                cv2.circle(img, (c,r), 1, [0, 0, 255], 2, cv2.LINE_AA)#标注
    return img
```

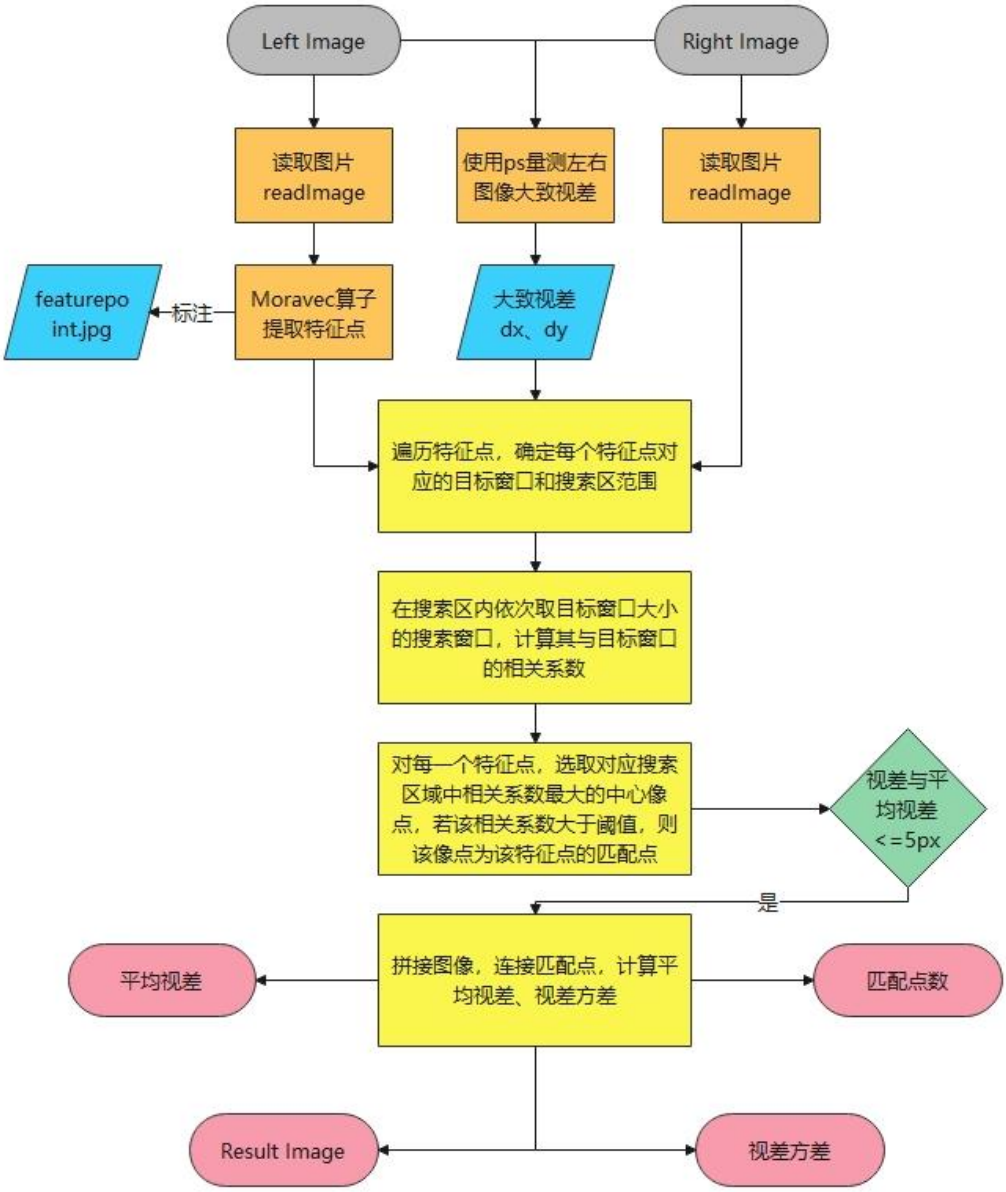
3>算法选择和结果保存

这里编写了一个简单的循环选择界面，可给用户提供简洁的使用需求。

```
if __name__=='__main__':
    while True:
        image=readImage('data\\panLeft.bmp')
        print('----------------------------------------点特征提取--------------------------------')
        print('0.退出程序')
        print('1.Moravec算法')
        print('2.Forstner算法')
        decision=input('请输入您想要使用的方法:(输入序号)')
        if int(decision)==0:
            print('程序退出成功')
            break
        if int(decision)==1:
            moravec_test=moravec(image,T=1200)#6000
            FP_map=moravec_test.execute()
            result=drawFeaturePoint(image,FP_map)
            cv2.imwrite('Result\\PointFeatureExtraction\\Moravec\\tenniscourt\\result.jpg',result)
        if int(decision)==2:
            forstner_test=forster(image)
            FP_map=forstner_test.execute()
            result=drawFeaturePoint(image,FP_map)
            cv2.imwrite('Result\\PointFeatureExtraction\\Forstner\\tenniscourt\\result.jpg',result)
        else:
            print('请输入正确的序号')
```

2. 相关系数法匹配

（1）流程图



（2）编程思路

相关系数匹配的实现主要是两个模块完成，一个是引用了之前点特征提取中的 Moravec 类完成算法中的特征提取部分，其与部分（图片读取、匹配、结果保存等）均在 main.py 中。

1>量测大致视差

为了防止遗太多误匹配（如果不量测大致视差，正确的匹配点有时可能会在搜索区域外，从而导致很多误匹配），我使用 ps 对左右图像的进行大致视差的量测。具体步骤如下：

①将两张图片在 ps 中打开，在左图中取一明显点，记录坐标(x1,y1)

②在右图中取相同明显点，记录坐标(x2,y2)

③计算视差：水平视差=x2-x1，上下视差=y2-y1

④重复①-③步骤 4-5 次，取视差平均值为大致视差

经过量测，大致视差为：水平视差=176，垂直视差=8

2>读取图片

和点特征提取中的 readImage 一致。

```
#图片读取
def readImage(imgPath,showName):
    '''
        读取图片,并返回图像数组\n
        Parameters:
            imgPath:图片路径
            showName:展示名称
        Return:
            img:读取图像
            h:图像高度
            w:图像宽度
    '''
    img=cv2.imread(imgPath,0)
    h=img.shape[0]
    w=img.shape[1]
    cv2.imshow(showName,img)
    cv2.waitKey(0)
    return img,h,w
```

```
print('------------------Read Image------------------')
l_img,l_h,l_w=readImage('data\\panLeft.bmp','original left image')
r_img,r_h,r_w=readImage('data\\panRight.bmp','original right img')
cv2.waitKey(0)
print('left image size:({},{})'.format(l_h,l_w))
print('right image size:({},{})'.format(r_h,r_w))
```

3>提取目标点并绘制目标点

这里引用的是点特征提取中的 Moravec 算法，特征点绘制也不过多赘述，和上述模块一致，可见【第三节 1.1】。

```
print('------------------Feature Point Extraction------------------')
moravec_test=moravec(l_img,T=0,noMax_size=100)
l_FP=moravec_test.execute()#moravec算子获取特征点
l_colorimg=cv2.imread('data\\panLeft.bmp')
l_colorimg=drawFeaturePoint(l_colorimg,l_FP)#标注特征点
cv2.imwrite('Result\\CorrectlationCoefficientMatching\\featurepoint.jpg',l_colorimg)
print('特征点个数:{}'.format(np.count_nonzero(l_FP)))
print('Feature Point Extraction Finish')
```

```
def drawFeaturePoint(img,FP_map):
        '''
            根据特征点在原图上标注\n
            Parameters:
                img:输入图像
                FP_map:特征点图
            Return:
                img:标注后图像
        '''
        h=img.shape[0]
        w=img.shape[1]
        for r in range(h):
            for c in range(w):
                if FP_map[r,c]==255:#如果是特征点
                    cv2.circle(img, (c,r), 1, [0, 0, 255], 2, cv2.LINE_AA)#标注
        return img
```

4>计算相关系数

相关系数的计算封装在 getCC 函数中，其输入输出均可见下图。基本思想就是根据输入的两个窗口进行遍历，然后根据公式求相关系数。

```
def getCC(leftImageWin,rightImageWin):
    '''
        计算得到两个窗口间图像的相关系数\n
        Parameters:
            leftImageWin:左图计算窗口
            rightImageWin:右图计算窗口
        Return:
            CC:两个计算窗口的相关系数
    '''
    if leftImageWin.shape!=rightImageWin.shape:#如果左右窗口大小不相等
        return 0.0#相关系数为0
    #窗口大小
    h,c=leftImageWin.shape[0],leftImageWin.shape[1]
    #计算窗口灰度平均值
    lg_avg=np.average(leftImageWin)
    rg_avg=np.average(rightImageWin)
    #计算相关系数累加量
    sum1=sum2=sum3=0
    for i in range(h):
        for j in range(c):#遍历窗口
            sum1+=(leftImageWin[i,j]-lg_avg)*(rightImageWin[i,j]-rg_avg)
            sum2+=(leftImageWin[i,j]-lg_avg)**2
            sum3+=(rightImageWin[i,j]-rg_avg)**2
    #计算相关系数
    CC=sum1/(sqrt(sum2*sum3))
    return CC
```

5>确定匹配点

确定匹配点的功能封装于 CCMatching 函数，函数的输入和输出可见下图。

```
def CCMatching(leftImage,rightImage,FP_map,winSize=7,winSearch=21,threshold=0.75):
    '''
        相关系数匹配\n
        Parameters:
            leftImage:左图
            rightImage:右图
            FP_map:左图像特征点图\n
            winSize:计算相关系数窗口大小(default:7)\n
            winSearch:搜索区大小(default:21)
            threshold:判断是否为同名点的阈值(default:0.75)
        Return:
            FP:特征点位置数组([x1,x2,x3...],[y1,y2,y3...])
            MP:与特征点对应匹配点位置数组[[x1,y1],[x2,y2],[x3,y3]...]
    '''
    leftImage=leftImage.astype(int)
    rightImage=rightImage.astype(int)
    k=int(winSize/2)
    s=int(winSearch/2)
    #左右图像坐标偏差
    dx=176
    dy=8
    #图像高宽
    l_h,l_w=leftImage.shape
    r_h,r_w=rightImage.shape
    #得到特征点的坐标
    FP=np.nonzero(FP_map)
    #匹配点图
    MP=np.tile(np.array([-1,-1]),(len(FP[0]),1))
```

首先遍历每个特征点，根据大致视差和搜索区域大小确定其对应的搜索区域；然后对搜索区域内的每个像素的计算窗口都将其和目标窗口进行相关系数计算；一个搜索区域内最大相关系数对应的像点，若其相关系数大于给定阈值，则该点确定为该特征点的匹配点。记录下特征点信息 FP 和对应匹配点信息 MP，其组织结构可见函数信息。

```
for p in range(len(FP[0])):
    #左图特征点坐标
    rl=FP[0][p]
    cl=FP[1][p]
    #右图特征点坐标
    rr=rl-dy
    cr=cl-dx
    #确定搜索区(防止越界)
    a1=rr-s if rr-s>=k else k
    a2=rr+s if rr+s<=r_h-k else r_h-k
    b1=cr-s if cr-s>=k else k
    b2=cr+s if cr+s<=r_w-k else r_w-k
    leftImageWin=leftImage[rl-k:rl+k,cl-k:cl+k]#左图计算窗口
    CC_map=np.zeros((winSearch,winSearch),np.float32)#搜索区内的相关系数图
    #遍历搜索区
    for i in range(a1,a2):
        for j in range(b1,b2):
            rightImageWin=rightImage[i-k:i+k,j-k:j+k]#右图计算窗口
            CC=getCC(leftImageWin,rightImageWin)#计算相关系数
            CC_map[i-a1,j-b1]=CC
    if np.max(CC_map)>=threshold:#如果搜索区最大相关系数大于阈值
        #得到搜索区相关系数最大的位置
        pos=np.where(CC_map==np.max(CC_map))
        #记录特征点的相应匹配点
        MP[p,0]=pos[0][0]+a1
        MP[p,1]=pos[1][0]+b1
return FP,MP
```

6>筛选匹配点、结果图绘制、结果输出

这部分内容封装于 drawMatchLine 函数中，函数详细信息可见下图。

```
def drawMatchLine(leftImage,rightImage,FP,MP,l_h,l_w,r_h,r_w):
    '''
        将匹配点之间连线\n
        Parameters:
            leftImage:左图
            rightImage:右图
            FP:特征点位置信息
            MP:匹配点匹配关系及位置
            l_h、l_w:左图大小
            r_h、r_w:右图大小
        Return:
            concatImage:标注好匹配关系的拼接图像
            dx:平均水平视差
            dy:平均上下视差
            ddx:水平视差方差
            ddy:上下视差方差
            num1:未经筛选的成功匹配点数
            num2:经过筛选后成功匹配点数
    '''
```

首先对之前匹配成功的点进行平均视差的计算，将平均视差作为视差真值（由于偶然误差的随机性，所以平均误差可近似看做真值）。

```
#视差
dx=0#左右视差
dy=0#上下视差
num1=0#匹配个数
#匹配点连线
for p in range(len(FP[0])):#遍历每个特征点
    #特征点位置
    fp_r=FP[0][p]
    fp_c=FP[1][p]
    #相应匹配点位置
    mp_r=MP[p,0]
    mp_c=MP[p,1]
    if mp_r!=-1 and mp_c!=-1:#如果有匹配点
        #视差计算
        dx+=mp_c-fp_c
        dy+=mp_r-fp_r
        num1+=1
#计算平均视差
dx0=dx/num1
dy0=dy/num1
```

在次遍历匹配点，筛选出视差与真值之差的绝对值在 5px 内的点作为最终匹配点。并且在图中连线标注，并计算此时的平均视差和视差方差。

```
#匹配连线，计算筛选后平均视差和方差
dx=dy=0
ddx=ddy=0
num2=0
for p in range(len(FP[0])):#遍历每个特征点
    #特征点位置
    fp_r=FP[0][p]
    fp_c=FP[1][p]
    #相应匹配点位置
    mp_r=MP[p,0]
    mp_c=MP[p,1]
    if mp_r!=-1 and mp_c!=-1:#如果有匹配点
        if abs(mp_c-fp_c-dx0)<=5 and abs(mp_r-fp_r-dy0)<=5:#视差与视差均值之差大于5像素的筛除
            #画点连线
            cv2.circle(concatImage, (fp_c,fp_r), 1, [0, 0, 255], 2, cv2.LINE_AA)#标注特征点
            cv2.circle(concatImage, (mp_c+l_w,mp_r), 1, [0, 0, 255], 2, cv2.LINE_AA)#标注匹配点
            cv2.line(concatImage,(fp_c,fp_r),(mp_c+l_w,mp_r),[0,255,0],1,8)#连线
            #视差计算
            dx+=mp_c-fp_c
            dy+=mp_r-fp_r
            #方差计算
            ddx+=(mp_c-fp_c-dx0)**2
            ddy+=(mp_r-fp_r-dy0)**2
            num2+=1
#最终平均视差
dx=dx/num2
dy=dy/num2
#最终方差计算
ddx=ddx/(num2-1)
ddy=ddy/(num2-1)
return concatImage,dx,dy,ddx,ddy,num1,num2
```

最后进行结果保存和输出。

```
print('------------------Correlation Coefficient Matching-----------------')
FP,MP=CCMatching(l_img,r_img,l_FP,threshold=0.85)#相关系数匹配
l_colorimg=cv2.imread('data\\panLeft.bmp')
r_colorimg=cv2.imread('data\\panRight.bmp')
result,dx,dy,ddx,ddy,num1,num2=drawMatchLine(l_colorimg,r_colorimg,FP,MP,l_h,l_w,r_h,r_w)#连线标注
cv2.imwrite('Result\\CorrectlationCoefficientMatching\\result.jpg',result)
print('为筛选匹配点个数:{}'.format(num1))
print('筛选后匹配点个数:{}'.format(num2))
print('左右平均视差:{}'.format(dx))
print('上下平均视差:{}'.format(dy))
print('左右视差方差:{}'.format(ddx))
print('上下视差方差:{}'.format(ddy))
print('Correlation Coefficient Matching Finish')
```

四、试验结果及分析

1. 点特征提取

1.1 结果分析

点特征提取部分我分别使用 chess.bmp 图像和 panLeft.bmp 图像进行了实验，具体结果如下。

（1）Moravec 算法

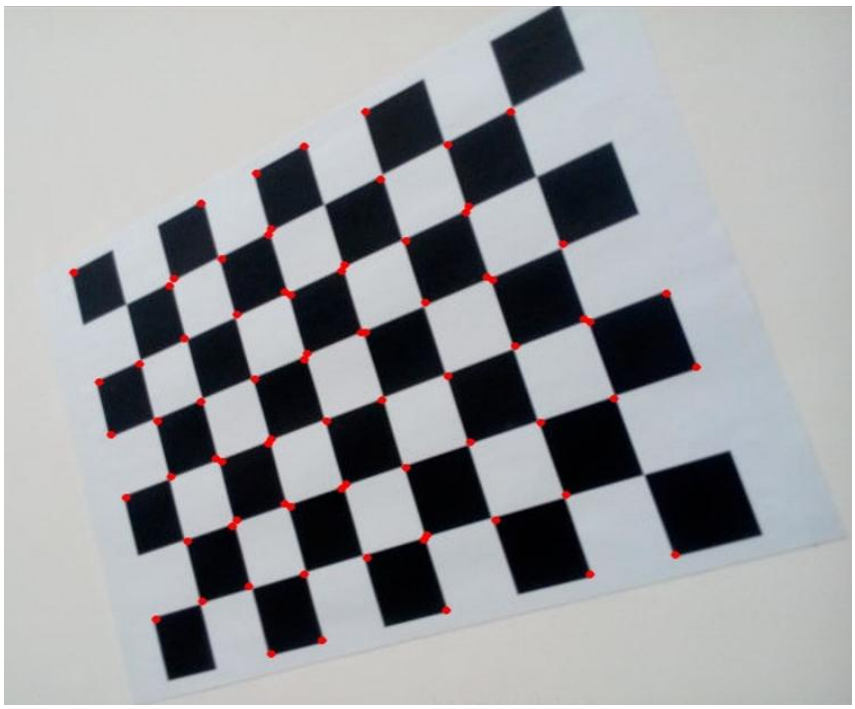
1>chess 图像结果：

经过试验，最终结果参数如下：

- 阈值：6000
- 兴趣窗口大小：5x5
- 抑制窗口大小：5x5

由结果图可以知道，Moravec 算法完成了大部分角点的提取，但是仍然有些许角点没有提取出来。值得一提的是这些未被提取出来的点大多数靠近图像边界。就运行速度而言，Moravec 算法比较快，因为其计算量并不大。总体而言，在小计算量的前提下，比较好的完成了特征点的提取工作。

本项目还保存了 Moravec 算法运行过程中的兴趣图、候选点图、特征点图，均可在 Result->PointFeatureExtraction->Moravec 文件夹中查看，在此就不过多引用。

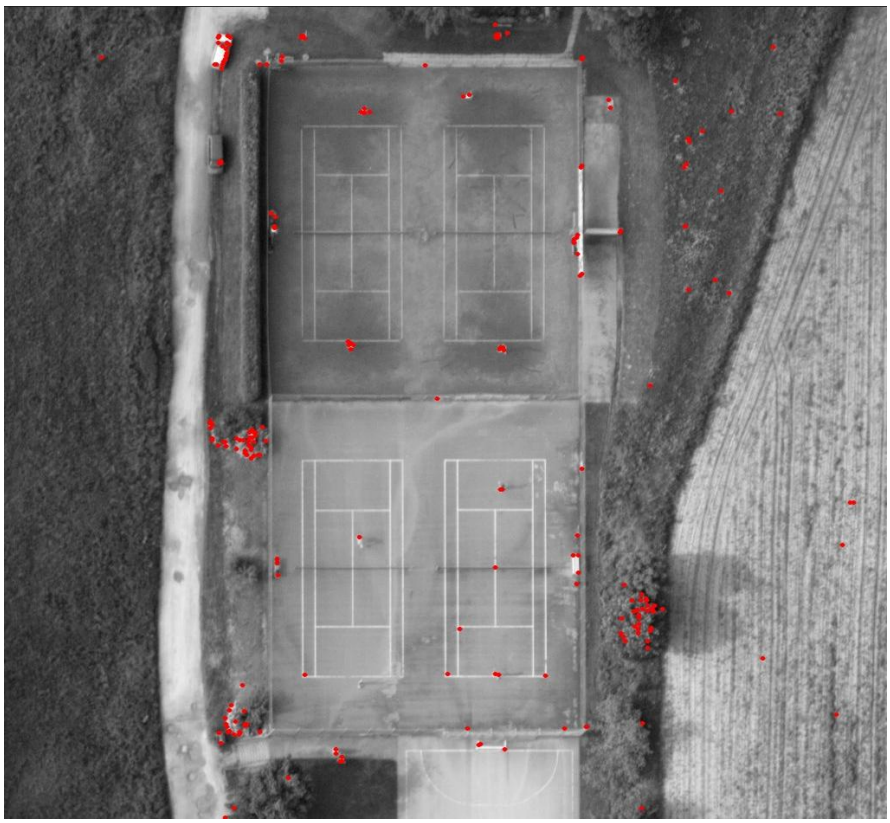


2>panLeft 图像结果

经过试验，最终结果参数如下：

- 阈值：1200
- 兴趣窗口大小：5x5
- 抑制窗口大小：5x5

从结果图可以看到 Moravec 算法将网球场上的人、角点都提取了出来。球场旁边的两棵树上特征点比较多，这应该是由于其灰度变化比较频繁。



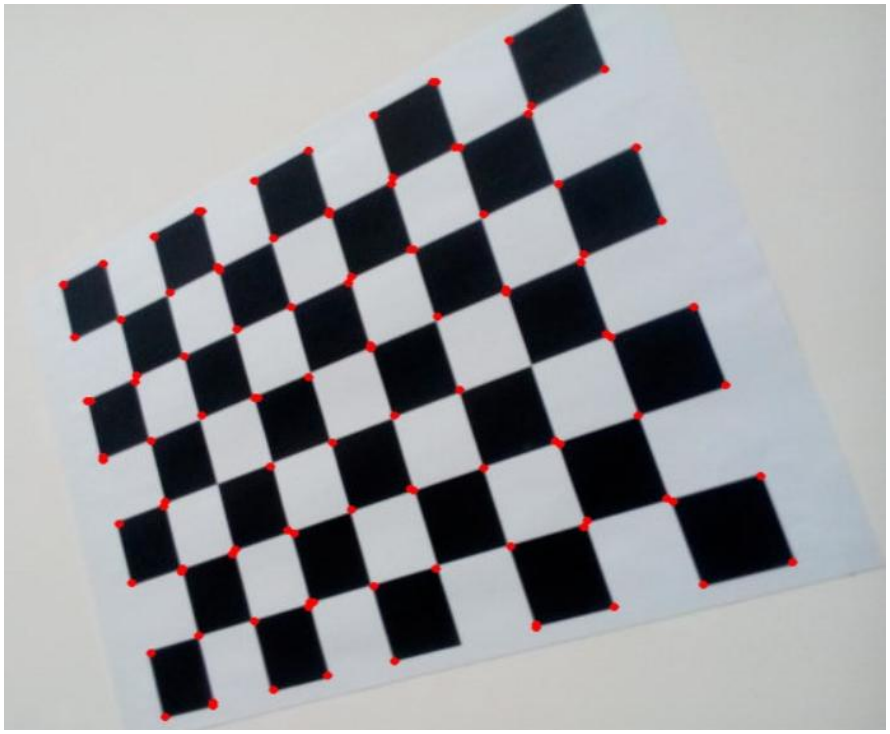
（2）Forstner 算法

1>chess 图像结果

经过试验，最终结果参数如下：

- 兴趣窗口大小：5x5
- 抑制窗口大小：7x7
- 初选阈值：20
- q 阈值：0.5
- f、c：0.5、5

由结果图可以知道，Forstner 算法完成了所有特征点的提取，效果上比 Moravec 算法更好，这是因为其计算的灰度协方差矩阵计算量更大，考虑的信息跟充分，但这导致了该算法在速度上较慢一些。

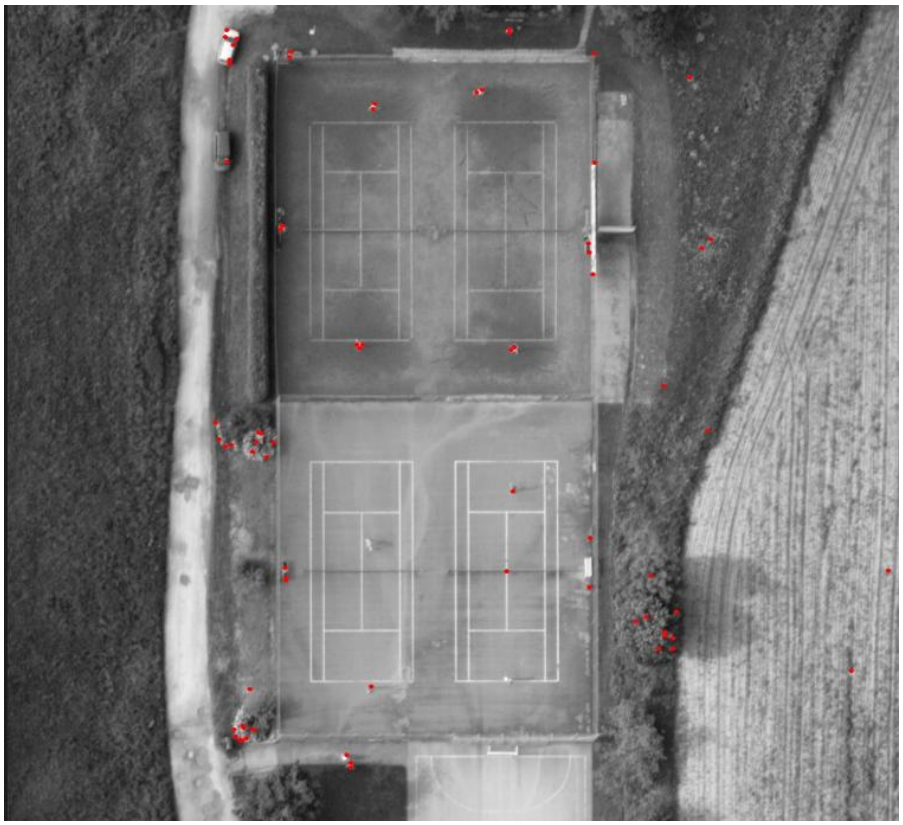


2>panLeft 图像结果

经过试验，最终结果参数如下：

- 兴趣窗口大小：5x5
- 抑制窗口大小：7x7
- 初选阈值：20
- q 阈值：0.5
- f、c：0.5、5

由结果图可以看到，使用 forstner 算法提取出的特征点较 Moravec 算法提取的特征点要少一些，经过分析，认为可能对于这种灰度变化较为细密的图像而言，forstner 的初始差分筛选会筛选掉一些不明显的特征点。



1.2 阈值分析

点特征提取中阈值会直接影响到特征点的数量，我做了多次试验（部分试验结果如下表）。

就 Moravec 算法而言，根据计算的兴趣值 IV，进行统计分析可以大致判断出较为合理的阈值在 1000 左右。然后设置多组不同阈值的实验，得到的特征点数目如下，根据结果图像分析，我认为阈值 T=1200 较为合适。当阈值过大时，一些特征点无法提取出来；当阈值过小时，很多非特征点会被提取出来。

阈值 T	特征点数目
1000	306
1100	253
1200	215
1400	152

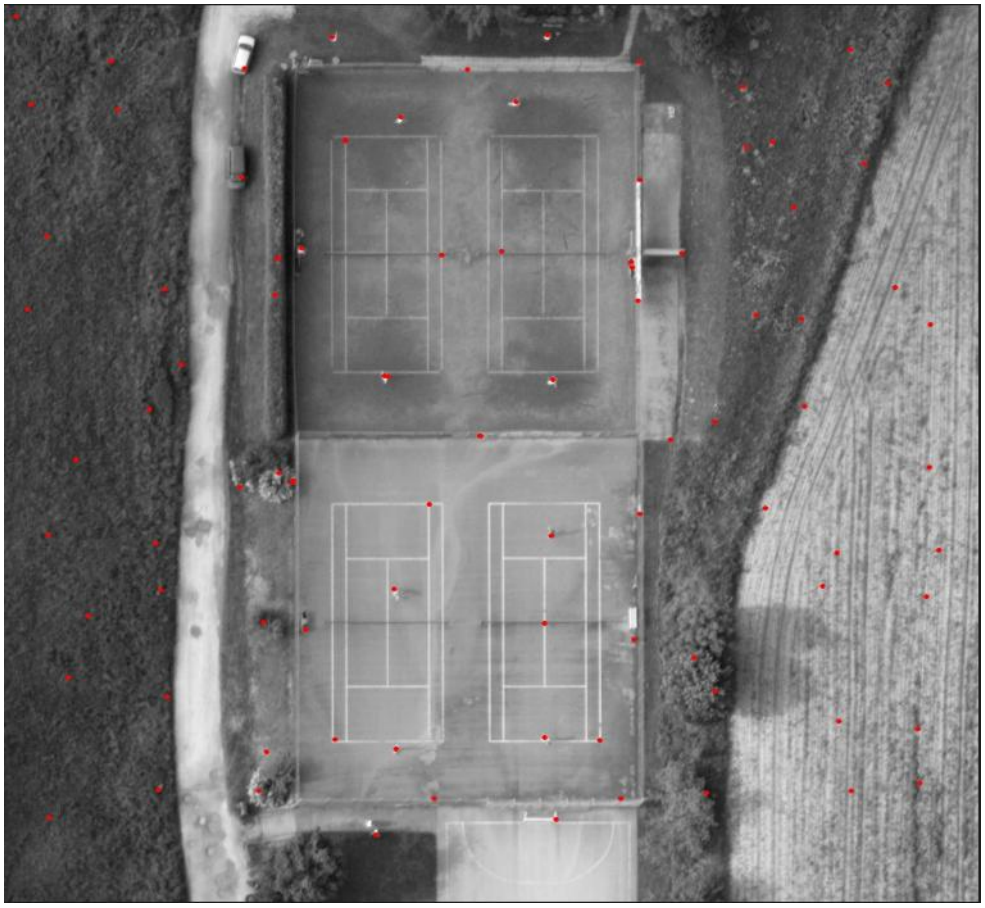
1.3 随机/均匀提取实施

特征点提取有两种模式，一是随机提取，即全图使用统一的阈值，只要满足条件则为特征点（上述结果图即为随机提取），但这样也可能会导致一些特征点过于集中（如上图中的树叶有很多特征点），而一些区域又没有特征点。

基于此第二种方法就是均匀提取，即在每个区域内使用一个阈值，以求达到特征点尽量均匀分布的目的，不过这样的操作也有可能导致提取出非特征点。

具体实施方法为：通过降低阈值（比如将 T=0），增大抑制窗口大小，来实现随机提取。降低阈值让全图大多数点成为候选点，增大抑制窗口，让一个抑制窗口内只能挑选出一个兴趣值最大的点作为特征点，这样就完成了均匀提取。操作和结果如下图：

```
moravec_test=moravec(image,T=0,noMax_size=100)
```



2. 相关系数法匹配

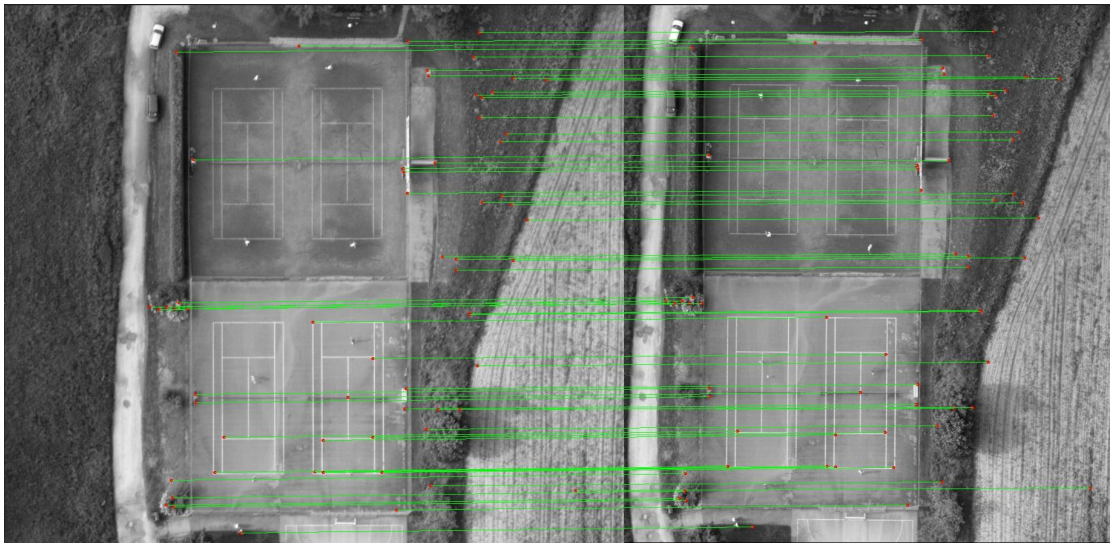
2.1 特征点随机分布

下面是随机目标点的匹配效果图，所用参数如下所示：

- Moravec 兴趣阈值：1000
- Moravec 兴趣窗口：5x5
- Moravec 抑制窗口：5x5
- 相关系数匹配计算窗口：7x7
- 相关系数匹配搜索区域：21x21
- 相关系数阈值：0.85

由计算结果可知，左右视差约为-175.94px，上下视差约为-6.19px。

```
------------------Read Image------------------
left image size:(942,1023)
right image size:(887,805)
------------------Feature Point Extraction------------------
特征点个数:141
Feature Point Extraction Finish
------------------Correlation Coefficient Matching-----------------
d:\本科\摄影测量\点特征提取及相关系数匹配\CorrelationCoefficientMatchi
e_scalars
  CC=sum1/(sqrt(sum2*sum3))
为筛选匹配点个数:104
筛选后匹配点个数:62
左右平均视差:-175.93548387096774
上下平均视差:-6.193548387096774
左右视差方差:5.734767557474049
上下视差方差:8.532544378698223
Correlation Coefficient Matching Finish
```



2.2 特征点均匀分布

下面是均匀目标点的匹配效果图，所用参数如下所示：

- Moravec 兴趣阈值：0
- Moravec 兴趣窗口：5x5
- Moravec 抑制窗口：100x100
- 相关系数匹配计算窗口：7x7
- 相关系数匹配搜索区域：21x21
- 相关系数阈值：0.85

由计算结果可知，左右视差约为-176.46px，上下视差约为-6.46px。

```
------------------Read Image------------------
left image size:(942,1023)
right image size:(887,805)
------------------Feature Point Extraction------------------
特征点个数:93
Feature Point Extraction Finish
------------------Correlation Coefficient Matching------------------
d:\本科\摄影测量\点特征提取及相关系数匹配\CorrelationCoefficientMatch
e_scalars
  CC=sum1/(sqrt(sum2*sum3))
为筛选匹配点个数:61
筛选后匹配点个数:37
左右平均视差:-176.45945945945945
上下平均视差:-6.45945945945946
左右视差方差:5.5869613903072555
上下视差方差:9.52191764459972
Correlation Coefficient Matching Finish
```