天津大学

计算机视觉实验报告

学　　院 智能与计算学部

专　　业 软件工程

学　　号 3020001267

姓　　名 王旭

目录

1 实验目标

使用 opencv 提供的 harris 角点检测函数以及 sift 角点检测函数完成图像的角点检测,并且按照要求分别使用绿色的加号、红色的圆圈进行标记。除此之外,在不直接调用 cv::cornerHarris 的情况下,按照课堂讲授方法及 ppt 内容实现 harris 角点检测。

2 程序

2.1 Harris 角点检测 opencv 调库实现

A．opencv 函数接口简介:

```
cv::cornerHarris(
  InputArray    src,
  OutputArray   dst,
  int  blockSize,
  int  ksize,
  double    k,
  int  borderType = BORDER_DEFAULT)
```

opencv 提供的 Harris 角点检测函数有 5 个需要提供的参数，

src：需要进行角点检测的图像

dst：Harris 角点检测后的响应值矩阵

blockSize ：指定计算每个响应值时所使用的邻域大小

ksize：计算梯度时 sobel 算子的大小，

k：角点检测参数。

一般的 blockSize，ksize，k 取值为 2，3，0.04

B．程序片段（带注释）：

```
Mat harris_matrix(Mat& img) {
    Mat gray,dst;
    //灰度值转换
    cvtColor(img, gray, COLOR_BGR2GRAY);
```

```
    //使用blocksize=2，ksize=3，k=0.04计算角点响应值
    cornerHarris(gray, dst, 2, 3, 0.04);
    return dst;
    }
```

```
// Harris角点检测的非极大值抑制+阈值抑制
Mat harris_nms(Mat& dst,int WINDOW_SIZE=3,float thers=0.0001)
{
    Mat dst_nms = dst.clone();
    //响应值小于指定阈值的变为0
    for (int i = 0; i < dst_nms.rows; i++)
    {
        for (int j = 0; j < dst_nms.cols; j++)
        {
            if (dst_nms.at<float>(i, j) < thers) {
                dst_nms.at<float>(i, j) = 0;
            }
        }
    }



    float eps = 1e-21;
    float max_resp = dst_nms.at<float>(0, 0);
    // 遍历所有像素点
    for (int i = 0; i < dst_nms.rows; i++)
    {
        for (int j = 0; j < dst_nms.cols; j++)
        {
            // 窗口的左上角和右下角坐标
            int x1 = max(j - WINDOW_SIZE / 2, 0);
            int y1 = max(i - WINDOW_SIZE / 2, 0);
            int x2 = min(j + WINDOW_SIZE / 2, dst_nms.cols - 1);
            int y2 = min(i + WINDOW_SIZE / 2, dst_nms.rows - 1);

            // 寻找窗口内的最大值
            float max_val = dst_nms.at<float>(i, j);
            for (int y = y1; y <= y2; y++)
            {
                for (int x = x1; x <= x2; x++)
                {
                    if (dst_nms.at<float>(y, x) > max_val)
                    {
                        max_val = dst_nms.at<float>(y, x);
```

```
                    }
                }
            }

            max_resp = max(max_resp, max_val);

            // 如果该点不是局部最大值，则将其响应值设为0
            if (dst_nms.at<float>(i, j)< max_val-eps )
            {
                dst_nms.at<float>(i, j) = 0;
            }
        }
    }
    //将所有留下来的响应值置为255方便显示
    threshold(dst_nms, dst_nms, thers, 255, THRESH_BINARY);

    return dst_nms;
    }
```

```
void harris_draw(Mat& dst, Mat& img1) {
    // 消除函数副作用
    Mat img = img1.clone();
    //遍历所有响应值，若响应值为255则画加号
    for (int i = 0; i < dst.rows; i++) {
        for (int j = 0; j < dst.cols; j++) {
            if (dst.at<float>(i, j) == 255) {
                line(img, Point(j - 10, i), Point(j + 10, i), Scalar(0, 255, 0));
                line(img, Point(j, i - 10), Point(j, i + 10), Scalar(0, 255, 0));
            }
        }
    }
    imshow("harris角点检测", img);
}
```

```
int main()
{
    //读入图片
    Mat img = imread("D:\\study\\CV\\assignment2\\20200222031859760.jpg");
    imshow("原图像", img);
    //生成角点响应矩阵
    Mat dst = harris_matrix(img);
    //对角点响应矩阵做非极大值抑制
    Mat dst_nms = harris_nms(dst,3,0.01);
```

```
    //工具响应值画出加号
    harris_draw(dst_nms, img);

    //sift_detect(img);

    waitKey(0);
    destroyAllWindows();
    return 0;
}
```

C．程序说明:

harris_matrix 调用 opencv 提供的 harris 角点检测得到角点响应矩阵。

harris_nms 有三个参数 Mat& dst,int WINDOW_SIZE=3,float thers=0.0001， dst 是角点响应矩阵，首先将响应值小于 thers 的置为 0，然后才做非极大值抑制，非极大值抑制算法步骤如下：

1. 遍历角点响应矩阵上的元素，以该元素为中心点选取一个窗口（窗口大小由参数 WINDOW_SIZE 指定）。
2. 找到窗口内的最大值。
3. 比较该最大值和窗口中心点的响应值，如果最大值不是中心点的响应值，则将中心点的响应值设为零。
4. 重复以上步骤直到矩阵每个位置都被遍历。

这个过程将去除角点响应图像中不是局部最大值的值，使最终的角点集合只包含响应值最大的那些点，即非极大值抑制。

然后 harris_draw 接收做了非极大值抑制的角点响应矩阵，在保留下来的非 0 值的位置上通过横、竖画一个直线，实现画加号的目的。

Main 函数依次调用以上三个函数即可在所有角点上画加号。

2.2 Sift 角点检测 opencv 调库实现

A．opencv 函数接口:

SIFT::create(

int nfeatures = 0,

int nOctaveLayers = 3,

double contrastThreshold = 0.04,

double edgeThreshold = 10,

double sigma = 1.6)

nfeatures：指定特征点的数量，默认为 0，表示不限制数量。

nOctaveLayers：指定金字塔中每组的层数，默认为 3。

contrastThreshold：用于控制图像特征点检测的灵敏度，值越大检测到的特征点越少。

edgeThreshold：用于过滤掉边缘响应点。

sigma：指定高斯金字塔的初始尺度。

一般来说使用默认值即可。

SIFT::detect(

cv::InputArray image,

std::vector<cv::KeyPoint>& keypoints,

cv::InputArray mask = cv::noArray())

image：输入图像。

keypoints：用于存储检测到的特征点。

mask：用于指定感兴趣区域，只在指定区域内检测特征点。

B．程序片段（带注释）：

```
void sift_detect(Mat& img1) {
    Mat gray, img;
    //消除副作用
    img = img1.clone();
    cvtColor(img, gray, COLOR_BGR2GRAY);
    //创建sift检测
    Ptr<SIFT> sift = SIFT::create();
    vector<KeyPoint> keypoints;
    //结果存放在keypoints这个vector中
    sift->detect(gray, keypoints);
    //遍历vector，并在响应位置画出红色的圆圈
    for (int i = 0; i < keypoints.size(); i++)
    {
        circle(img, keypoints[i].pt, 5, Scalar(0, 0, 255), 2); // Draw red "o" marker
    }
    imshow("sift角点检测", img);
  }


int main()
{
    //读入图片
    Mat img = imread("D:\\study\\CV\\assignment2\\20200222031859760.jpg");
    imshow("原图像", img);

    sift_detect(img);
```

```
    waitKey(0);
    destroyAllWindows();
    return 0;
}
```

C．程序说明:

在 sift_detect 首先对图像进行灰度化处理，然后使用 SIFT::create()创建一个 SIFT 检测器，接着使用 sift->detect()函数对图像进行角点检测，将检测结果保存在 vector<KeyPoint>类型的变量 keypoints 中。

最后，遍历检测结果，对每个检测到的角点在图像上画出红色的圆圈。具体而言，对于第 i 个检测到的角点，使用 circle()函数在图像 img 上画出以 keypoints[i].pt 为中心，半径为 5 的红色圆圈。最后，使用 imshow()函数将带有角点检测结果的图像显示出来。

2.3 Harris 角点检测手写实现

A．算法流程:

按照 PPT 内容，算法（不包括非极大值抑制）流程大致如下：

1. 由 ksize 指定 sobel 算子，计算图像的 X、Y 方向上的梯度
2. 逐像素计算响应值，先取出这个位置周围 blockSize 大小矩形的 x，y 梯度值
3. X 方向梯度逐像素求乘积在求和得 Ixx，Y 方向同理 Iyy，X 方向与 Y 方向梯度逐像素求乘积再求和得 Ixy
4. R=Ixx*Iyy-Ixy*Ixy-k*(Ixx + Iyy) * (Ixx + Iyy)
5. 重复 2~4 步骤直到所有位置都被遍历

B.需要用到的相关操作说明:

```
cv::Sobel(
    InputArray src,
    OutputArray dst,
    int ddepth,
    int dx, int dy,
    int ksize = 3,
    double scale = 1,
    double delta = 0,
    int borderType = BORDER_DEFAULT
```

);

src：要处理的输入图像，可以是任何支持的 Mat 类型。

dst：处理后的输出图像，与输入图像具有相同的尺寸和深度。

ddepth：输出图像的深度，可以是 CV_8U、CV_16U、CV_32F 等。如果指定输出深度为 CV_8U，则 Sobel 算子的值将被截断为 8 位无符号整数。默认情况下，输出深度与输入深度相同。

dx 和 dy：x 方向和 y 方向上的求导阶数，可以是 0、1、2。通常情况下，将 dx 或 dy 指定为 1 即可，这将在相应方向上应用一阶导数。

ksize：Sobel 算子的大小，可以是 1、3、5、7。通常情况下，将其设置为 3 即可。

scale：结果的比例因子，可以是任意 double 类型的值，默认为 1。通常情况下，将其设置为 1 即可。

delta：结果的偏移量，可以是任意 double 类型的值，默认为 0。通常情况下，将其设置为 0 即可。

borderType：边界处理方式，可以是 BORDER_CONSTANT、BORDER_REPLICATE、BORDER_REFLECT 等。默认情况下，使用 BORDER_DEFAULT 进行边界处理。

Mat（Rect），用 Rect 指定的矩形的大小，左上角所在位置截取 Mat，相当于 python 中的 Mat[x:x+dx,y:dy]

dot() 函数用于计算两个矩阵的点乘积。点乘积是将两个矩阵中对应元素的乘积相加得到的一个标量值。dot(A, B) = sum(A[i][j] * B[i][j]), i = 0,...,n-1, j = 0,...,m-1,需要 A、B 有相同的大小。

C.程序片段（带注释）：

```
Mat harris_R(Mat& img) {
    Mat gray, dst;
    //彩色三通道图像转换为灰度图像
    cvtColor(img, gray, COLOR_BGR2GRAY);
    //三个参数同opencv的三个参数含义一致
    int blockSize = 2;
    int ksize = 3;
    double k = 0.04;
    //expand_pixel ，对图像边缘处的特殊处理
    int expand_p = blockSize / 2;
    //使用sobel算子,求图像在x、y处的导数
    Mat ImgSobelX, ImgSobelY;
    Sobel(gray, ImgSobelX, CV_32F, 1, 0, ksize);
    Sobel(gray, ImgSobelY, CV_32F, 0, 1, ksize);

    //在图像边缘加上一圈expand_p，对图像边缘处的特殊处理
```

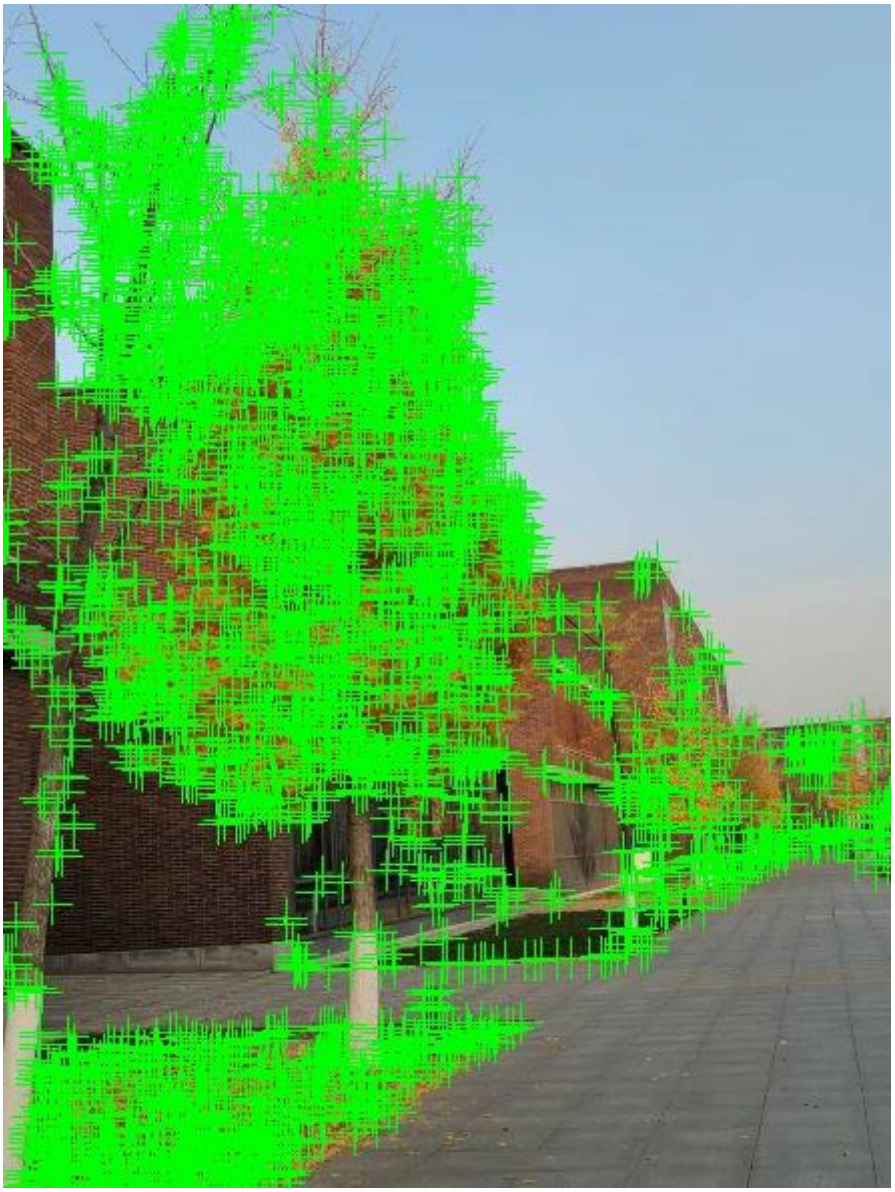
```
    Mat Operate_SX = Mat(ImgSobelX.rows + 2 * expand_p, ImgSobelX.cols + 2 * expand_p, CV_32FC1, Scalar(0));
    Mat Operate_SY = Mat(ImgSobelY.rows + 2 * expand_p, ImgSobelY.cols + 2 * expand_p, CV_32FC1, Scalar(0));
    //Mat(Rect) 拿一个矩阵区域去截取Mat这个矩阵，这里是ImgSobelX填充到了Operate_SX的非边缘区域，Operate_SY同理
    Rect rect = Rect(expand_p, expand_p, ImgSobelX.cols, ImgSobelX.rows);
    ImgSobelX.copyTo(Operate_SX(rect));
    ImgSobelY.copyTo(Operate_SY(rect));

    Mat resultImage = Mat(ImgSobelX.rows, ImgSobelX.cols, CV_32FC1, Scalar(0));

    //遍历resultImage每一个位置计算响应值
    for (int i = expand_p; i < resultImage.rows; i++)
    {
        for (int j = expand_p; j < resultImage.cols; j++)
        {
            //对于i,j 截取Operate_SX(j - expand_p~j - expand_p+blockSize, i - expand_p~i - expand_p+blockSize)的矩形区域
            //计算响应值，Operate_SY同理
            Rect rec = Rect(j - expand_p, i - expand_p, blockSize, blockSize);
            Mat Ix = Operate_SX(rec);
            Mat Iy = Operate_SY(rec);
            //dot运算，逐位置计算乘积然后累加
            float Ixx = Ix.dot(Ix);
            float Ixy = Ix.dot(Iy);
            float Iyy = Iy.dot(Iy);
            //响应值计算
            float R = Ixx * Iyy - Ixy * Ixy - k * (Ixx + Iyy) * (Ixx + Iyy);
            resultImage.at<int>(i - expand_p, j - expand_p) = (int)R;


        }

    }
    return resultImage;
  }
```

3 结果与讨论

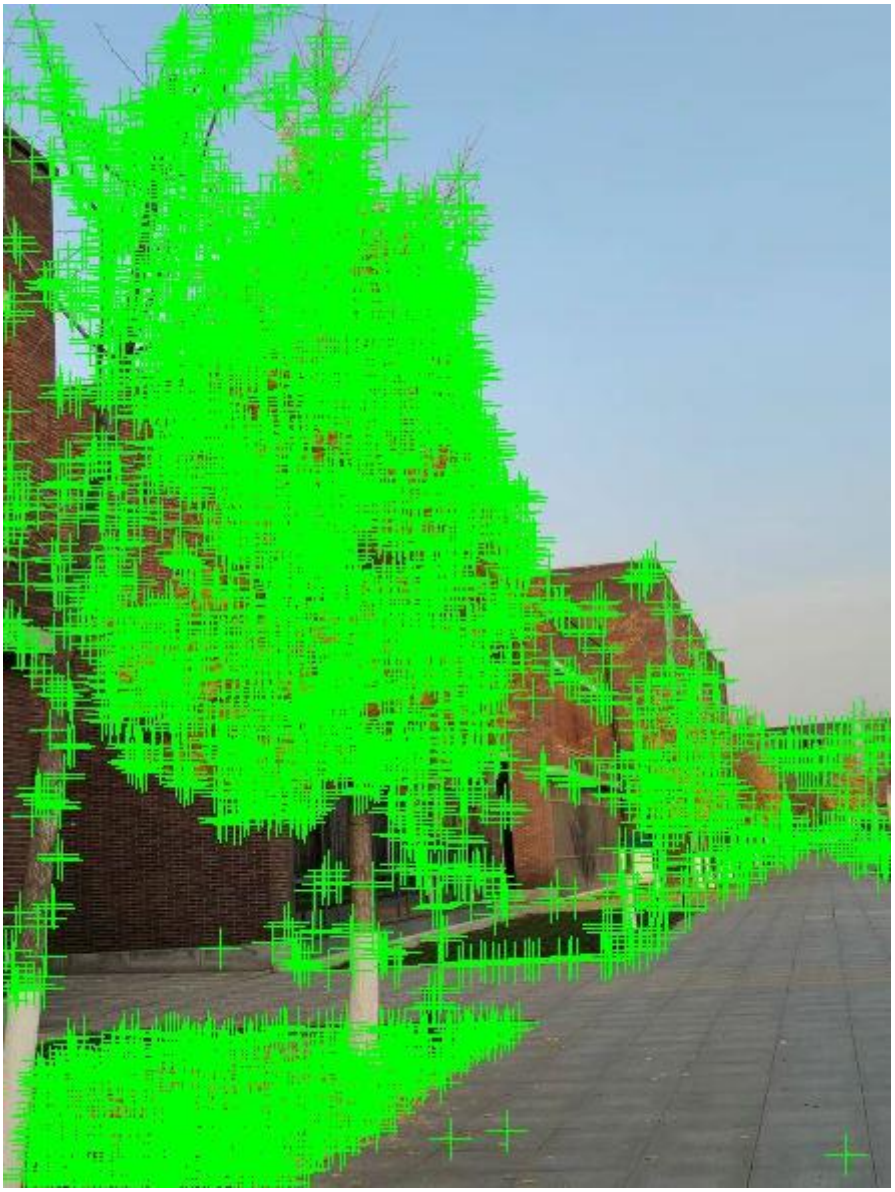
3.1 实验结果

A． 原图像

B．Harris 角点检测调库实现结果



C．Harris 角点检测手写实现结果



D． sift 检测调库实现结果