

**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 3](#_Toc129696719)

[2 程序 3](#_Toc129696720)

[2.1 Harris角点检测opencv调库实现 3](#_Toc129696721)

[2.2 Sift角点检测opencv调库实现 6](#_Toc129696722)

[2.3 Harris角点检测手写实现 8](#_Toc129696723)

[3 结果与讨论 10](#_Toc129696724)

[3.1 实验结果 11](#_Toc129696725)

# 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。

# 程序

## 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。

## 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。

## Harris角点检测手写实现

1. 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;

}

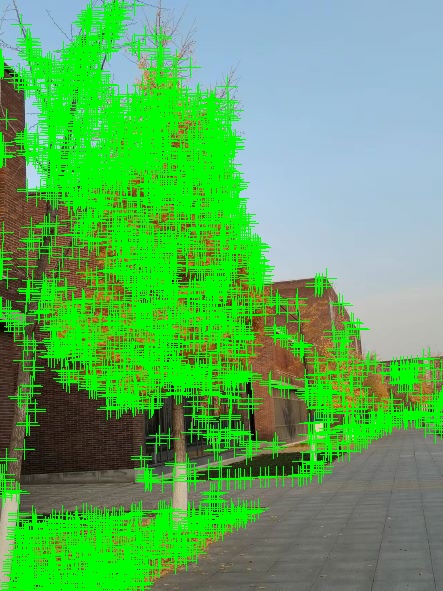
# 结果与讨论

## 实验结果

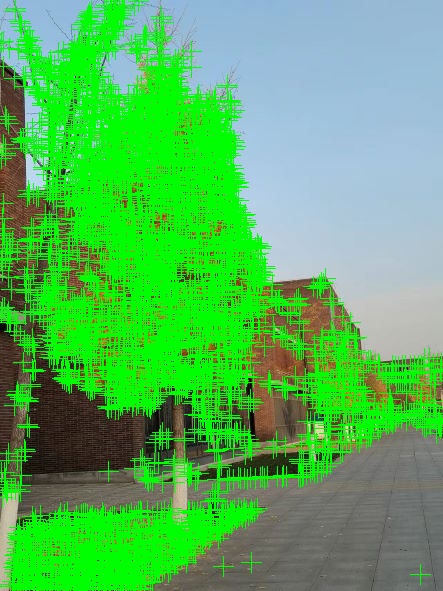
1. 原图像



1. Harris角点检测调库实现结果



1. Harris角点检测手写实现结果



1. sift检测调库实现结果

