Harris 角点检测原理及 OpenCV 代码描述

Dezeming Family

Dezeming

Art. No 0
ISBN 000–00–0000–00–0
Edition 0.0

Cover design by Dezeming Family

Published by Dezeming
Printed in China

目录

前言及简介

DezemingFamily 系列书和小册子因为是电子书，所以可以很方便地进行修改和重新发布。如果您获得了 *DezemingFamily* 的系列书，可以从我们的网站 *[https://dezeming.top/]* 找到最新版。对书的内容建议和出现的错误欢迎在网站留言。

0.1 本书前言

角点检测在计算机视觉中应用非常广泛，例如图像匹配、特征提取以及相机标定、三维重建等。其中，Harris 角点检测算法是其中非常基础和流行的角点检测方法。

本书会介绍 Harris 算法的基本原理，包括算法原理和数学原理。然后细致讲解 OpenCV3 中的 Harris 角点检测代码的应用方法。对于功能强大的 OpenCV，它的源码非常值得学习，因此我们会重点讲解源码实现。

因为 Harris 很简单，所以没有参考太多资料，仅仅对照一下网络描述的基本流程 [1][2][3]。但网上网友对于 Harris 的描述和理解实在是难以接受，可能很多博主只是脑袋里有个模糊的印象就写下来了，不过还是要感谢他们的辛勤付出和努力。角点检测的理解，尤其是特征值和 SSD 之间的关系和网上一般的描述有很大区别，因此在写作时我也只是参考一下他们讲了什么内容，以及规划一下我自己要讲什么内容。

本书定价为 7 元，但用户可以免费获取并使用。如果您得到了本书，在学习中对自己有所帮助，可以往 DezemingFamily 的支付宝账户中支持 7 元。您的支持将是我们 DezemingFamily 发布更多电子书和努力提高电子书质量的动力！

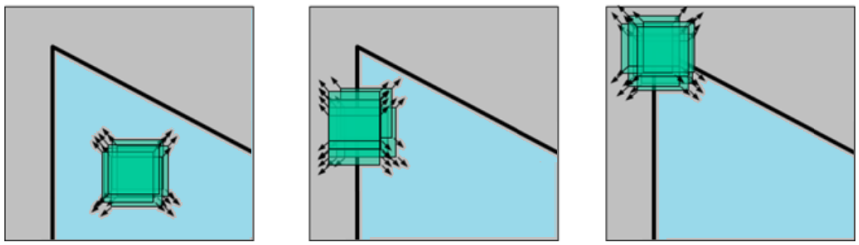
1. Harris 算法原理

本章介绍 *Harris* 角点检测算法的基本原理，包括算法和数学原理。

1.1 Harris 角点检测基础

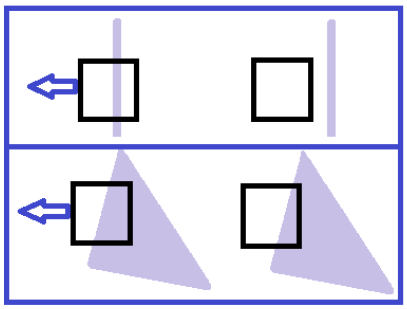
在图像匹配中，我们一般提取图像局部特征，有几个好处：特征是局部的，所以对图像上大范围遮挡和杂乱的位置表现非常鲁棒；局部特征也具有一定独特性，作为可以区分对象的大型数据库（数量多）；局部性因此计算量少，可实时实现特征检测；对特征有一定概括性，在不同情况下可以利用不同类型的功能。因此，在图像局部区域中，我们认为特征是具有颜色明显变化的位置，我们这里介绍一下图像的角点检测（准确来说角点不是特征，但检测出来的角点可以用来提取和表示总结为特征）。

在局部小范围里，如果在各个方向上移动窗口，窗口内区域的灰度发生较大变化，就认为在窗口内遇到了角点；如果窗口在各个方向移动时，窗口内图像的灰度没有发生变化，那么窗口内就不存在角点。



特征分为两种：如果窗口在某个方向移动时，窗口内图像的灰度发生了较大的变化，而在另一些方向上没有发生变化，那么，窗口内的图像可能就是边缘区域或者是单独的一条线。如果向任意方向移动，窗口灰度变化都很大，那么可能就是尖角区域或者是比较突出的点（比如夜空照片中的星星）。当然我们一定要注意的是特征的大小和窗口的大小需要密切相关。

当然，对于线来说，当窗口移动到线外，灰度才会有明显变化。对于物体边缘，窗口只要不沿着线移动，灰度就会有明显变化：

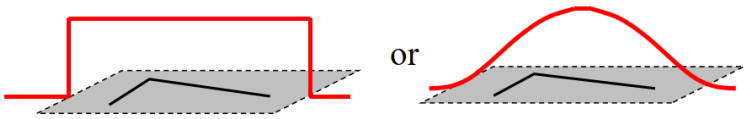


我们下面以 Harris 角点检测方法为例，讲解角点检测的原理。虽然 Sift 特征和 Harris 角点检测算法不同，但 Harris 作为基础算法，还是应该首先掌握。

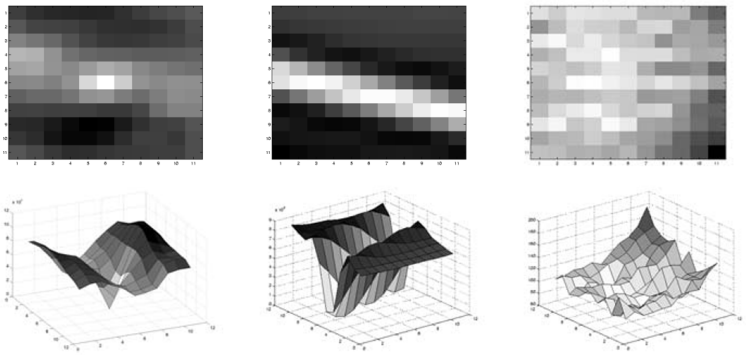
对于局部区域来说，局部块（局部窗）可以向四周移动，我们假设当前窗内的每个像素坐标为 (x_i, y_j)，其中 $(x,y) \in W$，当我们令窗偏移 (u,v) 像素时，则我们定义差分平方的加权求和 (weighted Summed Square Difference，简写为 SSD) 为:

$$E(u,v) = \sum_{(x,y)\in W} \omega(x,y)[I(x+u,y+v) - I(x,y)]^2 \tag{1.1}$$

其中 $\omega(x,y)$ 表示每个像素的权重，常用的权重核为高斯核，也可以设置全都是 1：



对于不同的图像局部区域，我们改变 (u,v) 就可以得到以 u 和 v 为坐标的 SSD 图像，如下图。局部区域出现点时，SSD 图像表现为坑洞；局部区域为边缘时，表现为沟壑；当局部区域为平缓或者杂乱时，则没有明显变化。



1.2 自相关系数矩阵

我们对 SSD 公式进行泰勒展开，$o(I)$ 表示高阶小项：

$$I(x+u,y+v) = I(x,y) + \frac{\partial I}{\partial x}u + \frac{\partial I}{\partial y}v + o(I) \tag{1.2}$$

去掉高阶小项，令 $I_x = \frac{\partial I}{\partial x}$，$I_y = \frac{\partial I}{\partial y}$，表示像素在 x 和 y 方向的梯度（变化强弱），可以近似为：

$$I(x+u,y+v) \approx I(x,y) + \begin{bmatrix} I_x & I_y \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} \tag{1.3}$$

这时我们用近似值计算 SSD：

$$E(u,v) = \sum_{(x,y)\in W} \omega(x,y)[I(x+u,y+v) - I(x,y)]^2 \tag{1.4}$$

$$\approx \sum_{(x,y)\in W} \omega(x,y)[I(x,y) + \begin{bmatrix} I_x & I_y \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} - I(x,y)]^2 \tag{1.5}$$

$$\approx \sum_{(x,y)\in W} \omega(x,y)[\begin{bmatrix} I_x & I_y \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix}]^2 \tag{1.6}$$

上面的式子可以进行化简，我们先把这一项内容拎出来：

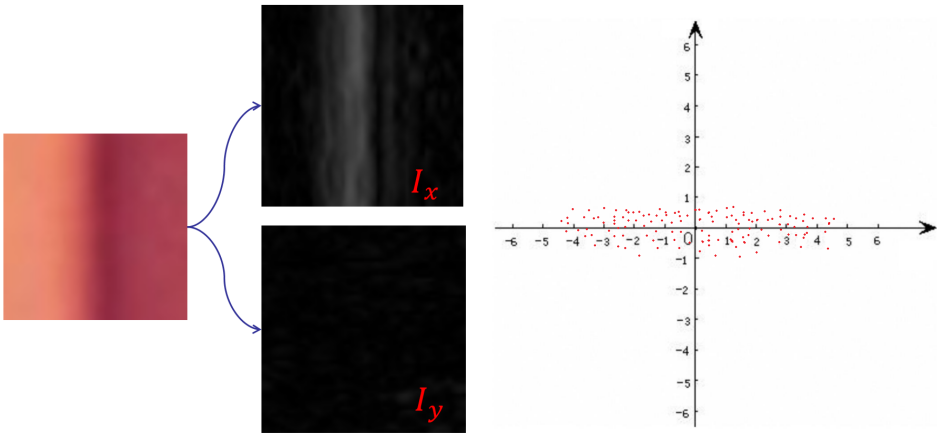
$$[\begin{bmatrix} I_x & I_y \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix}]^2 = \begin{bmatrix} u & v \end{bmatrix} \begin{bmatrix} I_x^2 & I_x I_y \\ I_y I_x & I_y^2 \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} \tag{1.7}$$

我们发现 $\begin{bmatrix} u & v \end{bmatrix}$ 是可以单独拿出来到 $\sum$ 外面的，因此近似的 SSD 可以写为：

$$E(u,v) \approx \begin{bmatrix} u & v \end{bmatrix} \left(\sum_{(x,y)\in W} \omega(x,y) \begin{bmatrix} I_x^2 & I_x I_y \\ I_y I_x & I_y^2 \end{bmatrix} \right) \begin{bmatrix} u \\ v \end{bmatrix} \tag{1.8}$$

我们把中间括号里的部分称为自相关系数矩阵 A，之后，对角点特征的分析就转换为了为自相关系数矩阵 A 的分析。

再次明确一下我们的目标：就是研究找像素点，该像素无论在哪个方向上，SSD 都会有很大的值。即 (u,v) 无论沿着水平轴变化还是沿着竖直轴变化，$E(u,v)$ 的值都会有很大的值，只有这样的点才是角点。我们显示一下边缘区域的图像梯度，可以看到，在 x 方向变化很明显，而在 y 方向变化不是很明显。当把这个区域的每个像素的 (I_x, I_y) 显示在坐标轴上时，表示如下，可以看到，I_x 的范围比 I_y 的范围要大得多了（注意 I_x 和 I_y 图像中越白表示梯度绝对值越大）：



对于平坦区域，I_x 和 I_y 的变化范围都很小；对于角点区域，I_x 和 I_y 的变化范围都很大。（我们检测是否为角点需要其统计意义上的信息，这是通过分析矩阵 A 来进行的。）

1.3 特征值与特征向量

矩阵 A 的特征向量 $\boldsymbol{x}$ 表示如下，对于矩阵 A，如果存在向量 $\boldsymbol{x}$ 和标量 λ：

$$A\boldsymbol{x} = \lambda\boldsymbol{x} \tag{1.9}$$

则称 $\boldsymbol{x}$ 为矩阵 A 的特征向量，λ 为矩阵 A 对应于特征向量 $\boldsymbol{x}$ 的特征值。我们化简一下上式，I 表示单位矩阵：

$$(A - \lambda I)\boldsymbol{x} = \boldsymbol{0} \tag{1.10}$$

这其实就相当于齐次方程式有解：

$$\begin{bmatrix} A_{1,1} & A_{1,2} \\ A_{2,1} & A_{2,2} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \tag{1.11}$$

有解的条件是行列式必须为 0（回顾线性代数基础），因此可得：

$$det(A - \lambda I) = det\left(\begin{bmatrix} A_{1,1} - \lambda & A_{1,2} \\ A_{2,1} & A_{2,2} - \lambda \end{bmatrix} \right) = 0 \tag{1.12}$$

解得（2×2 矩阵求行列式直接用对角相乘再相减即可）：

$$\lambda_{\pm} = \frac{1}{2}\left[(A_{1,1} + A_{2,2}) \pm \sqrt{4A_{1,2}A_{2,1} + (A_{1,1} - A_{2,2})^2} \right] \tag{1.13}$$

求解出 λ，就可以利用 λ 再求出特征向量来了。

1.4 特征与二次型

上面已经介绍过自相关系数矩阵 A：

$$A = \sum_{(x,y)\in W} \omega(x,y) \begin{bmatrix} I_x^2 & I_x I_y \\ I_y I_x & I_y^2 \end{bmatrix} \tag{1.14}$$

$$= \begin{bmatrix} \sum_{(x,y)\in W} \omega(x,y) I_x^2 & \sum_{(x,y)\in W} \omega(x,y) I_x I_y \\ \sum_{(x,y)\in W} \omega(x,y) I_y I_x & \sum_{(x,y)\in W} \omega(x,y) I_y^2 \end{bmatrix} \tag{1.15}$$

$$= \begin{bmatrix} A_{1,1} & A_{1,2} \\ A_{2,1} & A_{2,2} \end{bmatrix} \tag{1.16}$$

我们分析 $E(u,v)$ 的形式可得，$E(u,v)$ 随 (u,v) 而变化，其值仅跟 A 有关，故我们需要研究 A 是否存在某种特性，使得 (u,v) 在某个局部区域变化时，$E(u,v)$ 都能达到很大的值（联想第一节的 SSD 图像中角点显示为坑洞，这是因为局部区域向其他任何方向移动以后，得到的 $E(u,v)$ 值都很大）。

其中 $A_{1,2} = A_{2,1}$，说明这是个对称矩阵，又叫做二次型矩阵（见 DezemingFamily 的《矩阵二次型》，其实《矩阵二次型》对后面讲到的二次型与椭圆有非常细致的描述）。SSD 写为二次型矩阵的形式：

$$E(u,v) = \begin{bmatrix} u & v \end{bmatrix} A \begin{bmatrix} u \\ v \end{bmatrix} \tag{1.17}$$

我们可以将上面的二次型表示为一个椭圆，注意椭圆的公式：

$$a^2(x-a_1)^2+b^2(y-b_1)^2=1 \tag{1.18}$$

其中椭圆公式里展开后 x^2 和 y^2 项的系数都是大于 0 的，我们的二次型也是符合这个标准（因为 $A_{1,1}$ 和 $A_{2,2}$ 都大于 0）。

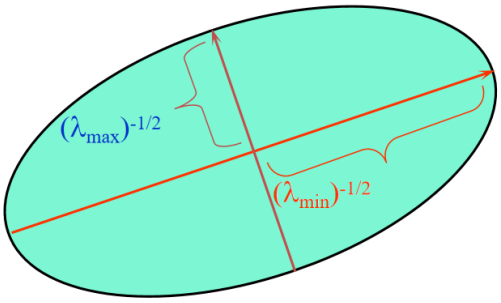
我们回顾高中学习的椭圆公式，假设椭圆公式中的 a_1 和 b_1 都是 0，则 a 和 b 越大，则椭圆面积越小，a 比 b 越大或者 b 比 a 越大，则椭圆越扁。

SSD 化为二次型的标准型，其中 λ_1 和 λ_2 为特征向量：

$$E(u,v)=\begin{bmatrix}u & v\end{bmatrix}A\begin{bmatrix}u\\v\end{bmatrix} \tag{1.19}$$

$$=\begin{bmatrix}u' & v'\end{bmatrix}\begin{bmatrix}\lambda_1 & 0\\0 & \lambda_2\end{bmatrix}\begin{bmatrix}u'\\v'\end{bmatrix} \tag{1.20}$$

令 $E(u,v)$ 为常数，得到该标准型的椭圆形式，其中二次方项的系数分别为 λ_1 和 λ_2，因此得到椭圆的形状为（设较大的特征值为 λ_{max}，较小的特征值为 λ_{min}）：



注意 λ 越大说明椭圆的该半轴越短。注意特征值反应了相互垂直的方向上变化情况，其中 λ_{max} 对应了变化最快的方向，λ_{min} 对应了变化最慢的方向（参考 DezemingFamily 的《特征值与特征向量》和《矩阵二次型》有详细解释，我还特别添加了关于二次型和特征向量的描述），即 (u,v) 沿着 λ_{max} 表示的方向变化时，可以用最快速度达到我们设定的 $E(u,v)$ 常数，当 (u,v) 沿着 λ_{min} 表示的方向变化时，会以最慢的速度达到我们设定的 $E(u,v)$ 常数。

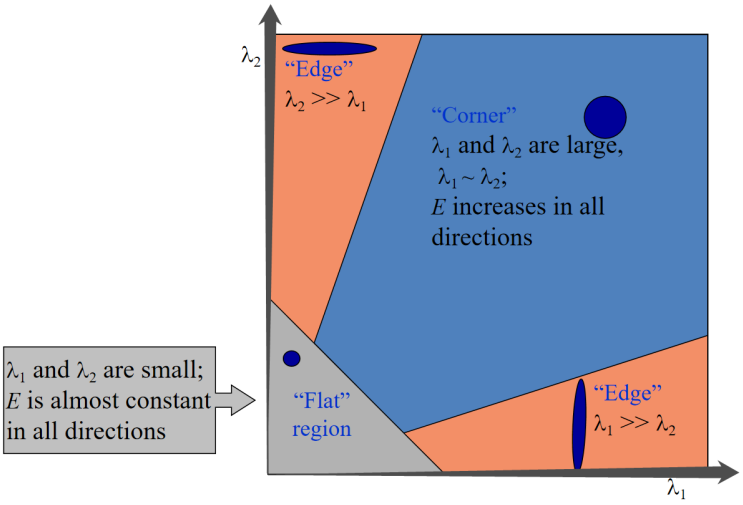
现在我们恢复 $E(u,v)$ 的自由变化，$E(u,v)$ 的值是一个随着 u 和 v 变化时也在变化的值了，这可以用一个三维图来表示。当 $(u,v)=(0,0)$ 时，显然 $E(u,v)$ 的值也是 0。当两个 λ 越大时，u 和 v 变大后 $E(u,v)$ 才会越大（回忆第一节角点区域的 SSD 表示为一个坑洞）。因此我们希望 λ_{min} 和 λ_{max} 都要足够大才可以。

1.5 Harris 特征检测

当特征值 λ_1 和特征值 λ_2 都很大时，I_x 和 I_y 在各个方向都能快速变到很大（I_x 和 I_y 能快速变大则 $E(u,v)$ 同样可以快速变大），我们可以想象出，图像在 x 方向和 y 方向整体变化都很大，因此这就是个角点。

当特征值 λ_1 和特征值 λ_2 一个很大一个很小时，则 I_x 或 I_y 只在某个方向能快速变到很大，我们可以想象出，图像仅在某个方向变化很大，因此这就是边缘区域。

当特征值 λ_1 和特征值 λ_2 都很小时，则该局部区域里 I_x 和 I_y 都很小，我们可以想象出，图像在各个方向变化都很小，因此这属于平坦区域，没有特征点。



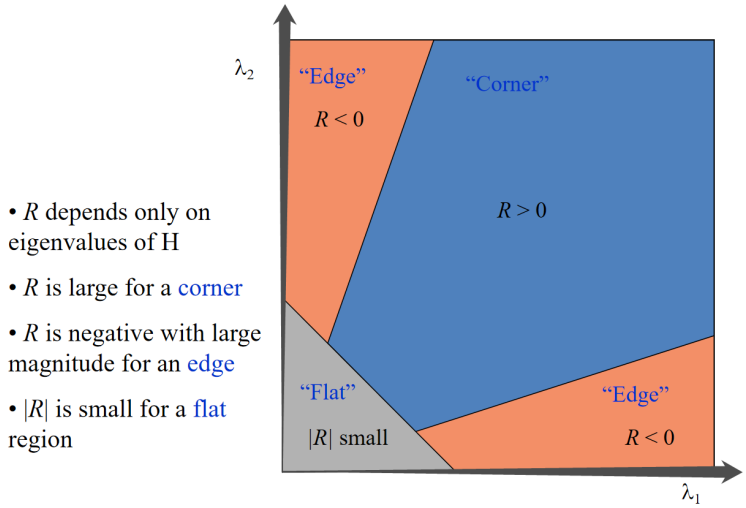
但毕竟使用两个特征值去判断角点并不是很方便，因为求特征值比较费时，且需要用两个量来比较判断也比较麻烦，因此定义了角点响应函数 R：

$$R = \lambda_1\lambda_2 - k(\lambda_1 + \lambda_2)^2 \tag{1.21}$$

$$= det(M) - k(trace(M))^2 \tag{1.22}$$

det 表示矩阵求行列式（2×2 矩阵求行列式很容易），*trace* 表示矩阵的迹（值等于对角线元素的和，同时也是特征值的和），当然我们现在不需要真正了解行列式和迹的实际意义和相关知识，毕竟上式是直接由特征向量定义得到的。k 一般取值为 $(0.04-0.06)$。

根据上式我们就能得到角点与 R 之间的关系：



以上内容就是 Harris 角点检测的全部内容了。我们检测出来以后可以对所有角点的 R 响应值进行排序，得到 R 值最大的 n 个角点作为特征点；以及对一个局部区域取 R 最大值点作为角点。

2. OpenCV 代码描述

本章介绍实际代码的实现过程，首先介绍一下实现上的细节，然后介绍 *OpenCV3* 中的实现方法。

2.1 实现细节

我们在上一章定义每个像素处的图像梯度为：

$$\begin{bmatrix} I_x & I_y \end{bmatrix} \tag{2.1}$$

我们还并没有介绍如何计算图像梯度。在实际运算中，图像梯度的计算经常会使用 Sobel 算子对一个局部区域 3×3 进行卷积：

$$Gradient_x = I * Sobel_x \tag{2.2}$$

$$Gradient_y = I * Sobel_y \tag{2.3}$$

$Gradient_x$ 和 $Gradient_y$ 是卷积后的输出图像，$Sobel_x$ 和 $Sobel_y$ 分别是卷积核：

$$Sobel_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \tag{2.4}$$

$$Sobel_y = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} \tag{2.5}$$

得到的卷积图像，在 $Gradient_x$ 图像的像素 i 处即为 I_{x_i}，在 $Gradient_y$ 图像的像素 j 处即为 I_{y_i}。

之后再得到每个像素处的 I_x^2、I_y^2 以及 I_xI_y（为了重复使用可以分别生成三张图像，然后再对这些图像进行高斯卷积（因为有加权和））：

$$A = \begin{bmatrix} \sum_{(x,y)\in W} \omega(x,y) I_x^2 & \sum_{(x,y)\in W} \omega(x,y) I_x I_y \\ \sum_{(x,y)\in W} \omega(x,y) I_y I_x & \sum_{(x,y)\in W} \omega(x,y) I_y^2 \end{bmatrix} \tag{2.6}$$

最后根据它们计算 R 响应值就可以了。

2.2 程序代码

首先先放上我们的用户使用代码，下一节开始讲解源码。

```
#include<opencv2/opencv.hpp>
using namespace cv;
int main()
{
    Mat src, gray;
    src = imread("p0-1.jpg");
    imshow("input", src);
    cvtColor(src, gray, CV_BGR2GRAY);
    Mat dst, normdst;
    dst = Mat::zeros(gray.size(), CV_32FC1);
    //角点检测核心代码
    cornerHarris(gray, dst, 2, 3, 0.04, BORDER_DEFAULT);
    //将dst内的值归一化到(0-255)之间（浮点数）
    normalize(dst, dst, 0, 255, NORM_MINMAX, CV_32FC1, Mat());
    //上面得到的浮点数缩放为8位uchar类型
    convertScaleAbs(dst, dst);
    Mat result = src.clone();
    //角点检测结果
    for (int r = 0; r < result.rows; r++){
        uchar* curRow = dst.ptr(r);
        for (int c = 0; c < result.cols; c++){
            if ((int)*curRow > 150) //画圈标注出结果
                circle(result, Point(c, r), 10, Scalar(0,255,0), 2,
                    1, 0);
            curRow++;
        }
    }
    imshow("output", result);
    waitKey(0);
    return 0;
}
```

注意因为进行了归一化，所以当某些特征响应非常大的时候，可能就只会显示出这几个特征。我们主要关心的是 cornerHarris 代码，下一节我们就重点讲解该代码的实现细节。

2.3 OpenCV 源码解读

在这里我使用的是 Opencv3.4.2 版本，函数声明在 imgproc.h 文件里，定义在 corner.cpp 文件里。

```
CV_EXPORTS_W void cornerHarris( InputArray src, OutputArray dst, int
    blockSize, int ksize, double k, int borderType = BORDER_DEFAULT
    );
```

其中，block_size 表示局部区域的大小，aperture_size 表示 Sobel 孔径的大小，一般设置为 3，k 表示前面所说的 R 响应的 k 值。borderType 表示滤波的时候边缘怎么处理，因为滤波核是有大小的，对于边缘像素的滤波需要考虑如何补充像素之类的问题。

cornerHarris 函数会首先做一些判断，判断参数值是否合理，然后调用 cornerEigenValsVecs 函数。该函数是根据特征值来寻找角点的核心函数。该函数的结构非常简单，就是先使用 Sobel 求梯度，然后再求 R 响应，我们一步一步来解析它。

求图像梯度

首先会根据 aperture_size 和 block_size 来计算 scale 值，进而判断是使用 Sobel 算子还是 Scharr 算子进行边缘检测。这里我只简单讲一下 Sobel 就可以了，毕竟都很简单。

cv::Sobel 只支持对单通道图像计算。该函数调用 getDerivKernels 得到对 x 方向和对 y 方向的滤波核，该函数虽然也是很多行，但其实生成的滤波核 3×3 与本章第一节描述的是一样的。

求自相关系数

我们现在要求 I_x^2、I_y^2 以及 I_xI_y 三个自相关系数，存到该 Mat 里：

```
Mat cov( size, CV_32FC3 );
```

过程除去使用加速指令的 haveAvx 和 haveSimd，其他部分都其实就是将 Dx 和 Dy 图像对应元素相乘：

```
for( i = 0; i < size.height; i++ ) {
    j = 0;
    float* cov_data = cov.ptr<float>(i);
    const float* dxdata = Dx.ptr<float>(i);
    const float* dydata = Dy.ptr<float>(i);
    for( ; j < size.width; j++ ) {
        float dx = dxdata[j];
        float dy = dydata[j];
        cov_data[j*3] = dx*dx;
        cov_data[j*3+1] = dx*dy;
        cov_data[j*3+2] = dy*dy;
    }
```

```
13    }
```

滤波（求加权系数）和响应检测

OpenCV3 使用的是盒滤波器：boxFilter，就是把一个区域内的值都取平均。

然后调用 calcHarris 函数。该函数前面仍然做了一堆判断用于 Debug 和进一步处理，我们其实不需要管是什么，我们只需要注意底下真正有意义的几行代码：

```
1   for( ; j < size.width; j++ ) {
2       float a = cov[j*3];
3       float b = cov[j*3+1];
4       float c = cov[j*3+2];
5       dst[j] = (float)(a*c - b*b - k*(a + c)*(a + c));
6   }
```

求得 R 响应值，存放到输出图像中。

以上就是 Harris 的 OpenCV 源码的实现了。我们做一个简单测试，使用图像处理女郎，在旁边我还点了几个小黑点（模拟噪声），对于 Harris 算法来说，噪声容易被当成特征：

block_size 调大以后就不会把这种单一的小黑点检测出来了，因为使用了 boxFilter，局部区域很大时，单一的小黑点一平均之后图像梯度就变得非常小了，但是比较大的特征点就不会被滤除：

因为这几个特征点的 R 响应值实在是太大了，总体进行归一化以后，导致上面检测出来的特征点的 R 响应都小于我们设定的 190 这个阈值，因此都没有显示出来。

Bibliography

[1] https://blog.csdn.net/linqianbi/article/details/78930239 （注意该博客存在不少知识性错误和理解性错误，给出的代码也有错误，不建议新手阅读。但本文的讲解顺序参考了该博客的内容。）

[2] https://zhuanlan.zhihu.com/p/67770305 （本文忽略了特征值和角点相关的介绍，但可以参考算法流程）

[3] https://blog.csdn.net/lwzkiller/article/details/54633670 （本文也有一定的知识性疏漏，但包含了作者的一些不充分理解，可以适当参考）