浙江大学

课程名称：计算机视觉

姓　　名：刘佳润

学　　院：计算机科学与技术学院

专　　业：数字媒体技术

学　　号：3180105640

指导教师：潘纲

2020　年　12　月　12　日

浙江大学实验报告

课程名称：计算机视觉 实验类型：综合

实验项目名称：Harris 角点特征检测

学生姓名：刘佳润 专业：数字媒体技术 学号：3180105640

同组学生姓名： 指导老师：潘纲

实验地点： 实验日期：2020 年 12 月 12 日

一、 实验内容和要求

读入摄像头，可以实时回放视频。键盘交互，空格键暂停，并对当前帧图像做一次 Harris Corner 检测，并将检测的结果叠加在原图上。

- Harris Corner 算法自己实现。
- 显示中间结果与最终结果，并保存：最大特征值图，最小特征值图，R 图，彩色 R 图，原图上叠加结果等。

二、 实验器材

C++ OpenCV 4.5.0

开发平台：Visual Studio 2019 Debug x64

三、 具体实现

1. 摄像头读取

摄像头读取利用 OpenCV 的 VideoCapture 类，初始化一个默认对象 capture(0)即可调用系统摄像头。设置 delay 为 30ms，用 waitkey()函数来判断键盘交互。

```
//读取摄像头
VideoCapture capture(0);
int delay = 30;
while (true)
{
    Mat frame;
    capture >> frame;
    int key = waitKey(delay);
    imshow("Camera", frame);
    if (delay >= 0 && key == 32)
    {
        cout << "Harris Corner Detecting..." << endl;
        HarrisCorner(frame);
        cout << "Done!" << endl;
        waitKey(0);
    }
    if (key == 27)
        break;

}
return 0;
```

2. Harris Corner 检测算法实现

Harris Corner 算法在一个窗内计算期望 E(x,y)。对于期望大于某个阈值的点，可以被视作角点。公式如下：

$$E(u,v)=\sum_{x,y} w(x,y)\left[I(x+u,y+v)-I(x,y)\right]^2$$

其中选择 w(x,y)窗函数为高斯函数。对于很小的 u、v 移动，可以近似将这个期望写作线性方程：

$$E(u,v)\cong\begin{bmatrix}u,v\end{bmatrix}\quad M\quad\begin{bmatrix}u\\v\end{bmatrix}$$

- 梯度计算（Ix、Iy）

梯度计算利用 Sobel 算子实现，上次实验中自己写边缘检测的时候已经用过了一次，稍加改写即可。

需要注意掩膜与原图像之间的关系，先对原图像进行一次行和列的扩展，尽量减少在 Sobel 滤波过程中的损失。扩展过程中保存原图像在中间部分，赋值上下左右最边缘的像素到外围。

```cpp
// 按图扩展，便于卷积
void expand(const Mat& raw, Mat& img_expand)
{
    Mat tmp = img_expand(Range(1, raw.size().height + 1), Range(1, raw.size().width + 1));
    raw.copyTo(tmp);

    tmp = img_expand(Range(0, 1), Range(1, raw.size().width + 1));
    raw.row(0).copyTo(tmp);
    tmp = img_expand(Range(raw.size().height + 1, raw.size().height + 2), Range(1, raw.size().width + 1));
    raw.row(raw.size().height - 1).copyTo(tmp);
    tmp = img_expand(Range(1, raw.size().height + 1), Range(0, 1));
    raw.col(0).copyTo(tmp);
    tmp = img_expand(Range(1, raw.size().height + 1), Range(raw.size().width + 1, raw.size().width + 2));
    raw.col(raw.size().width - 1).copyTo(tmp);

    cvtColor(img_expand, img_expand, COLOR_BGR2GRAY);
}
```

对扩展图像分别用 SobelX 和 SobleY 算子卷积，得到 Ix 和 Iy，即原图在 x 和 y 方向上的梯度。

```cpp
// 计算梯度(I)
void grad(Mat& Ix, Mat& Iy, Mat& img_expand)
{
    float acc_dx = 0, acc_dy = 0;          //accumulators
    float k1[] = { -1,0,1,-2,0,2,-1,0,1 };   //sobel kernal dx
    float k2[] = { -1,-2,-1,0,0,0,1,2,1 };    //sobel kernal dy

    for (int i = 0; i < Ix.rows; i++) {
        for (int j = 0; j < Ix.cols; j++) {
            acc_dx = acc_dy = 0;

            //apply kernel/mask
            for (int nn = 0; nn < 3; nn++) {
                for (int mm = 0; mm < 3; mm++) {

                    acc_dx += img_expand.at<float>(i + nn, j + mm) * k1[(mm * 3) + nn];
                    acc_dy += img_expand.at<float>(i + nn, j + mm) * k2[(mm * 3) + nn];

                }
            }
            //write final values
            Ix.at<float>(i, j) = acc_dx;
            Iy.at<float>(i, j) = acc_dy;
        }
    }

}
```

- 定义像素点矩阵 M

M 的定义如下：

$$M = \sum_{x,y} w(x,y) \begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix}$$

根据公式计算即可。

```
// 计算M矩阵
void getM(Mat& Ix, Mat& Iy, Mat& M, int row, int col, int windowSize)
{
    for (int i = row - windowSize / 2; i < row + windowSize / 2 + 1; ++i)
    {
        for (int j = col - windowSize / 2; j < col + windowSize / 2 + 1; ++j)
        {
            M.at<float>(0, 0) += Ix.at<float>(i, j) * Ix.at<float>(i, j);
            M.at<float>(1, 0) += Ix.at<float>(i, j) * Iy.at<float>(i, j);
            M.at<float>(0, 1) += Ix.at<float>(i, j) * Iy.at<float>(i, j);
            M.at<float>(1, 1) += Iy.at<float>(i, j) * Iy.at<float>(i, j);
        }
    }
}
```

- 计算每个像素的角点响应，即 R 值

响应值 R 的公式如下：

$$R = \det M - k\left(\operatorname{trace} M\right)^2$$

其中 k 是经验值，通常取 0.04-0.06，矩阵 M 的 det（行列式）与 trace（迹）可以由矩阵 M 的特征向量得出：

$$\det M = \lambda_1 \lambda_2$$
$$\operatorname{trace} M = \lambda_1 + \lambda_2$$

OpenCV 的自带函数可以很快求出矩阵 M 的 det 和 trace，然后反求出两个特征值。根据算法原理，当两个特征值都比较大的时候，属于 corner。

```
// 计算特征向量
void getLambda(Mat& M, float& lambda1, float& lambda2)
{
    float a = 1.0f, b = -trace(M).val[0], c = determinant(M);
    // lambda1 > lambda2
    lambda1 = 0.5f / a * (-b + sqrt(b * b - 4 * a * c));
    lambda2 = 0.5f / a * (-b - sqrt(b * b - 4 * a * c));
}
```

构建两个特征值图 Max_lambda 和 Min_lambda，用相应的 λ 来绘制。

```
for (int i = windowSize / 2, k = 0; k < R_Size.height; ++i, ++k)
{
    for (int j = windowSize / 2, l = 0; l < R_Size.width; ++j, ++l)
    {
        Mat M(2, 2, CV_32F, cv::Scalar(0.0));
        getM(Ix, Iy, M, i, j, windowSize);
        getLambda(M, lambda1, lambda2);
        Max_lambda.at<float>(k, l) = sqrt(sqrt(lambda1));
        Min_lambda.at<float>(k, l) = sqrt(sqrt(lambda2));

        R.at<float>(k, l) = sqrt(sqrt(lambda1 * lambda2 - K * (lambda1 + lambda2) * (lambda1 + lambda2)));
    }
}
```

- 非极大值抑制与可视化

构建 R 图后，在一个限定的窗口内，只取最大且大于阈值的点。设置

windowSize 为 30，对大于阈值且大于记录的域内最大值的点进行记录，否则就重置为 0。由于 windowSize 的大小，需要特别做边缘处理，否则会在最右一列和最下一行以及右下角出现大量的检测点。

```
double maxVal = 0;
int windowSize = 30;
for (int i = 0; i < R.rows - windowSize; i += windowSize)
{
    for (int j = 0; j < R.cols - windowSize; j += windowSize)
    {
        maxVal = 0;
        for (int m = i; m < i + windowSize; m++)
        {
            for (int n = j; n < j + windowSize; n++)
            {
                if (R.at<float>(m, n) > maxVal && R.at<float>(m, n) > threshold)
                {
                    maxVal = R.at<float>(m, n);
                }
                else {
                    R.at<float>(m, n) = 0;
                }
            }
        }
    }
}
```

最后遍历 R 图，根据 R 图的检测结果来显示角点。

四、 实验结果与分析

使用课件中的样图进行测试，经调试与修改结果基本符合预期。

实时调用摄像头，可以对人脸与环境的特征点进行检测。

效果基本符合预期。中间输出的最大最小特征值图与 R 图储存在 result 文件夹里。可以看出最大特征值图基本勾勒出图像的边缘，最小特征值图已经能够对

corner 点有基本的描绘。

五、 心得与体会

这次实验总体比较顺利，顺便对于 Release 格式下的发布做了一些实验。对于 R 图的绘制和彩色 R 图的绘制还有一些问题，现在的彩色 R 图比较难看，不是很了解课件中的彩色图是如何调整通道值画出来的。

下来也会考虑 Harris-Laplace，可以更加灵活地设置 kernel 的大小并对各种形态和大小的图片做检测。