**EE382-HW2**

**郭远帆 516021910700**

1. **摘要**

Harris 角点检测是一种经典的角点检测方法。 角点检测的基本思想为寻找在横纵坐标方向变化量均较大的点，而Harris算法使用泰勒展开的方法简化了计算，是一种低成本的实用角点检测算法。 本报告阐述了Harris角点检测算法的原理以及在实现过程中的一些选择，并给出了相应的实验数据和结果。

1. **Harris算法原理**

对于图像，其x,y 方向上的偏导数为 。在某点处，计算在微小平移下的梯度变化，有:

使用一阶泰勒展开近似，有:

表示成矩阵形式，即:

]

采用矩形窗口进行加窗，得到自相关矩阵M:

可以根据M的两个特征值来判断是否为角点，当与都较大且相近时，说明在x,y方向灰度均有大幅度变化，可以将该点判断为角点。在Harris算法中，使用R值来表示该点的Corner Response:

其中det(M)为M的行列式，trace(M)为M的迹，k是一个常数，通常选值为0.04~0.06

人为设定判断阈值, R>Thres的点可以视为角点（一般可以设R最大值的0.01）

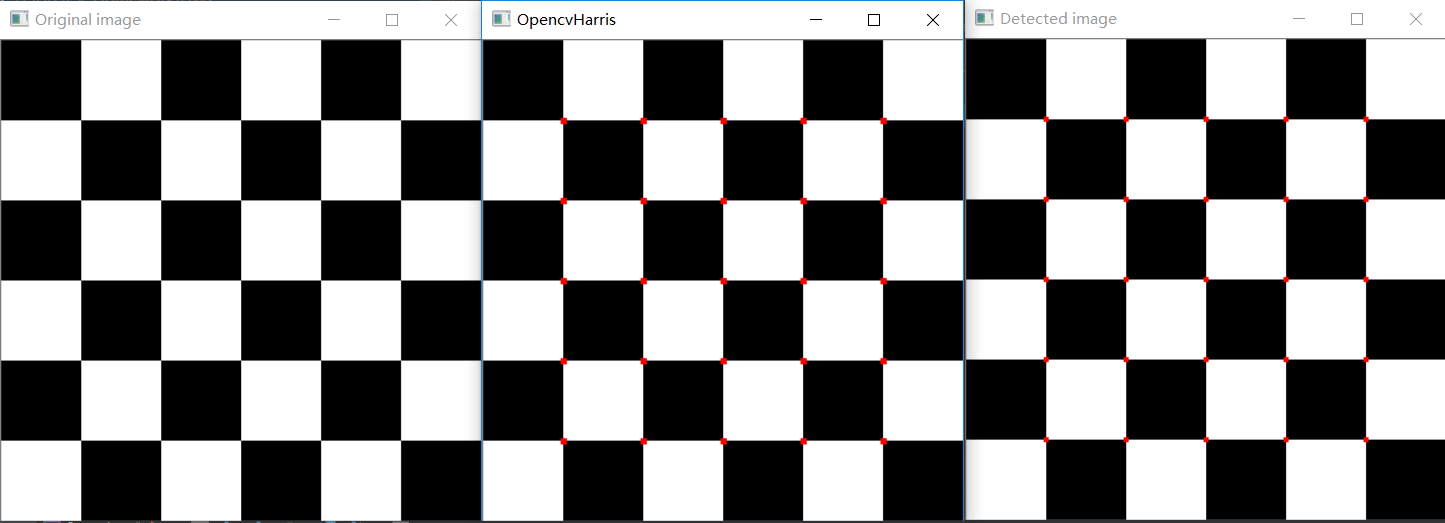
除此之外，为了防止采点过密，需要对角点进行非极大值抑制。通常选用3\*3或者5\*5的窗口，对于窗口中的角点只取R值最大的像素点，其余不记为角点。

本次实验使用Python语言编写Harris角点检测算法程序，并使用了面向对象的编程方法。主要文件为Harris类文件 HarrisDetector.py以及测试代码 Test.py

1. **实验结果与分析**

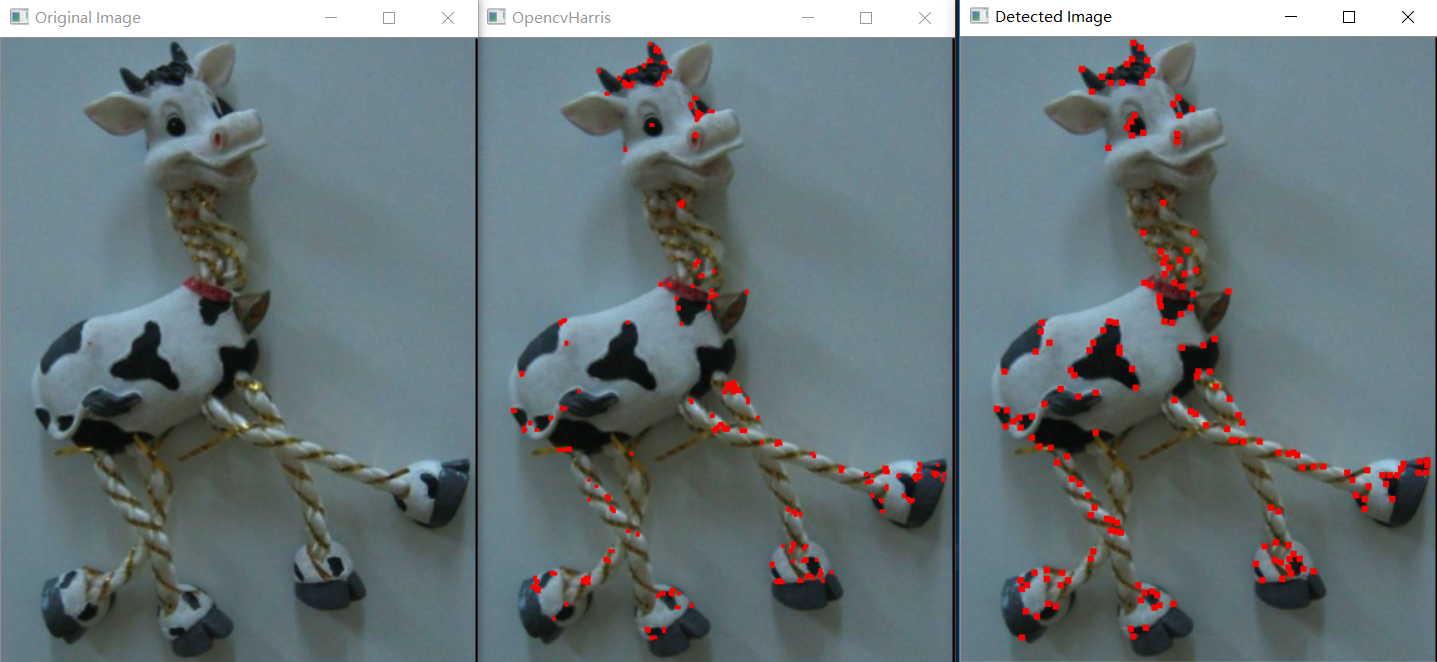
以下为测试图片极其结果，我们将自己设计的Harris角点检测与Opencv自带的Harris角点检测算法进行对比，发现还存在一定差距:

第一组实验结果:



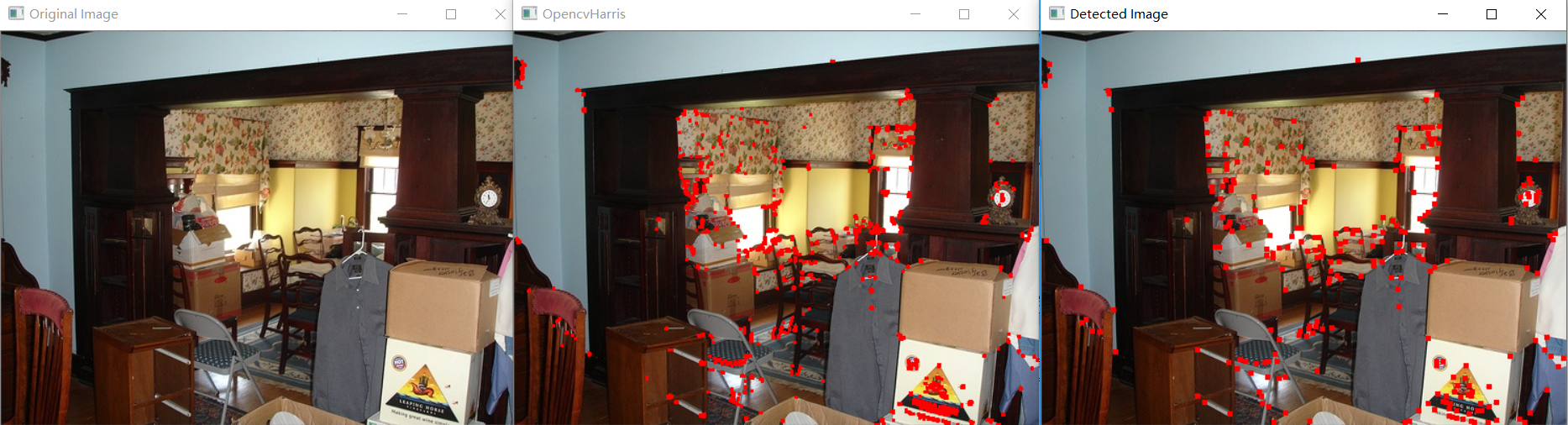
第一组图片较为简单，因此自己设计的算法与opencv内置算法得到的结果一样。所有的角点均成功被检测出来

第二组实验结果:



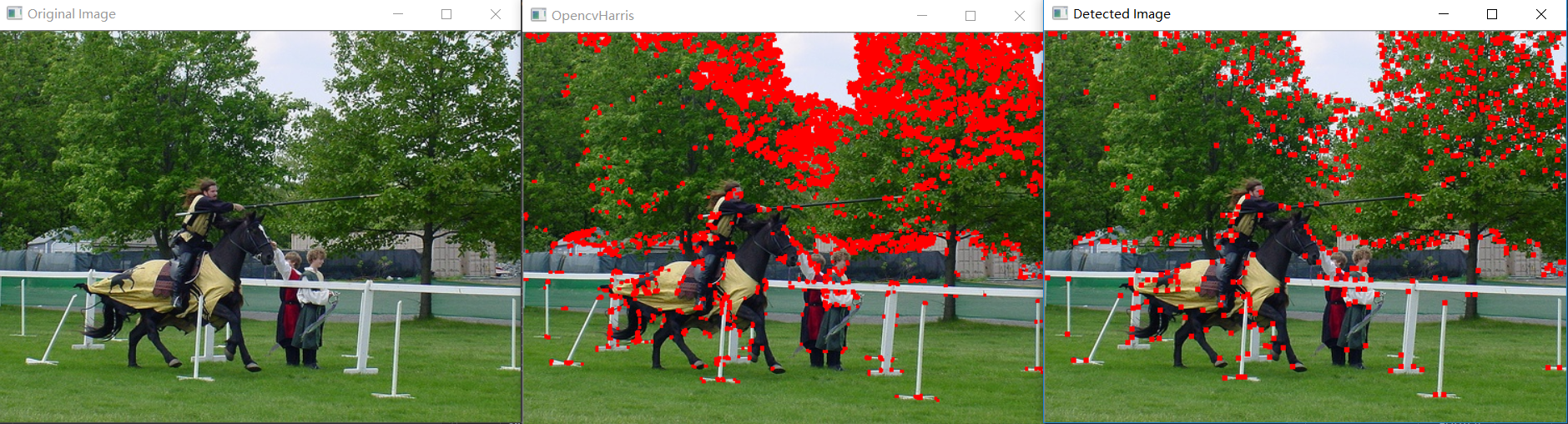
第二组图片为课件中使用的测试图片，可以看到自己设计的算法给出了更多的角点，可以调大阈值以达到更好的效果

第三组实验效果:



第三组图片比较复杂，但可以看到自己设计的算法与opencv内置得到的结果十分类似。

第四组实验结果:



我们发现第四组图片上自己设计的算法检测的角点更加准确（与阈值的调整有关）

1. **总结**

本文阐述了Harris算法的原理，并使用Python编程将其实现，给出了与Opencv内置的Harris算法的对比。在实验过程中，也遇到了不少困难与疑惑。在以下列出:

1. 一开始设计算法时，发现算得的R值特别大，最后发现与图片的img格式有关。在转换成灰度图像后，需要将其转换为有符号类型数组。最终选择了转换为numpy数组，数据类型为float32

**附录**

**HarrisDetector.py**

# HW2 Harris Detector implement

# HarrisDetector class

# Author : Yuanfan Guo

# Rev.0 2018.10.8

# Rev.1 2018.10.9

import cv2

import numpy as np

import matplotlib.pyplot as plt

class HarrisDetector():

def \_\_init\_\_(self,ThresHold = 0.02, WindowSize = 5 , K=0.05, Blocksize = 5):

self.ThresHold = ThresHold #Above threshold will count as corner

self.K = K #K is Empirical Constant, 0.04~0.06

self.WindowSize = WindowSize #The size of the window

self.Blocksize = Blocksize #The Blocksize (usually 3 or 5)

def SetPara(self,ThresHold,WindowSize,K,Blocksize):

#Set ThresHold, WindowSize and K and Blocksize of detector

self.ThresHold = ThresHold

self.K = K

self.WindowSize = WindowSize

self.Blocksize = Blocksize

def DetectCorner(self,InImg):

#InImg : image matrix (gray or color)

#Detect corners in an img and return Corner list and new image

Result = InImg.copy()

gray\_img = cv2.cvtColor(InImg, cv2.COLOR\_BGR2GRAY)

img = np.array(gray\_img,dtype=np.float64)

CornerResponse = np.zeros([img.shape[0],img.shape[1]])

#Calculate Ixx Ixy and Iyy

Ixx,Ixy,Iyy = self.CalGradient(img)

Half\_w = int(self.WindowSize/2)

for x in range(Half\_w, img.shape[0]-Half\_w):

for y in range(Half\_w , img.shape[1]-Half\_w):

##Compute sums of derivatives

M11 = sum(Ixx[x - Half\_w: x + Half\_w + 1, y - Half\_w: y + Half\_w + 1].ravel())

M12 = sum(Ixy[x - Half\_w: x + Half\_w + 1, y - Half\_w: y + Half\_w + 1].ravel())

M22 = sum(Iyy[x - Half\_w: x + Half\_w + 1, y - Half\_w: y + Half\_w + 1].ravel())

##Get Eigen Value and R

Det = M11\*M22 - M12\*\*2

Trace = M11 + M22

R = Det - self.K\*(Trace\*\*2)

CornerResponse[x][y] = R

#Find Local max Value and plot corner

Cornerlist = []

CornerMax = max(CornerResponse.ravel())

HalfBlcok = int(self.Blocksize/2)

for x in range(HalfBlcok, img.shape[0]-HalfBlcok):

for y in range(HalfBlcok , img.shape[1]-HalfBlcok):

LocalMax = max(CornerResponse[x - HalfBlcok:x+HalfBlcok+1,y-HalfBlcok:y+HalfBlcok+1].ravel())

#only the Localmax value count as corner

if (CornerResponse[x, y] >= CornerMax \* self.ThresHold) and (CornerResponse[x, y] == LocalMax):

Cornerlist.append((x, y))

#Plot the Corner point

Result[x - HalfBlcok:x+HalfBlcok+1,y-HalfBlcok:y+HalfBlcok+1] = [0,0,255]

return Result, Cornerlist

def CalGradient(self,img):

dy, dx = np.gradient(img)

Ixx = dx\*\*2

Iyy = dy\*\*2

Ixy = dx\*dy

return Ixx,Ixy,Iyy