图像与视频接口

2023/3/11

电子系统导论教学团队

实验目的

- 了解数字图像的基本概念和空-频分析
- 掌握图像的基本处理方法

USB摄像头

- 树莓派兼容USB摄像头列表：https://elinux.org/RPi_USB_Webcams
- Logitech Webcam C270

硬件连接

- 摄像头通过USB线与树莓派相连

确认正确连接和识别

- $ ls /dev

  是否有(一般名为)video0一项

- $ lsusb

  是否识别出摄像头信息

```
pi@raspberrypi:~ $ lsusb
Bus 001 Device 005: ID 046d:082b Logitech, Inc. Webcam C170
Bus 001 Device 007: ID 046d:c077 Logitech, Inc. M105 Optical Mouse
Bus 001 Device 006: ID 046d:c31d Logitech, Inc. Media Keyboard K200
Bus 001 Device 004: ID 1a40:0101 Terminus Technology Inc. Hub
Bus 001 Device 003: ID 0424:ec00 Standard Microsystems Corp. SMSC9512/9514 Fast
Ethernet Adapter
Bus 001 Device 002: ID 0424:9514 Standard Microsystems Corp. SMC9514 Hub
Bus 001 Device 001: ID 1d6b:0002 Linux Foundation 2.0 root hub
pi@raspberrypi:~ $
```

V4L驱动

- 全称：Video4Linux（Video for Linux）
- 是Linux内核中关于视频设备的子系统，为linux下的视频驱动提供了统一的接口，使得应用程序可以使用统一的API操作不同的视频设备，极大地简化了视频系统的开发和维护。
- 早期V4L有许多缺陷，Bill Dirks等人对其进行重新设计，取名为Video for Linux 2(V4L2)，最早出现于Linux2.5.x。应用程序V4L编程实际多指V4L2。

拍照--fswebcam

- 安装：$ sudo apt-get install fswebcam

- 拍摄一张照片：
  $ fswebcam [<options>] <filename>
  例：$ fswebcam --no-banner -S 10 -r 640*480 image.jpg
  --no-banner：禁用图片下方信息横幅
  -S：skip，跳过的帧数（实测第一次调用fswebcam必须跳过一些帧才能得到有效图像）
  -r：resolution，分辨率
  更多可选参数通过 $ man fswebcam 命令查看

- 终端默认路径是 /home/pi/　　保存的图片在那里。

摄像头获取实时视频

- 安装：$ sudo apt-get install mplayer
- 播放摄像头实时拍摄内容：
  $ sudo mplayer tv://
- 播放视频文件：
  $ mplayer <filename>

安装OpenCV

- 方法一：终端命令安装

```
$ sudo apt-get update
$ sudo apt-get install python3-opencv
```

- 方法二：左上角树莓派图标 →“Preferences”→“Add / Remove Software”进入软件中心，搜索“opencv”，勾选“Python bindings for the computer vision library”，点击“Apply”，即可完成安装

- 检查是否安装成功并查看当前版本：

```
$ python3
>> import cv2
>> cv2.__version__
```

OpenCV: 拍照、显示、保存图片

```
import cv2

cap = cv2.VideoCapture(0) # 获取摄像头句柄, 只连接一个摄像头时参数写0即可

while True:
    ret, frame = cap.read() # 读一帧
    cv2.imshow("display", frame) # 显示
    key = cv2.waitKey(1) & 0xFF # 检测键盘, 最长等待1ms (注意0表示永远而非0ms)
    if key == ord('p'): # ord(): 返回对应的 ASCII 数值，或者 Unicode 数值
        cv2.imwrite("image.jpg", frame) # 按p时保存图片
    if key == ord('q'):
        break # 按q时退出

cap.release() # 释放摄像头
cv2.destroyAllWindows() # 关闭所有显示窗体
```

注：图片保存在代码所在目录，按键不要在IDE的cmd窗口里输入，而要在外面焦点窗口输入。

OpenCV: 摄像、显示、保存视频

```
import cv2

cap = cv2.VideoCapture(0) # 获取摄像头句柄, 只连接一个摄像头时参数写0即可
out = cv2.VideoWriter("movie.avi", cv2.VideoWriter_fourcc('X', 'V', 'I', 'D'), 17, (640, 480)) # 打开/新建视频文件用于写入, 帧率=17(实测循环周期大约0.58s), 帧尺寸=640x480

while True:
    ret, frame = cap.read() # 读一帧
    cv2.imshow("frame", frame) # 显示
    out.write(frame) # 写入视频文件
    key = cv2.waitKey(1) & 0xFF # 检测键盘, 最长等待1ms
    if key == ord('q'):
        break # 按q时结束

cap.release() # 释放摄像头
out.release() # 关闭视频文件
cv2.destroyAllWindows() # 关闭所有显示窗体

# 视频保存在代码所在目录
```

回放保存的视频文件

- 安装：$ sudo apt-get install vlc

- 然后用vlc打开保存的movie.avi文件
- movie.avi保存在代码所在目录

重要函数解析

- 打开摄像头(或视频文件)：

  Python: `cv2.VideoCapture`(filename) → <VideoCapture object>

  Python: `cv2.VideoCapture`(device) → <VideoCapture object>

  filename：打开的视频文件的名称（例如video.avi）或图像序列（例如，img_00.jpg，img_01.jpg，img_02.jpg，...）
  device：打开的视频捕捉设备的ID（即相机索引）。 如果连接了一台摄像机，只需传递0即可。

- 读一帧：

  Python: `cv2.VideoCapture.read`([image]) → retval, image

- 释放摄像头(或视频文件)：

  Python: `cv2.VideoCapture.release`() → None ¶

重要函数解析

- 显示图像：

Python: `cv2.imshow(winname, mat) → None`

这个函数后面应该有一个waitKey函数，它显示图像指定的毫秒数。 否则，它将不会显示图像。 例如，waitKey（0）将无限显示窗口，直到任何按键（适合于图像显示）。 waitKey（25）将显示25 ms的帧，之后显示将自动关闭。 （如果你把它放在一个循环读取视频，它将逐帧显示视频）

- 写图片文件：

Python: `cv2.imwrite(filename, img[, params]) → retval` ¶

重要函数解析

- 打开视频文件：

  Python: cv2.**VideoWriter**([filename, fourcc, fps, frameSize[, isColor]]) → <VideoWriter object>

  filename：输出视频文件的名称。

  fourcc：4个字符的编解码器代码，用于压缩帧。 例如，CV_FOURCC（'P'，'I'，'M'，'1'）是MPEG-1编解码器，CV_FOURCC（'M'，'J'，'P'，'G' jpeg编解码器等。代码列表可以通过FOURCC页面在视频编解码器中获得。

  fps：创建的视频流的帧率。

  frameSize：视频帧的大小。

  isColor：如果不是零，则编码器将预期和编码彩色帧，否则将使用灰度帧（该标记当前仅在Windows上受支持。

- 写入一帧：

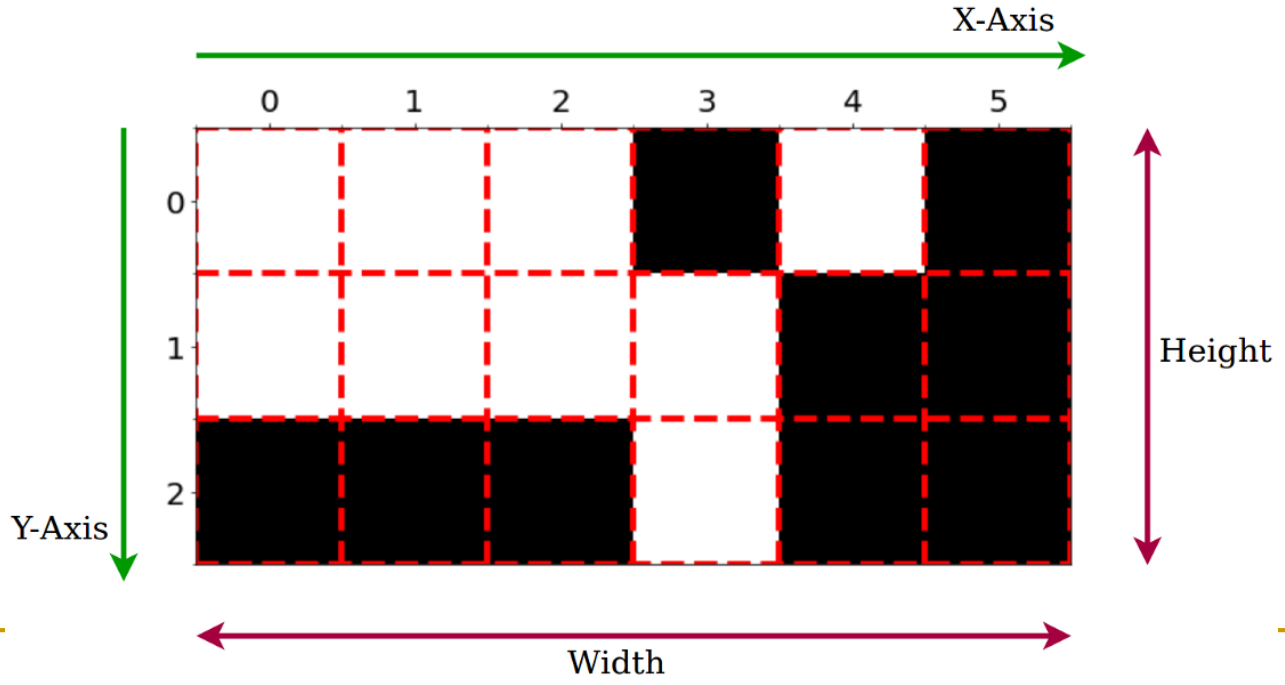
  Python: cv2.VideoWriter.**write**(image) → None

- 关闭视频文件

  Python: cv2.VideoWriter.**release**() → None

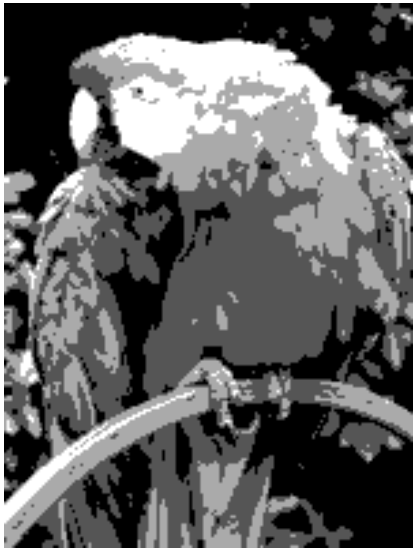
数字图像

- 像素：图像由水平垂直均匀分布的点（像素）构成
- 分辨率 指 水平垂直方向各有多少个像素
  - 例：1024x768
- 灰度图像：每个像素的值(0-255)表示黑-白之间的灰阶（亮度）
- 彩色图像：每个像素有RGB三个值(分别都为0-255)，混合产生不同的色彩。



灰度（GreyScale）

1-bit (二值图)



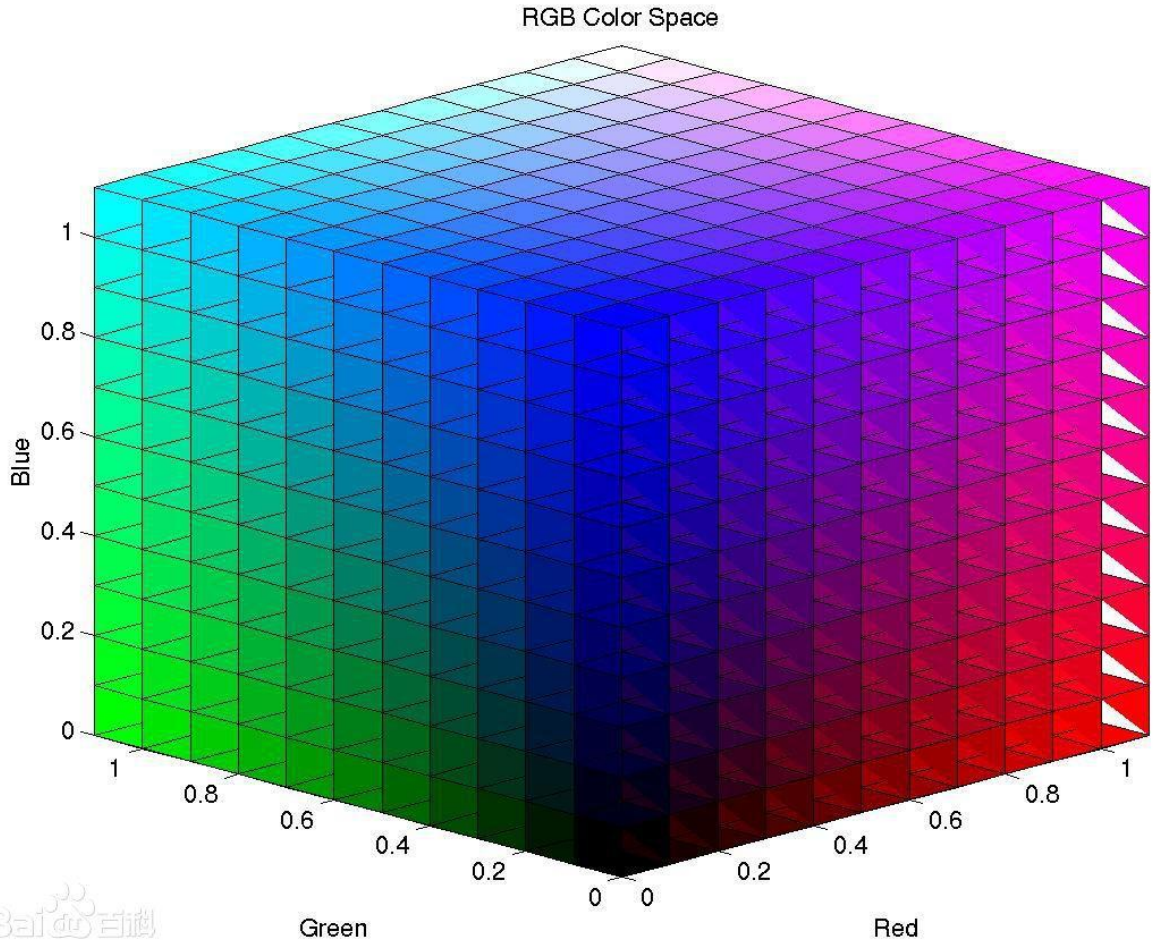
2-bit 灰度图

8-bit 灰度图

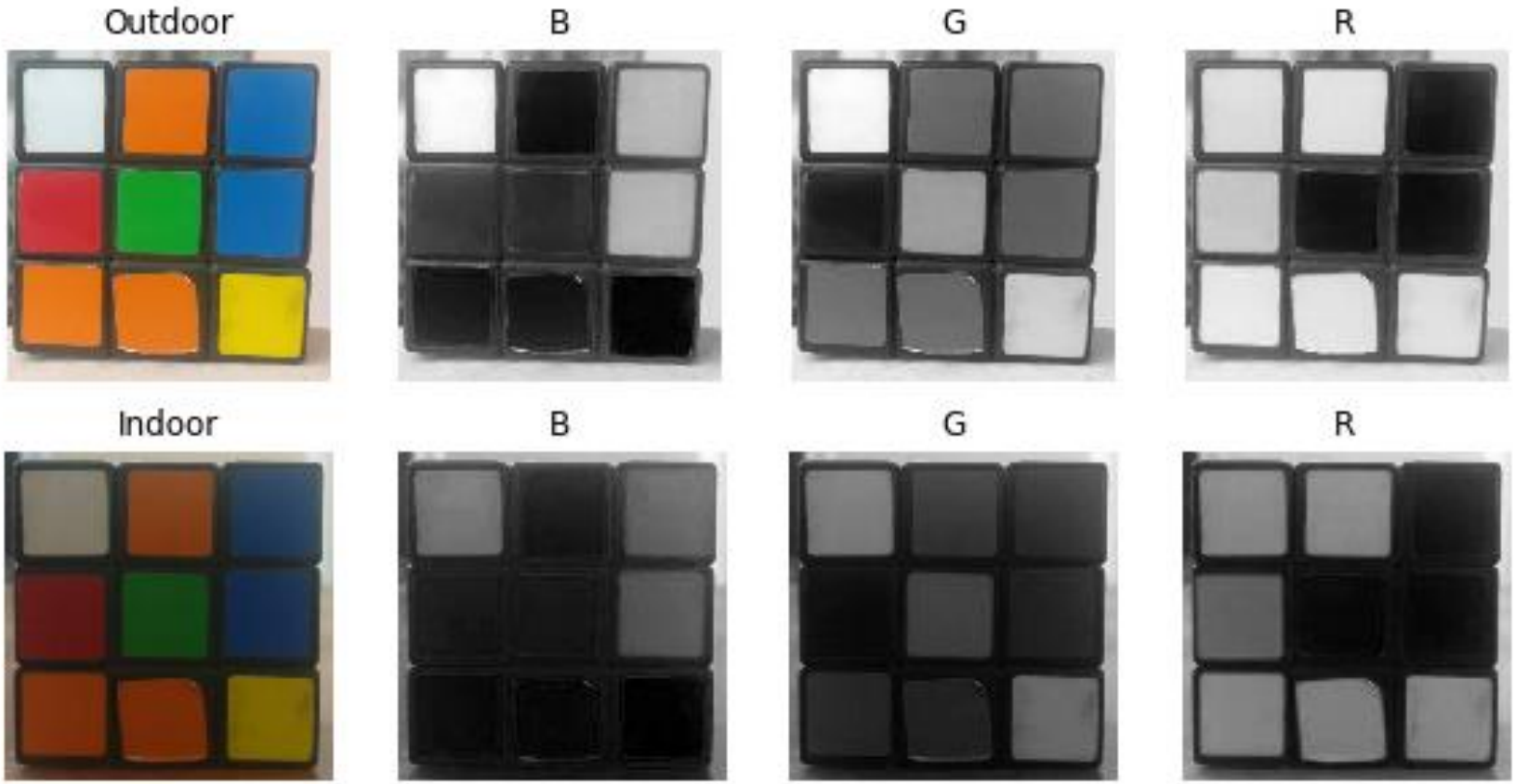
0x0	0x1	0x2	0x3	0x4	0x5	0x6	0x7	0x8	0x9	0xA	0xB	0xC	0xD	0xE	0xF

4-bit 灰度调色板

RGB色彩空间



RGB色彩空间

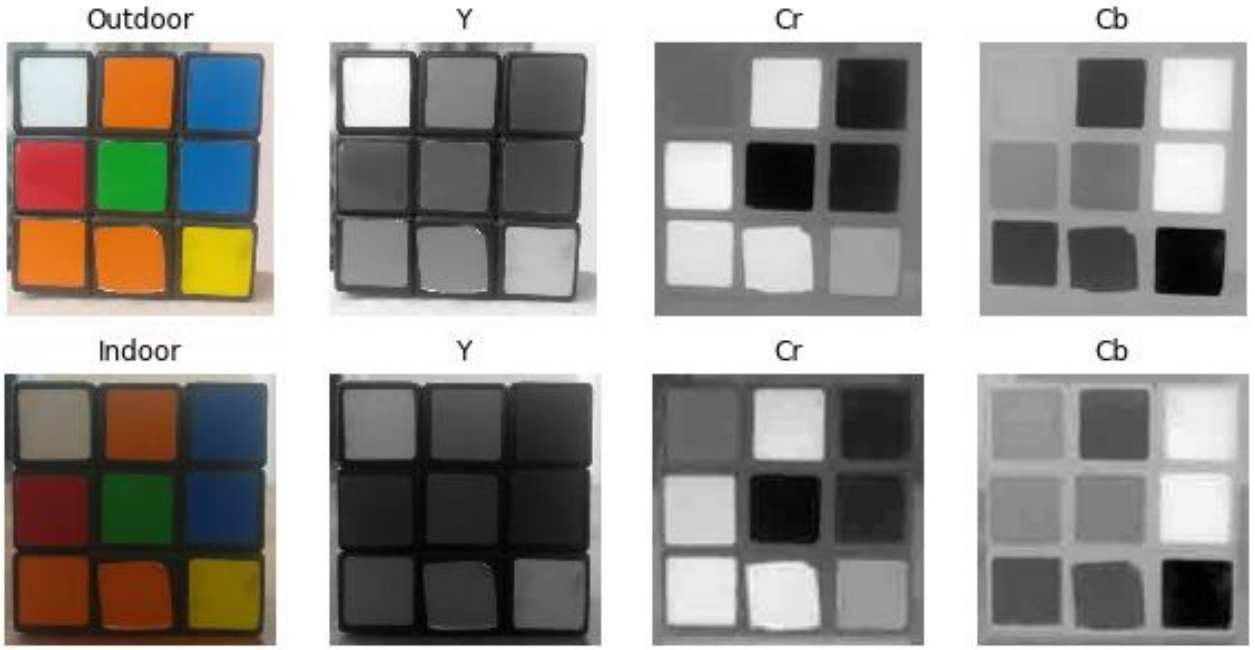


RGB分量均与整体亮度有关

YCrCb色彩空间

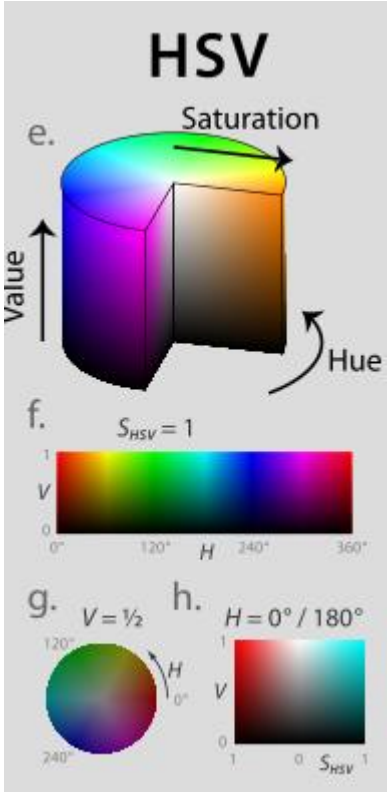
- Y: 亮度（由伽马校正后的RGB计算得到）
- Cr = R – Y
- Cb = B – Y

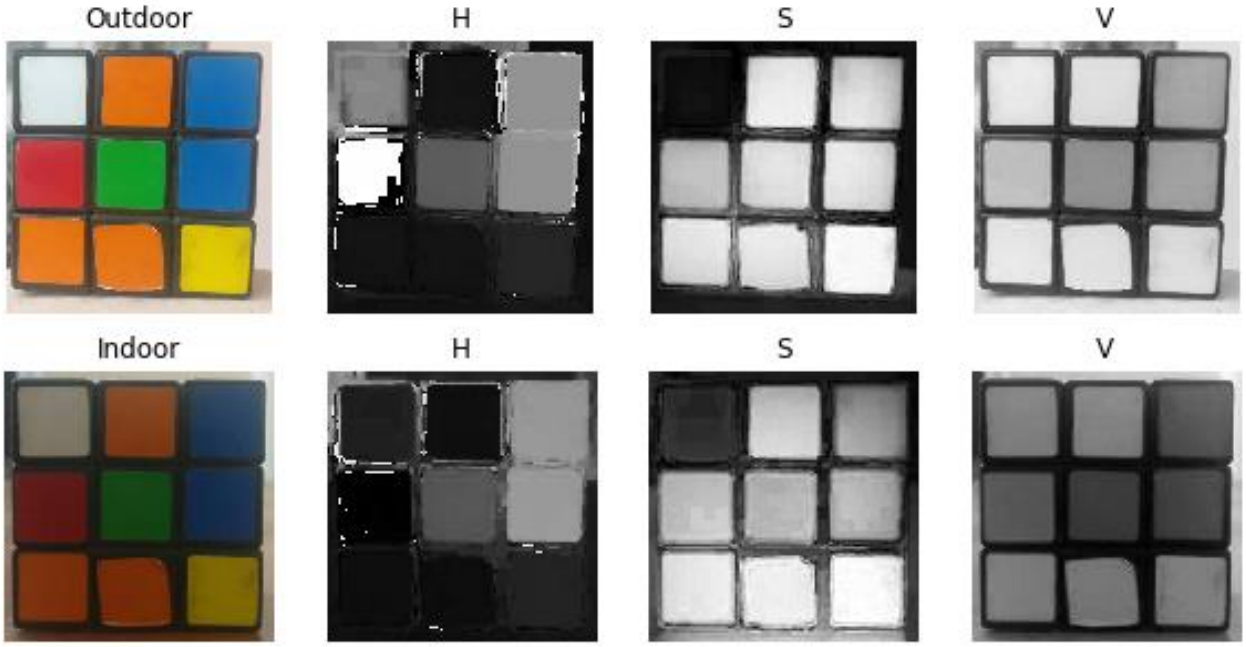
去除亮度影响后的红蓝分量



HSV色彩空间

- H -色相(**主波长，即颜色**)
- S -饱和度(纯度/色度)
- V –明度(强度)





Python-OpenCV图像处理:色彩空间变换

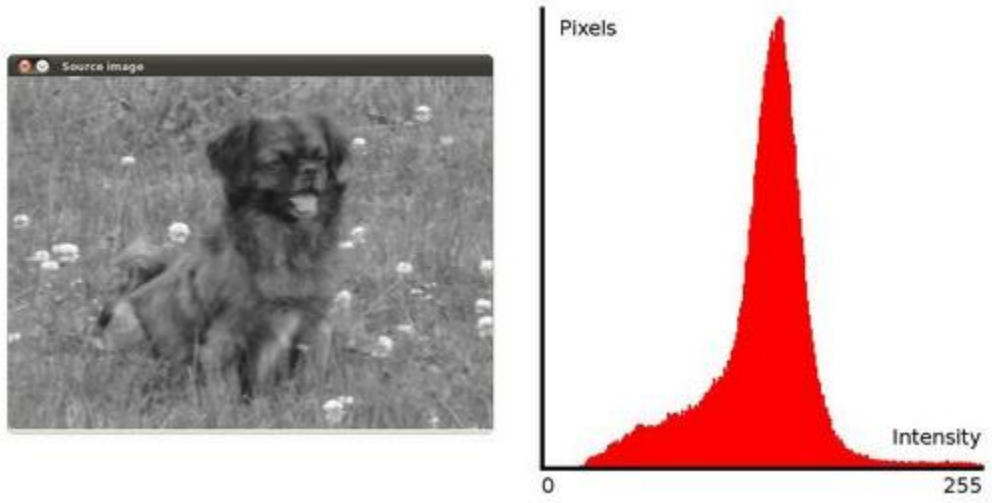
- cv2.cvtColor(image, code, dst, dstCn)
  - image: 要更改其色彩空间的图像。
  - code: 色彩空间代码。
  - dst: 与 src 图像大小和深度相同的输出图像，可选参数。
  - dstCn: 它是目标图像中的频道数。如果参数为 0，则通道数自动从 src 和代码得出，可选参数。

- 示例

```
imgYCB = cv2.cvtColor(img, cv2.COLOR_BGR2YCrCb)
imgHSV = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)
```

Python-OpenCV图像处理:直方图

- 直方图是图像中像素强度分布的图形表达方式.
- 它统计了每一个灰度强度值所具有的像素个数.

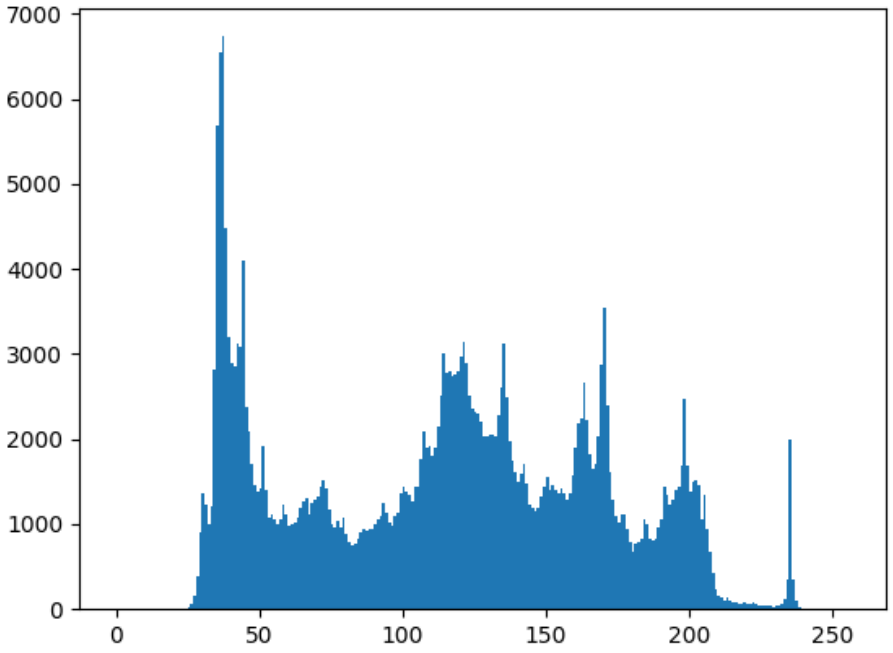


Python-OpenCV图像处理:统计直方图

- 使用Python-OpenCV统计直方图
- `cv2.calcHist(images,channels,mask,histSize,ranges[,hist[,accumulate]]`
- images：输入图像，传入时应该用中括号[]括起来
- channels：传入图像的通道，如果输入图像是灰度图像，值为[0]；如果是彩色图像，传入参数可以为[0]，[1]，[2]，分别对应B，G，R，使用时需用中括号[]
- mask：掩膜图像。如果统计整幅图，为none；如统计图像部分直方图，需构造相应的掩膜来计算
- histSize：灰度级BIN的个数，使用时需用中括号[]
- ranges：像素值范围，通常[0,256]
- hist：输出的 narray 类型，shape 是 256 x 1
- accumulate：是否积累

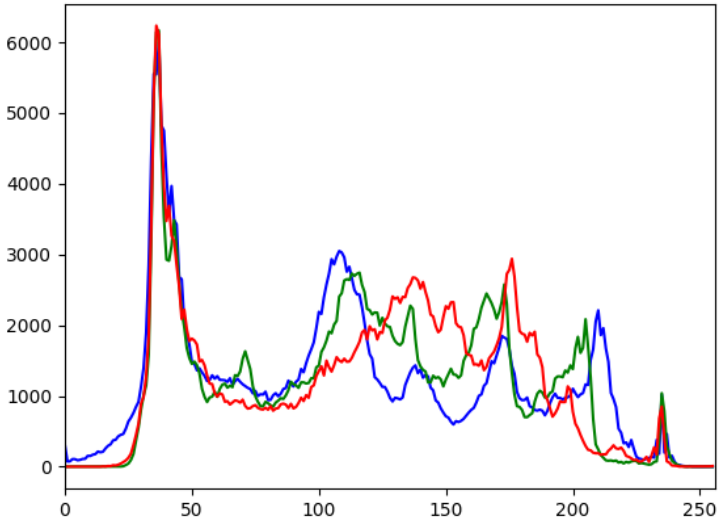
图像处理:Matplotlib绘制直方图

- 用python中的matplotlib库可以很容易的画出直方图：
- from matplotlib import pyplot as plt
- plt.hist(img.ravel(), 256, [0, 256]);
- plt.show()



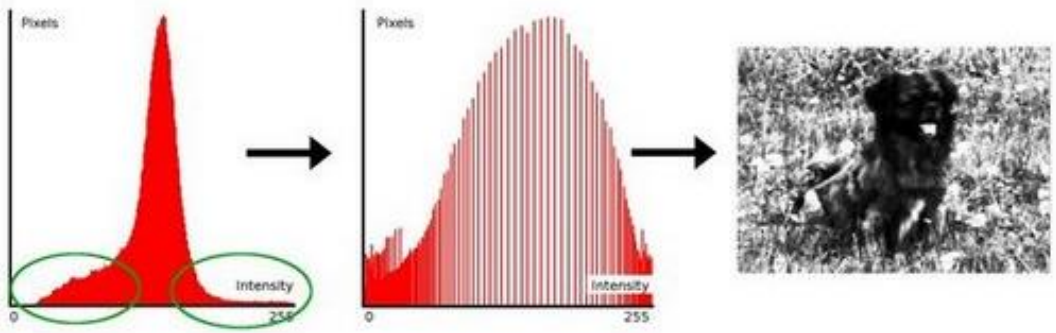
图像处理:Matplotlib绘制直方图

- 使用matplotlib库也可以绘制多通道（BGR）的直方图：
- img = cv2.imread('pic.jpeg')
- color = ('b','g','r')
- for i,col in enumerate(color):
- histr = cv2.calcHist([img],[i],None,[256],[0,256])
- plt.plot(histr,color = col)
- plt.xlim([0,256])
- plt.show()



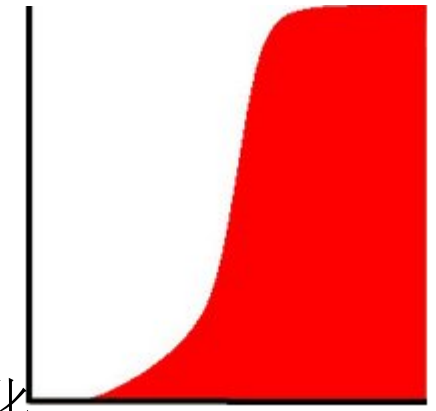
Python-OpenCV图像处理:直方图均衡化

- 直方图均衡化是通过拉伸像素强度分布范围来增强图像对比度的一种方法.
- 以上面的直方图为例, 你可以看到像素主要集中在中间的一些强度值上. 直方图均衡化要做的就是 拉伸 这个范围. 见下面左图: 绿圈圈出了少有像素分布其上的 强度值. 对其应用均衡化后, 得到了中间图所示的直方图. 均衡化的图像见下面右图.



Python-OpenCV图像处理:直方图均衡化

- 均衡化指的是把一个分布 (给定的直方图) 映射 到另一个分布 (一个更宽更统一的强度值分布), 所以强度值分布会在整个范围内展开.
- 要想实现均衡化的效果, 映射函数应该是一个累积分布函数 (cdf) . 对于直方图$H(i)$, 它的累积分布$H'(i)$ 是: $H'(i) = \sum_{0 \le j < i} H(j)$
- 要使用其作为映射函数, 我们必须对最大值为255 (或者用图像的最大强度值) 的累积分布$H'(i)$进行归一化. 同上例, 累积分布函数为:



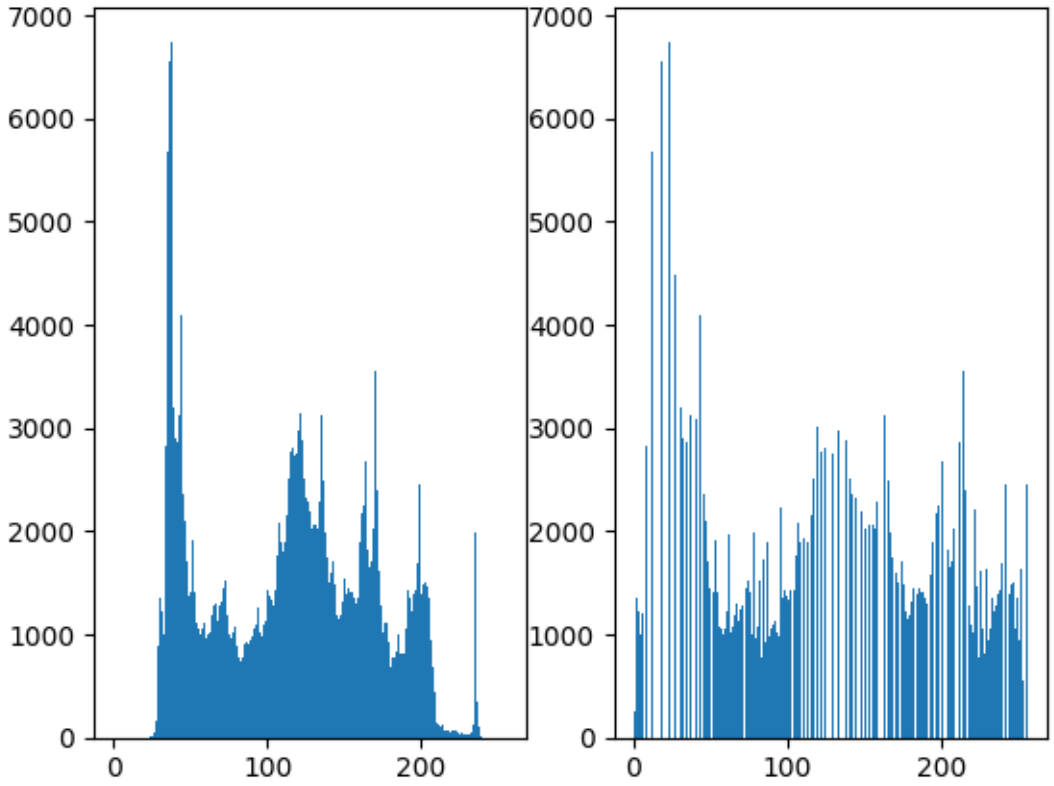
- 最后, 我们使用一个简单的映射过程来获得均衡化后像素的强度值: $equalized(x,y) = H'(src(x,y))$

Python-OpenCV图像处理:直方图均衡化

```
import cv2
import numpy as np
from matplotlib import pyplot as plt

img1 = cv2.imread('pic.jpeg')
img = cv2.cvtColor(img1,cv2.COLOR_BGR2GRAY) # 将图像转化为灰度图

equ = cv2.equalizeHist(img)  # 直方图均衡化
pic_equ = np.hstack([img,equ]) # 将两张照片加到栈里，并同时显示出来

plt.subplot(121)
plt.hist(img.ravel(),256,[0,256]) # 显示灰度图直方图
plt.subplot(122)
plt.hist(equ.ravel(),256,[0,256]) # 显示直方图均衡化后所得直方图
plt.show()
```

Python-OpenCV图像处理:直方图均衡化

- 直方图均衡化前后直方图对比：



Python-OpenCV图像处理:直方图均衡化

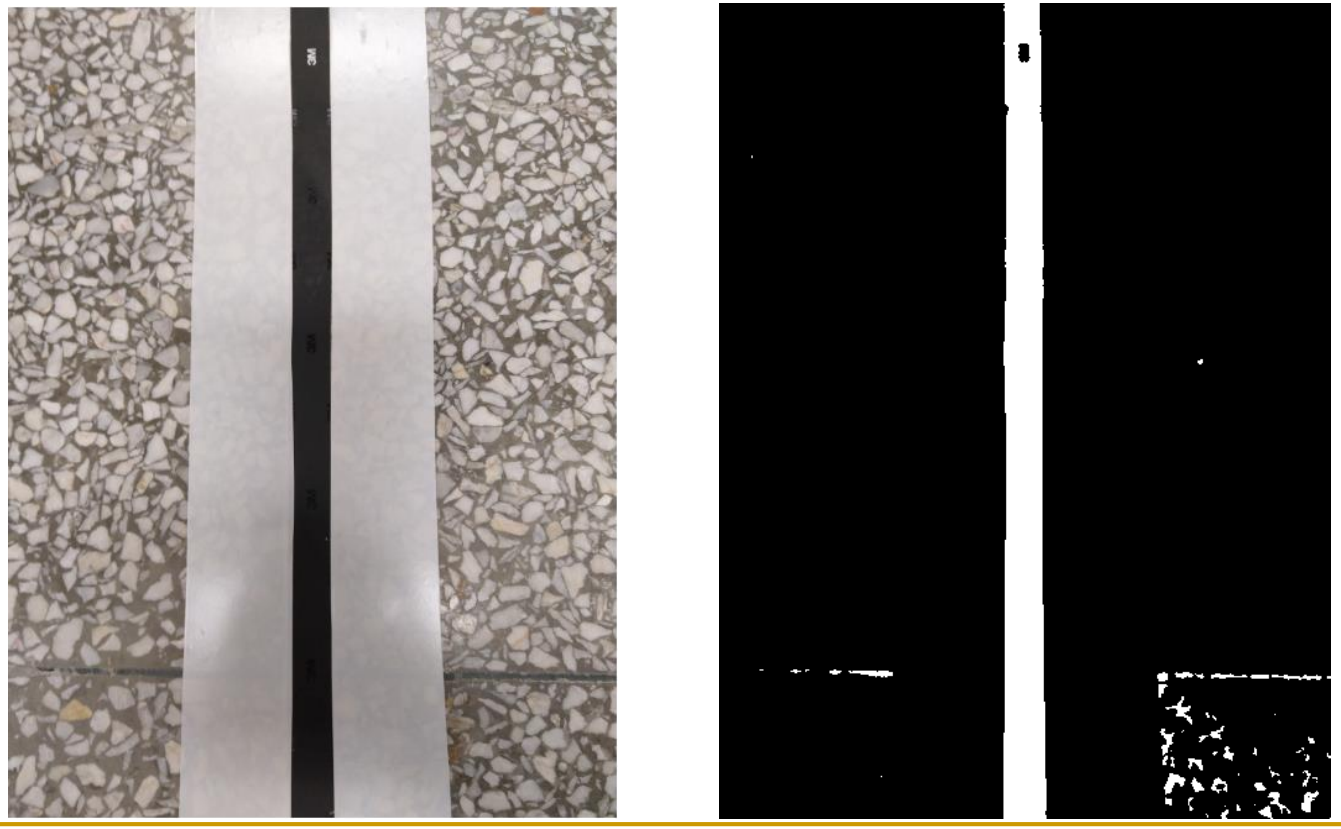
- 直方图均衡化前后图像对比：

Python-OpenCV图像处理:图像阈值处理

- 图像阈值处理是一种简化图像的方法。通过阈值处理，图像像素值取值单一，图像内容减少。
- 图像阈值处理有许多实现方法，可根据实际情况选择合适的方法，下面仅介绍“简单阈值”处理
- `ret,dst=cv2.threshold(src,thresh, maxval,type)`
- src：输入图像
- dst：输出图像　　ret：返回值
- thresh：阈值。对于8bit灰阶图像，取值范围为0至255
- maxval：像素最大值。取值范围同thresh，具体行为取决于type
- type：阈值类型，一共5种
  - cv2.THRESH_BINARY 黑白二值
  - cv2.THRESH_BINARY_INV 黑白二值反转

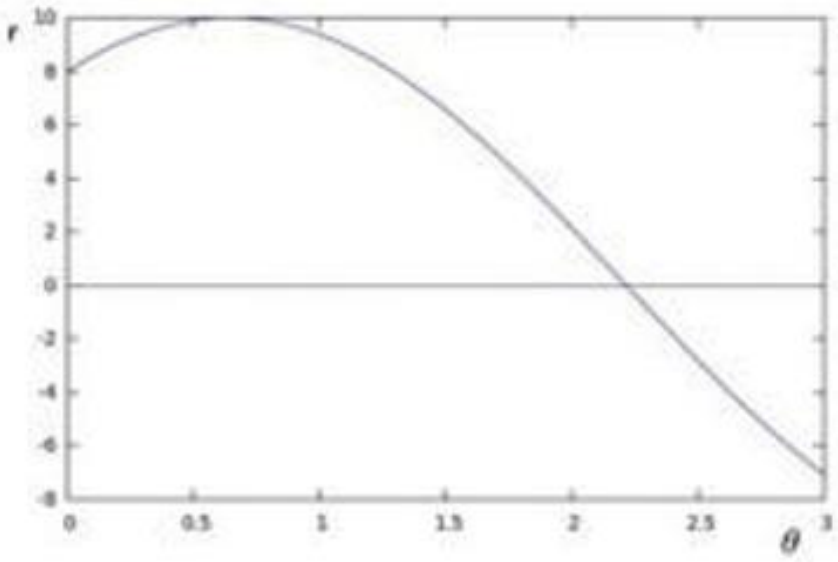
Python-OpenCV图像处理:图像阈值处理

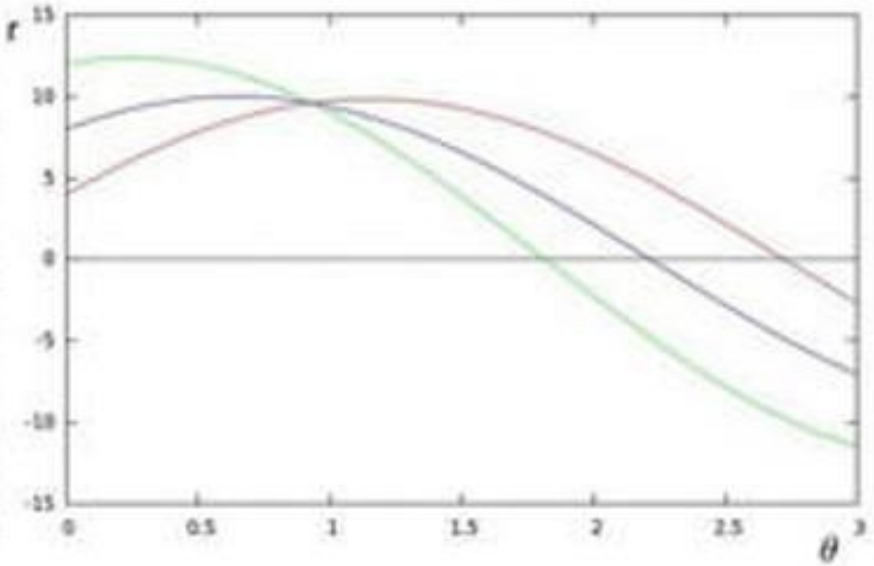
- 图像阈值处理应用举例
- 左侧为灰度图像src，右侧为经过处理后图像dst
- ret,dst=cv2.threshold(src,80,255,cv2.THRESH_BINARY_INV)



Python-OpenCV图像处理:霍夫直线变换

- 霍夫变换可以用来提取图像中的特定图形，如直线、圆。
- 霍夫直线变换原理
  - 二维平面上，定点(x_0, y_0)和极坐标对(r, θ)确定一条直线。
  - (x_0, y_0)一定，改变(r, θ)，可得到许多条经过(x_0, y_0)的直线，作$r-\theta$图像如图所示。不同的(x_i, y_j)，对应的$r-\theta$图像也不相同。
  - 若不同的(x_i, y_j)对应的$r-\theta$图像交与一点(r', θ')，可认为(x_i, y_j)共线





Python-OpenCV图像处理:霍夫直线变换

- OpenCV提供两种霍夫直线变换。以下仅介绍统计概率霍夫直线变换
- `lines=cv2.HoughLinesP(image,rho,theta,threshold[,lines[,minLineLength,[maxLineGap]]])`
- image：输入图像，需为黑白二值图像。霍夫变换处理对象为图像的白色区域。
- lines：输出直线，1x直线数x4的三维矩阵，最低维为直线始末坐标。
- rho：极坐标r步长/精度
- theta：极坐标θ步长/精度
- threshold：判断阈值。统计(r_m, θ_n)对应的(x_i, y_j)的数量，当数量大于threshold时，可认为(x_i, y_j)共线。
- minLineLength：直线最小长度。小于minLineLength时忽略该直线。
- maxLineGap：分段直线最大距离。小于maxLineGap时认为分段直线为同一直线。

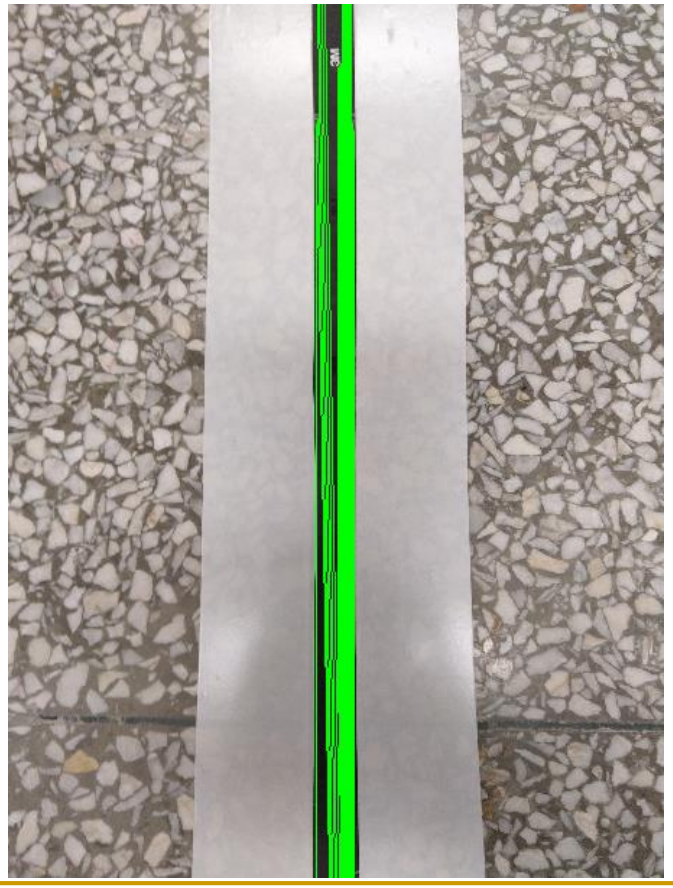
Python-OpenCV图像处理:霍夫直线变换

- 霍夫变换应用举例

```
import cv2
import math
img=cv2.imread(“pic.png”,cv2.IMREAD_COLOR)  #读取图片
imggray=cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)  #彩色图像转灰度图像
imggray=cv2.GaussianBlur(imggray,(3,3),0) #高斯去噪（选做）
#阈值处理
ret,imgbinary = cv2.threshold(imggray,80,255,cv2.THRESH_BINARY_INV)
cv2.HoughLinesP(imgbinary,1,math.pi/180,600,600,200) #霍夫变换
for x1,y1,x2,y2 in lines[0]:   #绘制直线
    cv2.line(img,(x1,y1),(x2,y2),(0,255,0),1)
cv2.imshow(“img”, img)  #显示图像
cv2.waitKey()
cv2.destroyAllWindows()
```

Python-OpenCV图像处理:霍夫直线变换

- 霍夫变换应用举例
- 左侧为原图，右侧绿线为提取出来的直线



Python-OpenCV图像处理: 空域和频域

- **空间域和频率域：**
- 空间域（spatial domain）：由图像像元组成的空间。在图像空间中以长度(距离)为自变量直接对像元值进行处理称为空间域处理。也叫空域，即所说的像素域，在空域的处理就是在像素级的处理，如在像素级的图像叠加。空间域指用图像的灰度值来描述一幅图像。

- 频率域（frequency domain）： 灰度图像的频率是表征图像中灰度变化剧烈程度的指标，是灰度在平面空间上的梯度。任何一个波形都可以分解用多个正弦波之和。每个正弦波都有自己的频率和振幅。所以任意一个波形信号有自己的频率和振幅的集合。对图像施行二维离散傅立叶变换,将图像由空间域转换到频率域。

Python-OpenCV图像处理: 图像频率

- **图像中的低频信号和高频信号：**

  也叫做低频分量和高频分量。

  图像中的高频分量，指的是图像强度（亮度/灰度）变化剧烈的地方，也就是我们常说的边缘（轮廓）；

  图像中的低频分量，指的是图像强度（亮度/灰度）变换平缓的地方.

- 图像的高低频是对图像各个位置之间强度变化的一种度量

  方法：

  低频分量:主要对整副图像的强度的综合度量

  高频分量:主要是对图像边缘和轮廓的度量

Python-OpenCV图像处理: 离散傅里叶变换

- **离散傅里叶变换：**
- 我们所看到的图像，均为空间域内的表现形式，我们无法辨识出频域内的图像。要进行频域内的滤波器处理，首先就需要进行傅里叶变换，然后直接进行滤波处理，最后再用反傅里叶变换倒回到空间域内。
- 一维离散傅里叶变换及其反变换：

  单变量离散函数$f(x)$ (其中x=0,1,2,⋯,M-1) 的傅里叶变换如下图：

$$F(u)=\frac{1}{M}\sum_{x=0}^{M-1}f(x)e^{-j2\pi ux/M},u=0,1,2,\ldots,M-1$$

  同样，给出$F(u)$，可用反DFT来获得原函数：

$$f(x)=\sum_{u=0}^{M-1}F(u)e^{j2\pi ux/M},x=0,1,2,\ldots,M-1$$

Python-OpenCV图像处理: 离散傅里叶变换

- 二维离散傅里叶变换及其反变换：

  一个图像尺寸为MxN的函数$f(x,y)$的离散傅里叶变换由以下等式给出：

$$F(u,v)=\frac{1}{MN}\sum_{x=0}^{M-1}\sum_{y=0}^{N-1}f(x,y)e^{-j2\pi(ux/M+vy/N)}$$

  与一维的情况一样，此表达式必须对u值(u=0,1,2,...,M-1)和v值(v=0,1,2,...,M-1)计算。

  同样，给出$F(u,v)$，可以通过傅里叶反变换获得$f(x,y)$：

$$f(x,y)=\sum_{u=0}^{M-1}\sum_{v=0}^{N-1}F(u,v)e^{j2\pi(ux/M+vy/N)}$$

  其中，x=0,1,2,...,M-1,y=0,1,2,...,N-1.

  变量u和v是变换或频率变量，x和y是空间或图像变量。

Python-OpenCV图像处理: 离散傅里叶变换

- **Numpy实现：**
- np.fft.fft2()：可以对信号进行频率变换，输出为一个复杂的数组。

第一个参数：灰度的输入图像，以array的形式存储。对于*M*x*N*大小的图像，数组中含有*N*个小数组，代表图像中的每一行；且每个小数组有*M*个元素，代表一个像素的灰度值。

第二个参数：可选。决定了输出数组的大小。如果大于输入图像的大小，则在计算FFT之前，输入图像用零填充。 如果小于输入图像，输入图像将被裁剪。 如果没有参数传递，输出数组大小将与输入相同。

- 此函数输出的结果，零频率分量(DC分量)将位于左上角。如果想让它在输出图像中心，还需要沿两个方向上平移*N/2*。函数np.fft.fftshift()可以完成的。进行完频率变换之后我们就可以构建振幅谱。
- f = np.fft.fft2(img) # 对图像进行FFT变换

  f_shift = np.fft.fftshift(f) # 对图像进行平移变换,平移频谱到中央

  magnitude_spectrum1 = 20 * np.log10(np.abs(f_shift))

  # 将频谱转换成dB

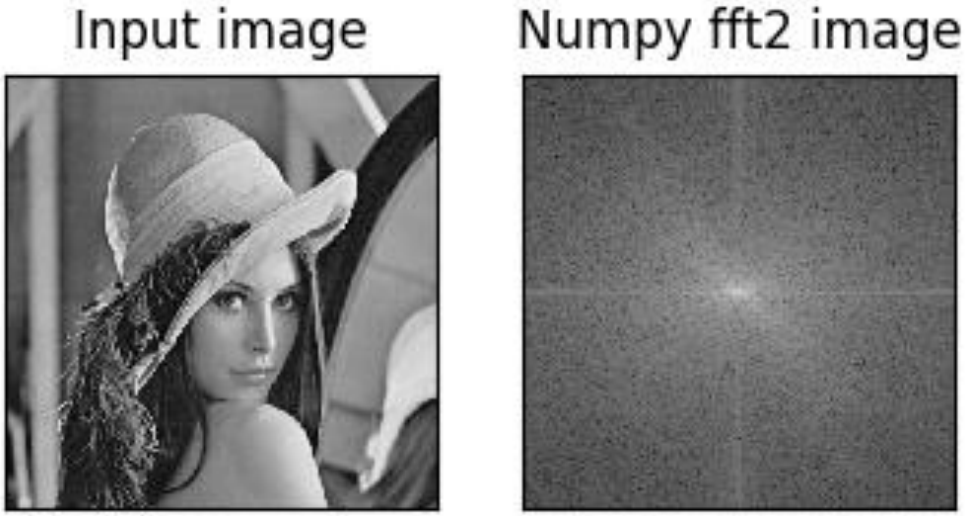
Python-OpenCV图像处理: 离散傅里叶变换

- **OpenCV实现：**
- OpenCV中相应的函数是cv2.dft()，输出为一个复杂的双通道数组。第一个通道是结果的实数部分，第二个通道是结果的虚数部分。且输入图像首先应转换成np.float32格式。

- cv2.dft(src, dst, flags, nonzeroRows=0) → None
- src：输入array，可以是实数或者复数
- dst：输出array，它的大小由标着flags决定
- flags：转换标志，详细信息可见
  https://docs.opencv.org/2.4/modules/core/doc/operations_on_arrays.html?highlight=cv2.dft#cv2.dft

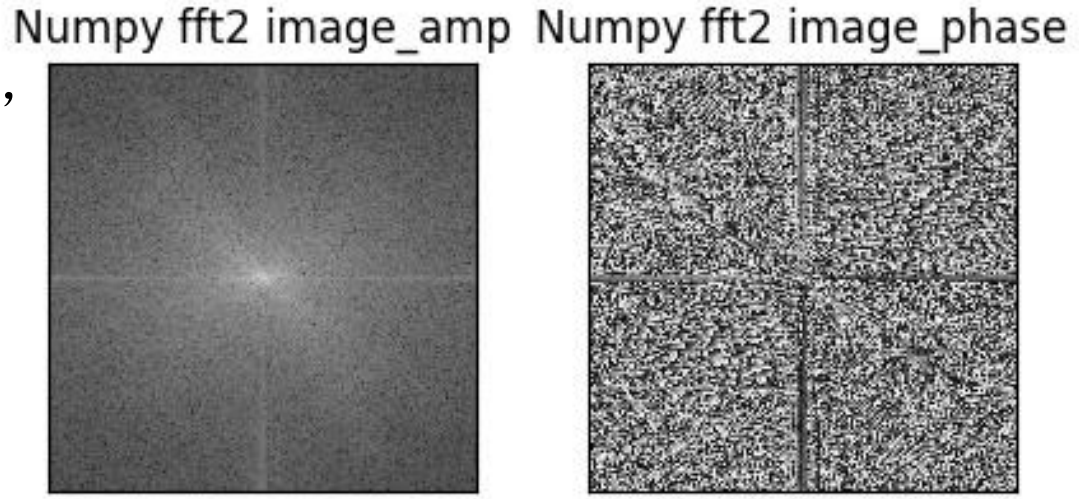
Python-OpenCV图像处理: 离散傅里叶变换

- **幅度谱和相位谱：**
- 将二维图形$f(x,y)$分解成一系列平面波的和，其中在x方向角频率是u，在y方向角频率是v。
- 原点(0,0)的傅里叶变换是图像的平均灰度。$F(0,0)$称为频率谱的直流分量，其他$F(u,v)$称为交流分量。
- 经过平移变换后的幅度谱图像，中心为直流分量$F(0,0)$。中心部分较亮说明图像的低频分量较多。



Python-OpenCV图像处理: 离散傅里叶变换

- 图像的幅度谱表现了频域中图像的振幅信息，并没有相位信息。numpy中自带一个angle函数可以直接根据复数的实部和虚部求出角度（默认出来的角度是弧度）。对离散傅里叶变换后所得f或f_shift执行np.angle()，可得图像的相位信息。
- 相位谱告诉我们每一种频率分量的相位信息。在二维傅里叶变换中，相位信息表征了各个正弦分量偏离原点的程度，也就是每一种频率分量在图像中的位置。将所得角度映射到[0,255]灰度空间，点越明亮代表角度越大，偏离原点程度越大。
- ph_f = np.angle(f)
  ph_fshift = np.angle(fshift)



Python-OpenCV图像处理: 傅里叶逆变换

- Numpy实现：

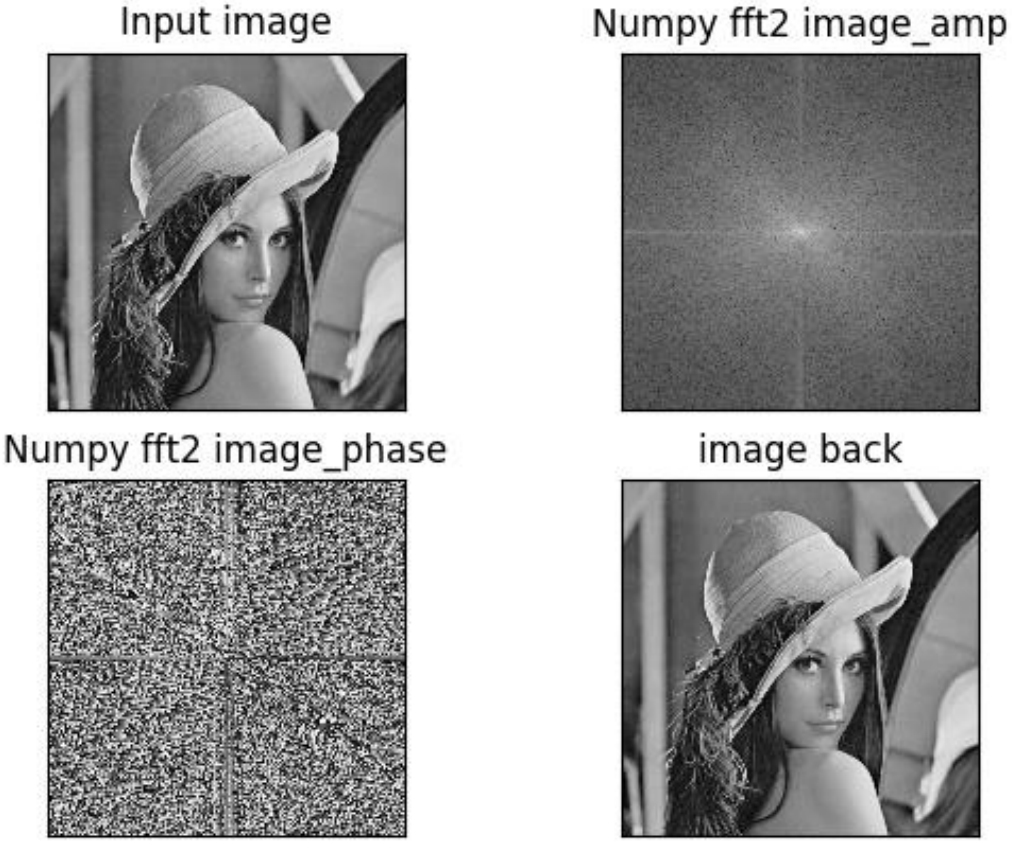
```
f1shift = np.fft.ifftshift(f_shift)  # 对图像进行平移变换
img_back1 = np.fft.ifft2(f1shift)  # 对图像进行逆傅里叶变换
img_back = np.abs(img_back1)  # 取绝对值
```

- OpenCV实现：

```
dft_ishift = np.fft.ifftshift(dft_shift)  # 对图像进行平移变换
img_back2 = cv2.idft(dft_ishift)  # 对图像进行逆傅里叶变换
img_back3 = cv2.magnitude(img_back2[:,:,0],img_back2[:,:,1])
```

Python-OpenCV图像处理: 离散傅里叶变换

- **离散傅里叶变换及逆变换效果图：**



Python-OpenCV图像处理: 图像平滑滤波

- 平滑滤波是低频增强的空间域滤波技术。它的目的有两类：一类是模糊；另一类是消除噪音。空间域的平滑滤波一般采用简单平均法进行，就是求邻近像元点的平均亮度值。邻域的大小与平滑的效果直接相关，邻域越大平滑的效果越好，但邻域过大，平滑会使边缘信息损失的越大，从而使输出的图像变得模糊，因此需合理选择邻域的大小。
- 处理要求：

  一是不能损坏图像的轮廓及边缘等重要信息；

  二是使图像清晰视觉效果好。
- 滤波函数的使用需要一个核模板，对图像的滤波操作过程为：将和模板放在图像的一个像素A上，求与之对应的图像上的每个像素点的和，核不同，得到的结果不同，而滤波的使用核心也是对于这个核模板的使用。

Python-OpenCV图像处理: 2D卷积

- OpenCV提供的函数cv.filter2D()可以对一幅图像进行卷积操作。练习一幅图像使用平均滤波器。举例下面是一个5X5的平均滤波器核：
- 操作如下，将核放在图像的一个像素A上，求与核对应的图像上25（5x5）个像素的和，再取平均数，用这个平均数代替像素A的值。重复以上操作直到将图像的每一个像素值都更新一遍。

$$K = \frac{1}{25}\begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$

- cv2.Filter2D(src, dst, kernel, anchor=(-1, -1))
  - src：输入图像.
  - dst：输出图像.
  - kernel：卷积核, 单通道浮点矩阵.
  - anchor核的锚点表示一个被滤波的点在核内的位置。

Python-OpenCV图像处理: 高斯模糊

- 现在把卷积核换成高斯核。简单的说方框不变，将原来每个方框的值是相等的，现在里面的值是符合高斯分布的，方框中心的值最大，其余方框根据距离中心元素的距离递减，构成一个高斯小山包，原来的求平均数变成求加权平均数，权就是方框里的值。实现的函数是cv2.GaussianBlur()。需要指定高斯核的宽和高（必须是奇数），以及高斯函数沿X,Y方向的标准差。如果我们只指定了X方向的标准差，Y方向也会取相同值，如果两个标准差都是0.那么函数会根据核函数的大小自己计算，高斯滤波可以有效的从图像中去除高斯噪音。

- cv2.GaussianBlur(src, ksize, sigmaX[, dst[, sigmaY[, borderType]]]) → dst
  - sigmaX:x方向的标准方差。可设置为0让系统自动计算。
  - sigmaY:y方向的标准方差。可设置为0让系统自动计算。

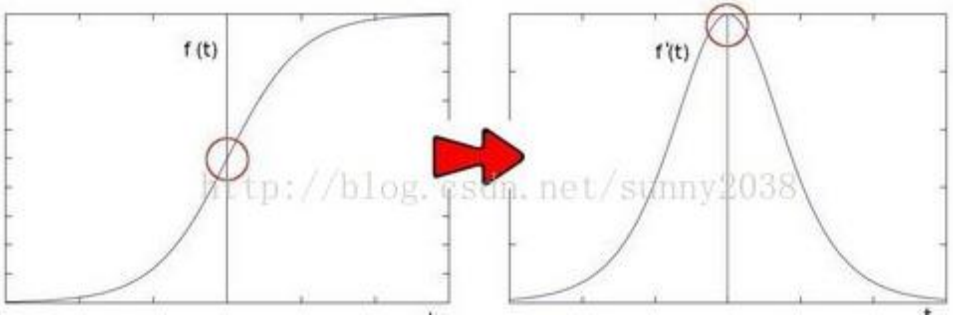
Python-OpenCV图像处理: 中值滤波

- OpenCV2函数 medianBlur 执行中值滤波操作,中值滤波模板就是用卷积框中像素的中值代替中心值，达到去噪声的目的。这个模板一般用于去除椒盐噪声。前面的滤波器都是用计算得到的一个新值来取代中心像素的值，而中值滤波是用中心像素周围（也可以使他本身）的值来取代他，卷积核的大小也是个奇数。

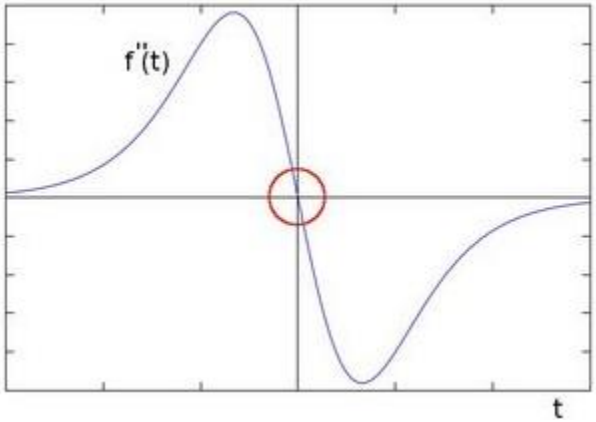
- cv2.medianBlur(src, ksize[, dst]) → dst
  - src：输入1，3或4通道图像; 当ksize为3或5时，图像深度应该是CV_8U，CV_16U或CV_32F，对于较大的光圈大小，它只能是CV_8U。
  - dst ：与src具有相同大小和类型的目标数组。
  - ksize： 卷积核大小; 它必须是奇数且大于1，例如：3,5,7 ...

Python-OpenCV图像处理: Laplace算子

- 图像中的边缘区域，像素值会发生“跳跃”，对这些像素求导，在其一阶导数在边缘位置为极值，这就是Sobel算子使用的原理——极值处就是边缘。如下图：



- 如果对像素值求二阶导数，会发现边缘处的导数值为0。



Python-OpenCV图像处理: Laplace算子

- Laplace函数实现的方法是先用Sobel 算子计算二阶x和y导数，再求和

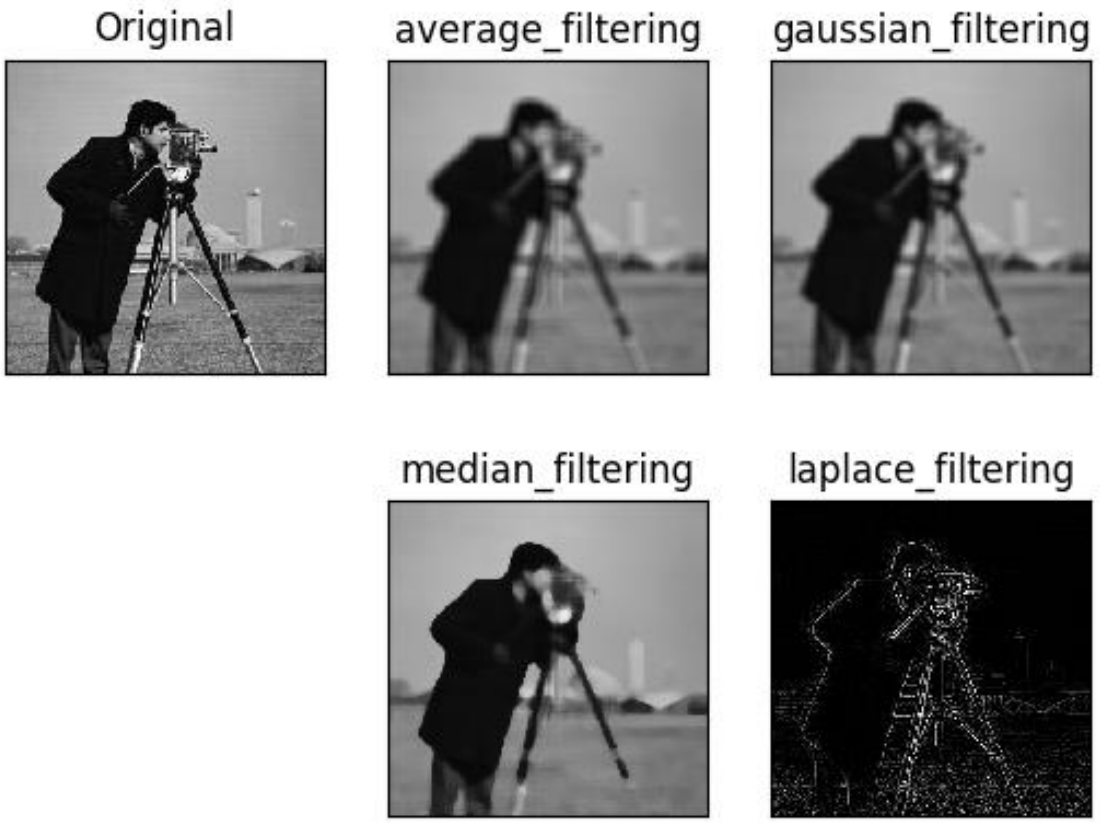
$$\mathtt{dst} = \Delta \mathtt{src} = \frac{\partial^2 \mathtt{src}}{\partial x^2} + \frac{\partial^2 \mathtt{src}}{\partial y^2}$$

- 其核模板为：

$$\text{kernel} = \begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

- cv2.Laplacian(src, ddepth[, dst[, ksize[, scale[, delta[, borderType]]]]])
  - 第一个参数是需要处理的图像；
  - 第二个参数是图像的深度，-1表示采用的是与原图像相同的深度。目标图像的深度必须大于等于原图像的深度；
  - 其后是可选的参数：
  - ksize是算子的大小，必须为1、3、5、7。默认为1。
  - scale是缩放导数的比例常数，默认情况下没有伸缩系数；
  - delta是一个可选的增量，将会加到最终的dst中，同样，默认情况下没有额外的值加到dst中；
  - borderType是判断图像边界的模式。这个参数默认值为cv2.BORDER_DEFAULT

Python-OpenCV图像处理: 结果对比



应用：磨皮美颜

- 低通滤波：去除高频噪点

参考代码列表

- camera_picture.py——拍照、显示、保存图片
- camera_movie.py——摄像、显示、保存视频
- hist_just——直方图均衡化
- HoughLine.py——二值化及直线检测
- FFT+iFFT_Numpy+OpenCV.py——离散傅里叶变换
- pic_smoothing.py——图像平滑滤波
- ColorSpaces.zip —— 各种色彩空间变换代码示例

参考资源

- Python OpenCV官方教程

https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_tutorials.html

- Learn OpenCV (C++ / Python) 网站

https://www.learnopencv.com/

- CSDN上各种技术博客

实验内容

- 在树莓派上实现从摄像头获取并保存图片。
- 实现从摄像头获取并录视频。
- 观察图片的频域系数。
- 对图片进行直方图均衡和平滑滤波。

实验报告中需要回答的问题

1. 为什么大部分自然图像低频成分比高频成分多？
2. YCrCb颜色空间相比RGB有什么好处？
3. 如果霍夫变换检测出同一位置的多条直线，如何处理？

致谢

- 本课件由以下同学协助编写
  - 覃枫凡 (14307130387)
  - 林思婕 (14300680217)
  - 肖戈川 (15307130200)