目录

实验报告者：李森

实验一 高斯滤波（GaussianBlur）、边缘检测（Canny）、直线与圆的检测(Hough)

0514 smooth- GaussianBlur

一、实验目的：

通过滤波操作来对图片中存在的噪声进行处理，让图片变得更加平滑。

二、功能说明

首先读取了一张图片，然后对其进行高斯模糊，接着显示、保存。完成对高斯模糊的应用认识。调用高斯平滑函数 cv.GaussianBlur，使用对称的平滑窗口，5x5，9x9，11x11，并显示结果。

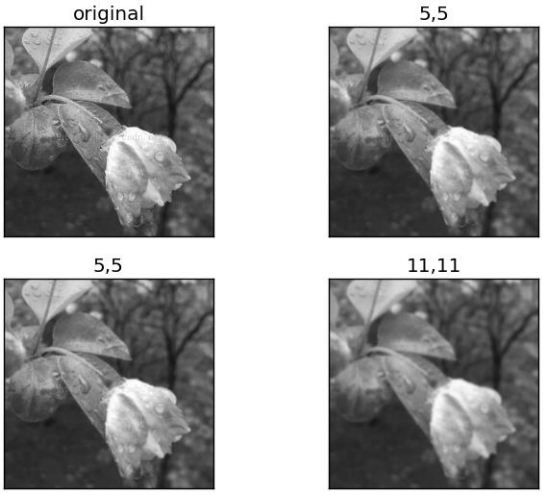
高斯滤波：cv2.GaussianBlur(img, (k,k), 0) (k 为内核的大小)

cv2.GaussianBlur 函数是 opencv 中用来对图像实现高斯滤波操作的算法。高斯滤波器时一种线性滤波器，能够有效的抑制噪声，平滑和模糊图像。

三、源程序清单及源代码（附必要注释）

```
import cv2
# import numpy as np
img = cv2.imread('v1.jpg')
#
gaussian1 = cv2.GaussianBlur(img,(5,5),0)
gaussian2 = cv2.GaussianBlur(img,(9,9),0)
gaussian3 = cv2.GaussianBlur(img,(11,11),0)
cv2.imshow('original',img)
cv2.imshow('(5,5)x1',gaussian1)
cv2.imshow('(9,9)x1',gaussian2)
cv2.imshow('(11,11)x1',gaussian3)
cv2.imwrite('(5,5)x1.jpg',gaussian1)
cv2.waitKey(-1)
```

四、实验结果分析



高斯模糊主要作用就是去除噪声。因为噪声也集中于高频信号，很容易被识别为伪边缘。应用高斯模糊去除噪声，降低伪边缘的识别。但是由于图像边缘信息也是高频信号，高斯模糊的半径选择很重要，过大的半径很容易让一些弱边缘检测不到。

0515 canny 边缘检测

一、实验目的

了解 Canny 算子阈值参数大小比例对结果的影响

二、功能说明

通过设置 cv.Canny() 中不同比例的高低阈值，观察实验结果。函数：cv2.canny(img, low_threshold, high_threshold)

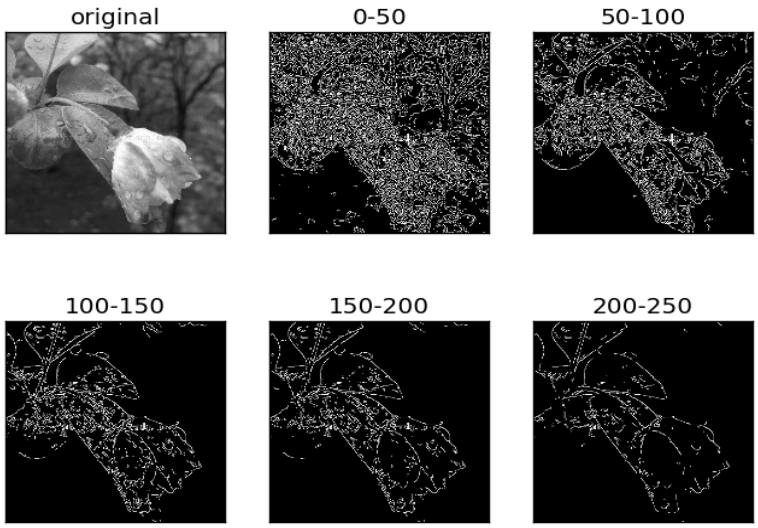
cv2.canny 函数是 opencv 中用来对图像进行边缘检测的函数算法，他有五个步骤，即使用高斯滤波对图像进行去噪、计算梯度、在边缘上使用非最大抑制、在检测到的边缘上使用双阈值去除假阳性，最后还会分析所有的边缘及其之间的连接，以保留真正的边缘并消除不明显的边缘

三、源程序清单及源代码（附必要注释）

```
import numpy as np
import cv2
import pylab
import matplotlib.pyplot as plt

img = cv2.imread('v1.jpg',0)
#设置按不同比例设置上下阈值
edges0 = cv2.Canny(img,30,45)
edges1 = cv2.Canny(img,60,90)
edges2 = cv2.Canny(img,100,150)
edges3 = cv2.Canny(img,120,180)
edges4 = cv2.Canny(img,160,240)
#利用 plt 输出结果
plt.subplot(231).imshow(img, cmap='gray')
plt.title('original'), plt.xticks([]), plt.yticks([])

plt.subplot(232).imshow(edges0, cmap='gray')
plt.title('0-50'), plt.xticks([]), plt.yticks([])

plt.subplot(233).imshow(edges1, cmap='gray')
plt.title('50-100'), plt.xticks([]), plt.yticks([])

plt.subplot(234).imshow(edges2, cmap='gray')
plt.title('100-150'), plt.xticks([]), plt.yticks([])

plt.subplot(235).imshow(edges3, cmap='gray')
plt.title('150-200'), plt.xticks([]), plt.yticks([])

plt.subplot(236).imshow(edges4, cmap='gray')
plt.title('200-250'), plt.xticks([]), plt.yticks([])
```

```
pylab.show()
cv2.imwrite('GG.jpg',Figure1)

cv2.waitKey(-1)
```

四、实验结果分析



随着阈值的提高，边缘效果越来越明显，整体边缘轮廓变得清晰。

0515 hough_lines

一、实验目的

学会用霍夫变换调参检测直线

二、功能说明

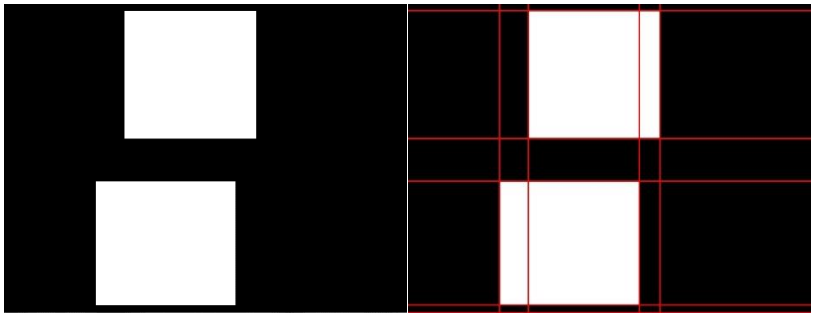
调用霍夫直线变换查看图像反应。函数：cv2.HoughLines（img, 1, np.pi/180, 200（阈值）,）。这是 opencv 中用来进行霍夫直线变换的函数，使用概率 Hough 变换。它通过分析点的子集并估计这些点都属于一条直线的概率。要注意的是这个函数接受的是单通道二值图像。最好先进过图像滤波处理和灰度图处理。

官方函数文档链接：https://docs.opencv.org/3.4.1/d6/d10/tutorial_py_houghlines.html

三、源程序清单及源代码（附必要注释）

```
import cv2 as cv
import numpy as np
img = cv.imread('v2.tif')
gray = cv.cvtColor(img,cv.COLOR_BGR2GRAY)
edges = cv.Canny(gray,50,150)
#直线检测
lines = cv.HoughLines(edges,1,np.pi/180,200)
#画线
for line in lines:
    rho,theta = line[0]
    a = np.cos(theta)
    b = np.sin(theta)
    x0 = a*rho
    y0 = b*rho
    x1 = int(x0 + 1000*(-b))
    y1 = int(y0 + 1000*(a))
    x2 = int(x0 - 1000*(-b))
    y2 = int(y0 - 1000*(a))
    cv.line(img,(x1,y1),(x2,y2),(0,0,255),2)
cv.imwrite('houghlines3.jpg',img)
cv.imshow('houghlines3.jpg',img)
cv.waitKey(-1)
```

四、实验结果分析



0515 hough_circles

一、实验目的

学会用霍夫变换调参检测圆。

二、功能说明

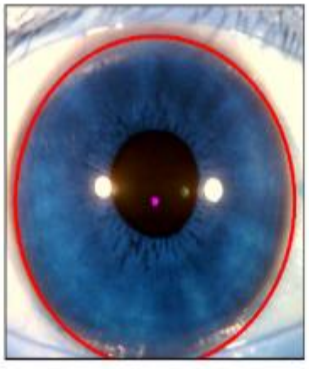
调用霍夫圆变换查看图像反应，函数：cv2.HoughCirecles(img, cv2.HOUGH_GRADIENT, 1, 120, param1=100, Param2=30, minRadius=0, maxRadius=0)

这是 opencv 中用来进行霍夫圆变换的函数，与上面函数相似，有一个圆心间的最小距离和圆的最小及最大半径。

四、源程序清单及源代码（附必要注释）

```
import cv2
import numpy as np
#import matplotlib.pyplot as plt
img = cv2.imread('v3.jpg')
gray = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
#霍夫圆检测
circles1 =
cv2.HoughCircles(gray,cv2.HOUGH_GRADIENT,1,100,param1=100,param2=30,minR
adius=150,maxRadius=200)
circles = circles1[0,:,:]
circles = np.uint16(np.around(circles))
for i in circles[:]:
    #画出外圆
    cv2.circle(img,(i[0],i[1]),i[2],(0,0, 255),5)
    #画出圆心
    cv2.circle(img,(i[0],i[1]),2,(255,0,255),10)
cv2.imshow('houghlines3.jpg',img)
cv2.imwrite('houghlines3.jpg',img)
cv2.waitKey(-1)
```

四、实验结果分析



实验二 角点检测（Harris）、SIFT 特征（sift）提取与图像拼接（sitch）

0516 corner detection

一、实验目的

对输入图像进行 Harris 角点检测并且标记出来

二、功能说明

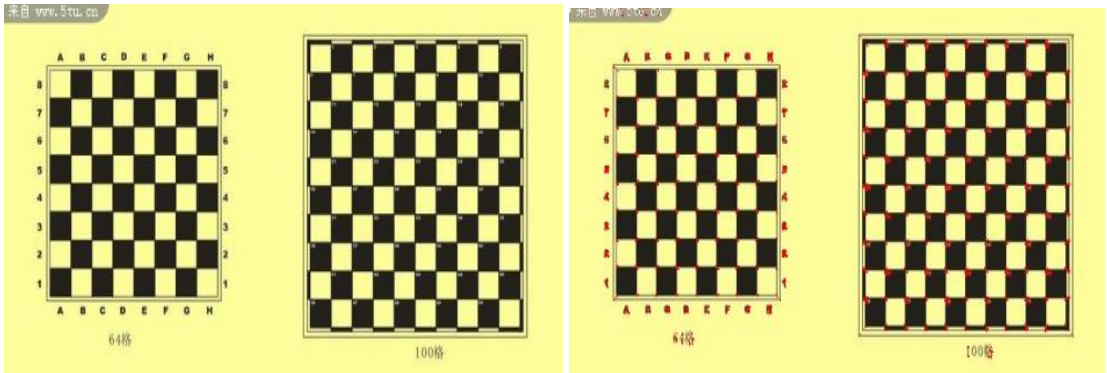
函数：cv2.cornerHarris(img, blockSize, ksize, k)

这是 opencv 中用来检测 Harris 角点的函数，其中 img 为输入图像，它应该是灰度和 float32 类型；blocksize: 这是考虑角点检测的邻域大小；ksize: 使用 Sobel 衍生物的孔径参数;k 为方程中的哈里斯检测器自由参数。

三、源程序清单及源代码（附必要注释）

```
# -*- coding:utf-8 -*-
__author__ = 'Microcosm'
import cv2
import numpy as np
img = cv2.imread("v6.jpg")
  #转化为 32 位浮点数灰度图
gray = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
gray = np.float32(gray)
  #Harris 角点检测
dst = cv2.cornerHarris(gray,2,3,0.05)
# dst = cv2.dilate(dst,None)
img[dst>0.02*dst.max()] = [0,0,255]

cv2.imshow("harris_points", img)
cv2.imwrite("harris_points.jpg", img)
if cv2.waitKey(0) & 0xff == 27:
    cv2.destroyAllWindows()
```

四、实验结果分析



Sift

一、实验目的

学会用 opencv 中的 sift 模块进行特征点提取

二、功能说明

提取图像 sift 特征点，展示结果

四、源程序清单及源代码（附必要注释）

```
import cv2
import numpy as np
img = cv2.imread('v3.png')
gray= cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
#sift 算子
sift = cv2.xfeatures2d.SIFT_create()
rows,cols = gray.shape
#深复制
gray2 = np.zeros((rows,cols))
img2 = img.copy()
a=1
b=50
#遍历图像，画出特征点
for i in range(rows):
    for j in range(cols):
            gray2[i,j]=gray[i,j]*a+b
            if gray2[i,j]>255:
                gray2[i,j]=255
            elif gray2[i,j]<0:
                gray2[i,j]=0
gray2 = np.uint8(gray2)
kp = sift.detect(gray,None)
kp2 = sift.detect(gray2,None)
img = cv2.drawKeypoints(gray,kp,img)
img2 = cv2.drawKeypoints(gray2,kp2,img2)
cv2.imshow('sp1',img)
cv2.imshow('sp2',img2)
cv2.waitKey(0)
```

四、实验结果分析

Picture_stich

一、实验目的

学会用 opencv 中的 stitch 函数进行图像拼接

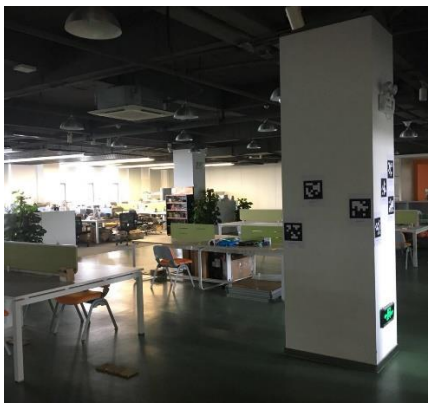
二、功能说明

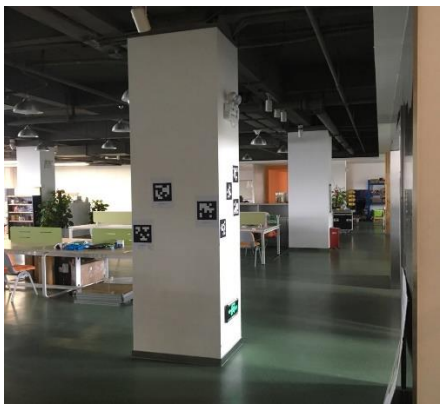
将两个不同角度拍摄的同一个场景的照片拼接为一张

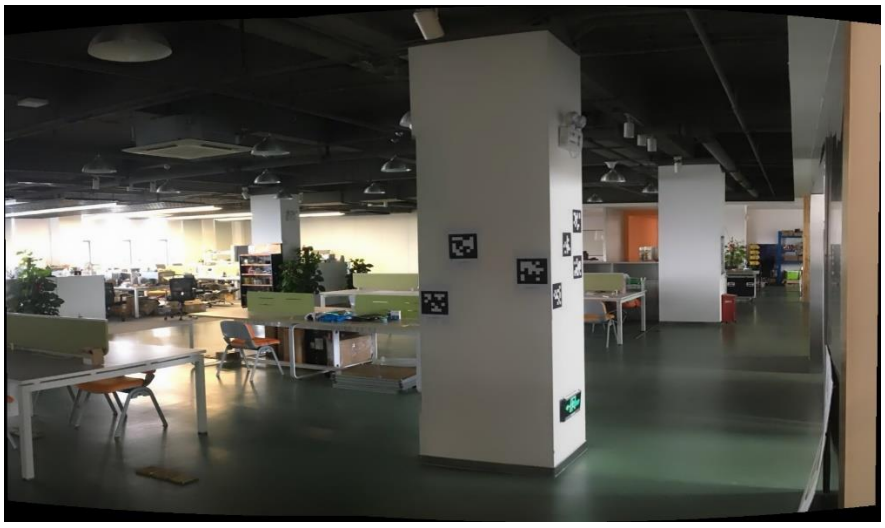
三、源程序清单及源代码（附必要注释）

```
import cv2
stitcher = cv2.createStitcher(False)
#输入两张待拼接图像
foo = cv2.imread("v1.jpg")
bar = cv2.imread("v2.jpg")
#使用 sitch 函数
result = stitcher.stitch((foo,bar))
cv2.imwrite("v3.jpg", result[1])
#输出结果
cv2.imshow("v3.jpg", result[1])
cv2.waitKey(0)
```

四、实验结果分析







实验三 （用聚类算法）实现图像分割（k-means）

0529 k-means

一、实验目的

学会用 opencv 中的 K-means 函数进行分类，对图像进行分割。

二、功能说明

使用 k-means 聚类算法，将图片中的像素点分为 4 类（设定的 K 值），之后得出分类后，用阈值分割，得到二值化图像，画出手的边缘。

cv2.kmeans(data,K, bestLabels,criteria, attempts, flags)

data: 分类数据，最好是 np.float32 的数据，每个特征放一列。

K: 分类数

bestLabels：预设的分类标签 没有的话 None

criteria：迭代停止的模式选择 3 种

attempts：重复试验 kmeans 算法次数

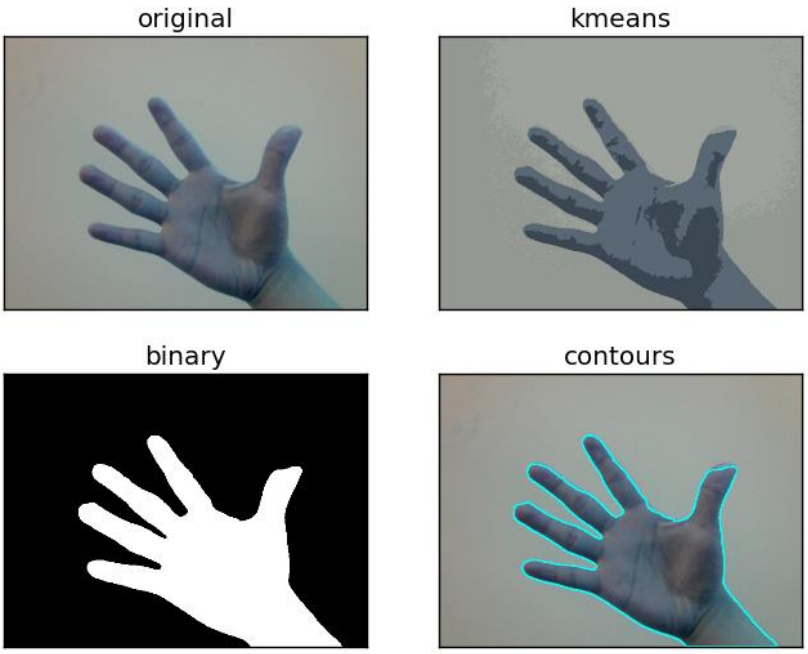
flags：初始类中心选择 2 种，如: cv2.KMEANS_RANDOM_CENTERS

三、源程序清单及源代码（附必要注释）

```
import numpy as np
import cv2
import pylab
import matplotlib.pyplot as plt

img=cv2.imread('v3.png')
img2=cv2.imread('v3.png')
Z = img.reshape((-1,3))
# convert to np.float32 图像转化为 32 位浮点图像
Z = np.float32(Z)
# define criteria, number of clusters(K) and apply kmeans() 设置参数
criteria = (cv2.TERM_CRITERIA_EPS + cv2.TERM_CRITERIA_MAX_ITER, 10, 1.0)
K = 4
ret,label,center=cv2.kmeans(Z,K,None,criteria,10,cv2.KMEANS_RANDOM_CENTE
RS)
# Now convert back into uint8, and make original image
center = np.uint8(center)
res = center[label.flatten()]
res2 = res.reshape((img.shape))
blured = cv2.blur(res2,(7,7))
#定阈值，找边界，找轮廓
```

```
gray = cv2.cvtColor(blured,cv2.COLOR_BGR2GRAY)
_, thresh = cv2.threshold(gray,140, 255, cv2.THRESH_BINARY_INV)
image, contours, hierarchy =
cv2.findContours(thresh,cv2.RETR_EXTERNAL,cv2.CHAIN_APPROX_NONE)
cnt = contours[0]
approx = cv2.approxPolyDP(cnt,0.001,False)
draw = cv2.polylines(img, [approx], False, (0, 255, 255), 2)
hull = cv2.convexHull(cnt)
#输出结果
plt.subplot(221).imshow(img2, cmap='gray')
plt.title('original'), plt.xticks([]), plt.yticks([])
plt.subplot(222).imshow(res2, cmap='gray')
plt.title('kmeans'), plt.xticks([]), plt.yticks([])
plt.subplot(223).imshow(thresh, cmap='gray')
plt.title('binary'), plt.xticks([]), plt.yticks([])
plt.subplot(224).imshow(draw, cmap='gray')
plt.title('contours'), plt.xticks([]), plt.yticks([])
pylab.show()
cv2.waitKey(0)
cv2.destroyAllWindows()
```

四、实验结果分析



小项目：剪刀石头布（手势跟踪识别应用）

实验目的 ：

基于 OpenCV 的动态手势跟踪识别，根据分类识别手势，并依据“剪刀石头布”规则瞬间做出赢人类玩家的反应的程序。
就是一个机器人跟你玩剪刀石头布，当你快出来的时候，机器识别你的手势，迅速给出反应。

功能（步骤）说明：

1. 滤波去噪
   由于边缘检测容易受到噪音影响，所以第一步是使用高斯滤波器去除噪声。
2. 去除背景，提取手部的轮廓
   2.1 方法一：肤色检测手部区域
   利用肤色来检测手部是手部检测最直接的方法。皮肤颜色稳定，不轻易受到缩放、平移和旋转的影响，且对图像的尺寸、拍摄方向和角度的依赖性较小
   颜色空间主要有两种，分别是 YCbCr 颜色空间和 HSV 颜色空间，根据 Enamin D. Zarit 等人对肤色在这些彩色空间分布的研究，以及在检测中性能的分析，本文通过对手部皮肤颜色进行特征分析(选取黄种人的肤色)，选用 HSV 颜色空间进行手部区域检测。
   RGB 颜色空间和 YCbCr 颜色空间的混合肤色检测器。像素值满足如下条件：

$$\begin{cases} R > G \wedge R > B \\ (G \geq B \wedge 5R - 12G + 7B \geq 0) \vee (G < B \wedge 5R + 7G - 12B \geq 0) \\ C_r \in (135, 180) \wedge C_b \in (85, 135) \wedge Y > 80. \end{cases}$$

   然而，利用 python 编程的结果很卡顿。
   结论分析：有点卡顿，这与 python 计算能力有关，若用 c++，则没有此现象，故使用方法二。
   此方法也有一个技术难点就是生成新的二值图像时，需要注意图像格式的问题。后来我使用了深复制的思想，完整地实现了。
   2.2 方法二：灰度图像，阈值分割：
   图像阈值化分割是一种传统的最常用的图像分割方法，因其实现简单、计算量小、性能较稳定而成为图像分割中最基本和应用最广泛的分割技术。它特别适用于目标和背景占据不同灰度级范围的图像。难点在于如何选择一个合适的阈值实现较好的分割。
   我们将 RGB 图像转变为灰度图像，便于后续处理。
   2.3 方法三：k-means 分割
   Kmeans 是最简单的聚类算法之一，应用十分广泛，Kmeans 以距离作为相似性的评价指标，其基本思想是按照距离将样本聚成不同的簇，两个点的距离越近，其相似度就越大，以得到紧凑且独立的簇作为聚类目标。
   缺点：易受光源影响，参数需根据光源修改。

3. 找出轮廓：
   利用 Opencv 中通过使用 findContours 函数，简单几个的步骤就可以检测出物体的轮廓。
4. 凸包
   凸包(凸壳)是指如果在集合 A 内连接任意两个点的直线段都在 A 的内部，则称集合 A 是凸形的。简单点理解，就是一个多边型，没有凹的地方。凸包(凸壳)能包含点集中所有的点
5. 求 moment，后求质心（用 mean shift 的方法求质心）
   计算手指的个数，来识别手势特征并进行跟踪。首先要提取手掌轮廓，计算轮廓形状特征有：轮廓的质心、轮廓的最短最长径长、轮廓的外接圆(圆心和半径)、轮廓的周长和面积、轮廓在图像中的矩形框位置、轮廓的外包络点集合、轮廓的点集合、轮廓的各阶矩、轮廓的有效的特征向量的提取、手指指尖的定位。手的位置特征是指手掌的质心位置，针对二维图像，质心位置是可以通过计算零阶距和 X、Y 的一阶距得到的。假设二值化之后的图像为 I (X , Y) ， 质心 (x c , y c) 计算公式如下：

$$M_{00} = \sum_x \sum_y I(x,y) \quad (6)$$

$$M_{10} = \sum_x \sum_y xI(x,y) \quad (7)$$

$$M_{01} = \sum_x \sum_y yI(x,y) \quad (8)$$

$$x_c = \frac{M_{10}}{M_{00}}, y_c = \frac{M_{01}}{M_{00}} \quad (9)$$

6. 标出手指和手掌
   质心处即为手掌，手指求法如下：

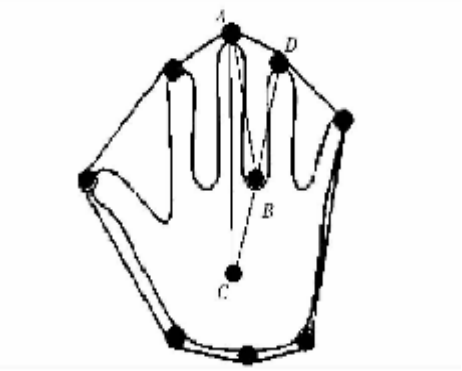
经过实验测试具体的指尖筛除条件如下:

(1) 当 $AC > BC$ 时 $0.1R \leqslant BC \leqslant 1.3R$; 当 $AC < BC$ 时 $0.1R \leqslant AC \leqslant 1.3R$;

(2) $\mathrm{MIN}(BC, AC) / \mathrm{MAX}(BC, AC) \leqslant 0.8$;

(3) $AB \geqslant 10$,$BD \geqslant 10$,$\mathrm{MAX}(AB, BD) / \mathrm{MIN}(AD, BD) \geqslant 0.8$。

将满足以上条件的指尖点存储到点集 A_p 中，然后以 A_p 中的点 P_i 为原点，在手部轮廓中各取 P_i 之前和之后的 j 个点，并计算这 $2j$ 个点的 K 曲率值，即向量 A_iA_{i-j} 与 A_iA_{i+j} 之间的夹角的余弦值。若点 P_i 的 K 曲率值小于 60°，并且是 $2j$ 个点中 K 曲率值最小的点，则其为准确的指尖点。

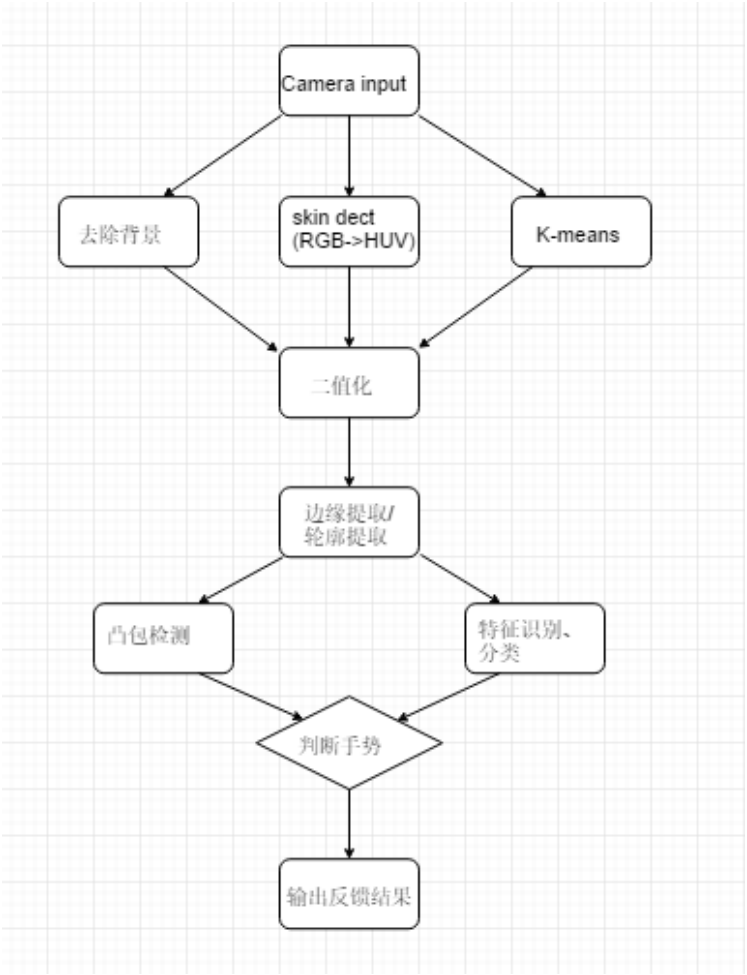


7. 判断手势和形状
   7.1 方法一：特征点
   把提取的特征点和手势字典进行对比，然后判断手势和形状
   7.2 方法二：手指的个数
   根据图像中凹凸点中的 (开始点, 结束点, 远点)的坐标, 利用余弦定理计算两根手指之间的夹角, 其必为锐角, 根据锐角的个数判别手势。
   7.3 方法三：轮廓的匹配程度
   函数 cv2.matchShape() 可以帮我们比较两个形状或轮廓的相似度。如果返回

值越小，匹配越好。它是根据 Hu 矩来计算的。

8. 输出结果：
   动态输出克制当前输出的手势（依据角刀石头布规则）

解决方案：



源程序清单及源代码（附必要注释）

以下第一份代码是使用了灰度图像，阈值分割和手指个数的方法。

```
import cv2
import numpy as np
import math
##输入结果库
pic_1 = cv2.imread('v1.png')
pic_2 = cv2.imread('v2.png')
pic_3 = cv2.imread('v3.png')
##摄像机输入
cap = cv2.VideoCapture(0)
```

```
while( cap.isOpened() ) :
    ret,img = cap.read()
    gray = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
    blur = cv2.GaussianBlur(gray,(5,5),0)
    ##阈值分割
    ret,thresh1 =
cv2.threshold(blur,70,255,cv2.THRESH_BINARY_INV+cv2.THRESH_OTSU)

    aa,contours, hierarchy =
cv2.findContours(thresh1,cv2.RETR_TREE,cv2.CHAIN_APPROX_SIMPLE)
    ##深复制
    drawing = np.zeros(img.shape,np.uint8)

    max_area=0
    ##找轮廓
    for i in range(len(contours)):
            cnt=contours[i]
            area = cv2.contourArea(cnt)
            if(area>max_area):
                max_area=area
                ci=i
    cnt=contours[ci]
    hull = cv2.convexHull(cnt)#0621
    ##meanshift 求质心
    moments = cv2.moments(cnt)
    #print len(cnt)
    #print hull
    ##求质心公式
    if moments['m00']!=0:
                cx = int(moments['m10']/moments['m00']) # cx = M10/M00
                cy = int(moments['m01']/moments['m00']) # cy = M01/M00

    centr=(cx,cy)
    cv2.circle(img,centr,5,[0,0,255],2)
    #cv2.circle(img,centr,5,[0,255,255],2)#0621
    #cv2.rectangle(original, p1, p2, (77, 255, 9), 1, 1)#0621

    cv2.drawContours(drawing,[cnt],0,(255,255,0),2)
    #cv2.drawContours(drawing,[hull],0,(0,0,255),2)

    cnt = cv2.approxPolyDP(cnt,0.01*cv2.arcLength(cnt,True),True)
    hull = cv2.convexHull(cnt,returnPoints = False)

    ndefects = 0 #0622 for counting finger_number
```

```
    ###根据图像中凹凸点中的 (开始点，结束点，远点)的坐标，可利用余弦定理计算两根手指之间的夹角，
    if(1):
                defects = cv2.convexityDefects(cnt,hull)
                #mind=0
                #maxd=0
                for i in range(defects.shape[0]):
                     s,e,f,d = defects[i,0]
                     start = tuple(cnt[s][0])
                     end = tuple(cnt[e][0])
                     far = tuple(cnt[f][0])
                     #dist = cv2.pointPolygonTest(cnt,centr,True)
                     a = np.sqrt(np.square(start[0]-end[0])+np.square(start[1]-end[1]))#0622
                     b = np.sqrt(np.square(start[0]-far[0])+np.square(start[1]-far[1]))#0622
                     c = np.sqrt(np.square(end[0]-far[0])+np.square(end[1]-far[1]))#0622


                     angle   = math.acos((b ** 2 + c ** 2 - a ** 2) / (2 * b * c)) # * 57#0622
    ##其必为锐角，根据锐角的个数判别手势
                     if angle <= math.pi/2 :#90:#0622
                           ndefects = ndefects + 1#0622

                     #cv2.line(img,start,end,[0,255,255],2)
                     cv2.line(img,start,centr,[0,255,255],2)
                     cv2.circle(img,start,20,[0,255,255],4)
                     #cv2.circle(img,end,20,[0,255,0],4)

                     cv2.circle(img,far,5,[0,0,255],-1)
                #print(i)
                print ndefects
                i=0
                if ndefects == 0:
                         print 07
                         cv2.imshow("RESULT",pic_3)
                else:
                  if ndefects == 1:
                         print 27
                         cv2.imshow("RESULT",pic_2)
                  else:
                         if ndefects == 4:
```

```
                                print 57
                                cv2.imshow("RESULT",pic_1)
                        else:
                                print 0000

    cv2.imshow('output',drawing)
    cv2.imshow('input',img)

    k = cv2.waitKey(10)
#Esc
    if k == 27:
                break
```

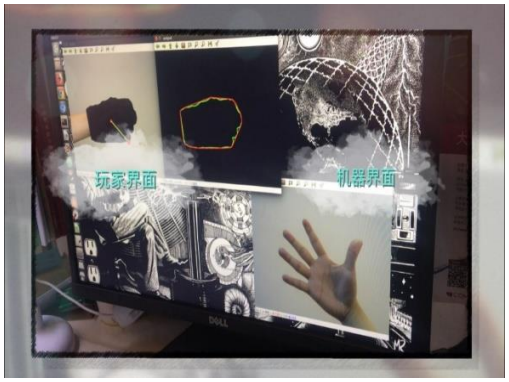
以下这份代码是用了肤色检测（将 RGB 空间转换到 Ycbcr 空间）的方法。

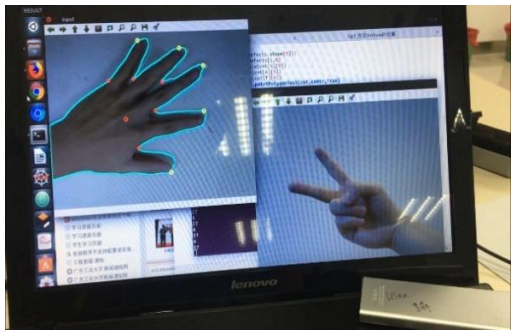
```
import cv2
import numpy as np

cap = cv2.VideoCapture(0)
while( cap.isOpened() ) :
        ret,img = cap.read()
        # load an original image
        #img = cv2.imread(imgFile)
        rows,cols,channels = img.shape
        # convert color space from rgb to ycbcr
        imgYcc = cv2.cvtColor(img, cv2.COLOR_BGR2YCR_CB)

        # convert color space from bgr to rgb
        img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

        # prepare an empty image space
        imgSkin = np.zeros(img.shape, np.uint8)
        # copy original image/深复制
        imgSkin = img.copy()
        for r in range(rows):
            for c in range(cols):

                # non-skin area if skin equals 0, skin area otherwise
                skin = 0
                # get values from rgb color space
                R = img.item(r,c,0)
                G = img.item(r,c,1)
                B = img.item(r,c,2)
```

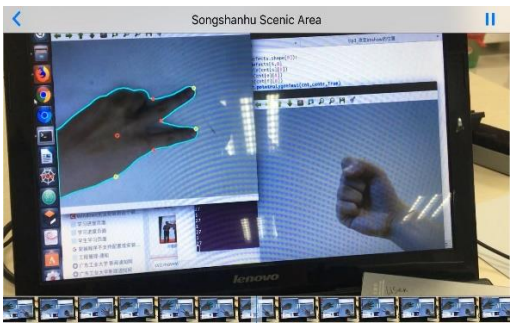
```
                # get values from ycbcr color space
                Y = imgYcc.item(r,c,0)
                Cr = imgYcc.item(r,c,1)
                Cb = imgYcc.item(r,c,2)


                # skin color detection

                if R > G and R > B:
                        if (133 <= Cr and Cr <= 173) and (77 <= Cb and
Cb <= 127):
                            skin = 1
                            # print 'Skin detected!'

                if 0 == skin:
                    imgSkin.itemset((r,c,0),0)
                    imgSkin.itemset((r,c,1),0)
                    imgSkin.itemset((r,c,2),0)
                if 1 == skin:
                    imgSkin.itemset((r,c,0),255)
                    imgSkin.itemset((r,c,1),255)
                    imgSkin.itemset((r,c,2),255)

        cv2.imshow("RESULT",imgSkin)

        k = cv2.waitKey(10)
        if k == 27:
                break
```

实验结果分析

效果见视频。







结论分析：用 python 写肤色检测程序的话，效果有点卡顿，这与 python 计算能力有关，若用 c++，则没有此现象，故使用阈值分割的方法代替。最终效果没有延迟，检测效果，能够迅速检测出手势，并且计算出手指的个数和手掌质心位置，与结果想匹配。此程序系统鲁棒性很好。