2018-2019-2 学期

多媒体技术实验报告

设　计　题　目：基于肤色分割的手势识别

学　院　名　称：信息科学技术学院

班　　级：软件工程（一）班

姓名：蔡子桐　　学号：2220161461

2019 年　6 月　10 日

《多媒体技术》实验报告

一、 实验选题

基于肤色分割的手势识别

二、 实验要求

设计一个应用系统实现相关功能，提供原始文件库，以及测试库。对系统的功能可以进行定量的评价，可以逐步演示功能模块。

三、 实验内容

1. 平台选择

本实验基于 OpenCV 库，并在使用 python 语言进行实现。

OpenCV 是一个开源的跨平台计算机视觉库，可以运行在 Linux、Windows、Android 和 Mac OS 操作系统上。它轻量级而且高效——由一系列 C 函数和少量 C++ 类构成，同时提供了 Python、Ruby、MATLAB 等语言的接口，实现了图像处理和计算机视觉方面的很多通用算法。

选用 OpenCV 的原因是能够很方便的调用连接在电脑的摄像头，而且能够清楚直观地对图像容器进行精确的，创新性的操作。

使用 python 语言的原因是其开源特性使得可以加入更多的函数库，如引入 numpy 库进行数组的操作，也方便今后的扩展。

2. 处理技术介绍

2.1 处理基本思路

基本思路如图 2.1-1 所示。

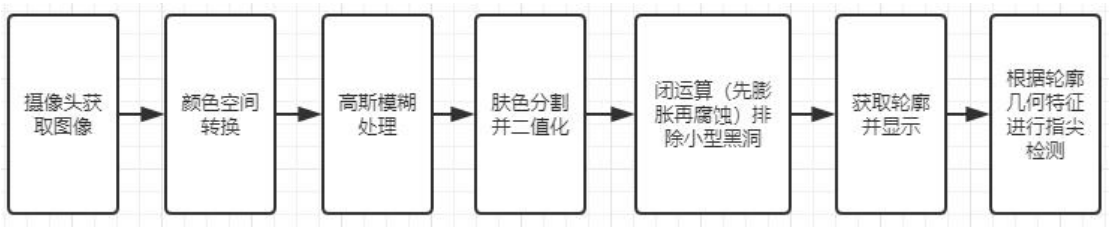

图 2.1 处理基本思路

2.2 摄像头获取图像

使用 openCV 库的 cv2.VideoCapture(0)函数获取摄像头信息，摄像头使用 Logitech HD Webcam C270(720P)。

2.3 颜色空间转换

通常摄像头得到的图像都是基于 RGB 颜色空间的，RGB 三个颜色通道分别表示红绿蓝。在 RGB 色彩空间中 R、G、B 三分量相关性强，并且光亮信息融合在这三个分量当中，RGB 空间的图像很容易收到光照影响，分别查看高亮度和低亮度条件下手掌皮肤图片(图 2.2 和图 2.3)的 RGB 通道图像。

使用 PS 软件提取手掌区域的无背景图片，进行 RGB 通道分离显示直方图，

再合并生成直方图。

图 2.2 高亮度条件下手掌图片

图 2.3 低亮度条件下手掌照片

可以从图 2.4 与图 2.5 看出，高亮度手掌与低亮度手掌的的 RGB 分布区块差异很大。由于光照的改变引起肤色亮度的变化会使 R、G、B 也存在显著的变化，在这样的条件下，在 RGB 色彩空间直接利用 R、G、B 值建立肤色模型进行肤色分割得到的结果是不可靠的。

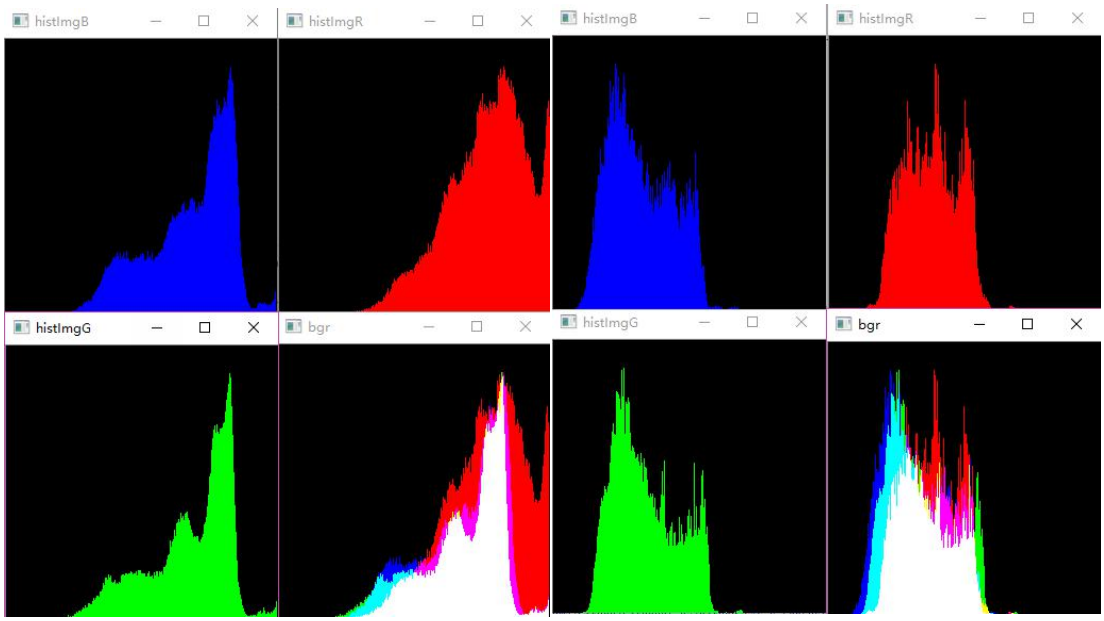

图 2.4 高亮度下 RGB 分布直方图

图 2.5 低亮度下 RGB 分布直方图

而 HSV(也叫 HSB)颜色空间都可以很好的分割出肤色区域，HSV 色彩空间中三个分量 H、S、V 分别用色调(Hue)、饱和度(Saturation)和亮度(Intensity)来描述色彩。HSV 色彩空间不仅直接反映了人类观察色彩的方式，也有利于图像处理，在对色彩信息的利用中，光照强度对于图像的影响大多体现在对于亮度分量 V 的影响。所以如果能够将亮度信息从色彩中分离出来，而只使用能反映色彩本质的色

度和饱和度来实现聚类分析，会达到很好的效果，这也是 HSV 空间在色彩图像处理和计算机视觉中应用非常广泛的原因。

将高亮度和低亮度条件下手掌皮肤无背景图片(图 2.2 和图 2.3)，转换为 HSV 颜色空间，进行 H、S、V 通道分离，并生成分布直方图。

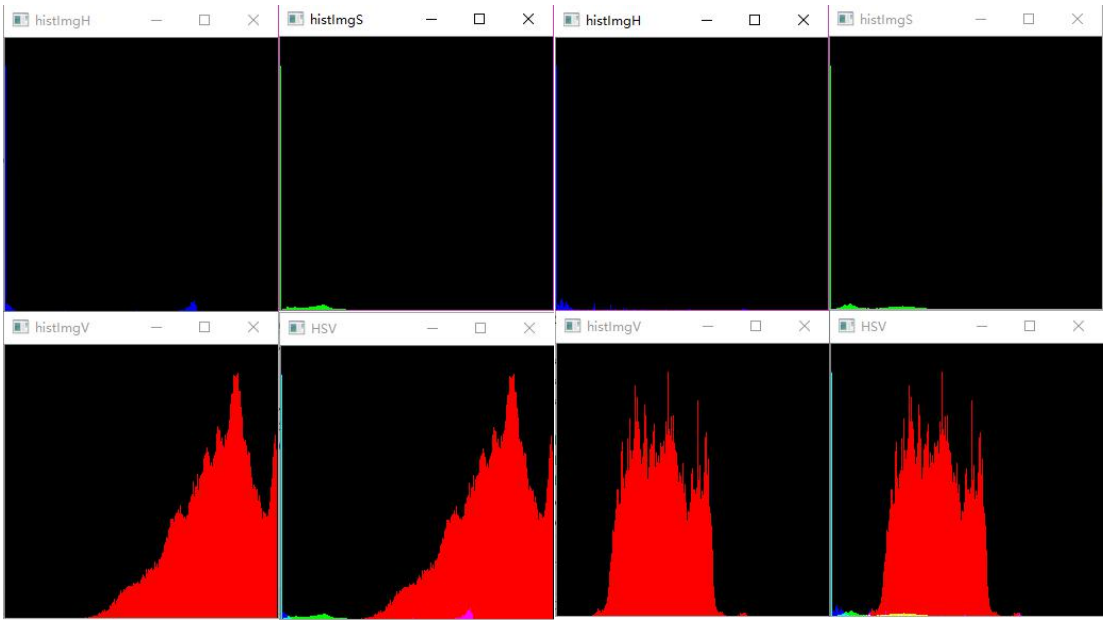

图 2.6 高亮度下 HSV 分布直方图

图 2.7 低亮度下 HSV 分布直方图

可以从图 2.6 与图 2.7 看出，高亮度手掌与低亮度手掌的的 HSV 分布中，只有 V 分量很不稳定，而 H、S 分量十分稳定在一开始直方图左侧。有一个相对固定的范围来检测肤色颜色。

RGB 颜色空间为一个立方体，而 HSV 空间可表示为一个圆柱体或圆锥体，他们之间的转换关系为：

$$H = \arccos \frac{\frac{1}{2}\left((R-G)+(R-B)\right)}{\sqrt{\left((R-G)^2+(R-B)(G-B)\right)}}$$

$$S = 1-3\frac{\min(R,G,B)}{R+G+B}$$

$$V = \frac{1}{3}(R+G+B)$$

2.4 高斯模糊处理

2.4.1 噪声

图像中噪声是指由于成像传感器噪声、相片颗粒噪声、图片在传输过程中的通道传输误差等因素会使图片上出现一些随机的、离散的、孤立的像素点，这就是图像噪声。图像噪声在视觉上通常与它们相邻的像素明显不同，例如黑区域中的白点、白区域中的黑点等。

噪声在理论上可以定义为“不可预测，只能用概率统计方法来认识的随机误差”。因此将图像噪声看成是多维随机过程是合适的，因而描述噪声的方法完全可以借用随机过程的描述，即用其概率分布函数和概率密度分布函数。

2.4.2 噪声

高斯噪声是指它的概率密度函数服从高斯分布（即正态分布）的一类噪声。如果一个噪声,它的幅度分布服从高斯分布,而它的功率谱密度又是均匀分布的,则称它为高斯白噪声。高斯白噪声的二阶矩不相关，一阶矩为常数，是指先后信号在时间上的相关性。

产生原因：1）图像传感器在拍摄时市场不够明亮、亮度不够均匀；2）电路各元器件自身噪声和相互影响；3）图像传感器长期工作，温度过高。

2.4.3 高斯模糊

高斯滤波器是一种性滤波器，能够有效的抑制噪声，平滑图像。其作用原理和均值滤波器类似，都是取滤波器窗口内的像素的均值作为输出。其窗口模板的系数和均值滤波器不同，均值滤波器的模板系数都是相同的为 1；而高斯滤波器的模板系数，则随着距离模板中心的增大而系数减小。所以，高斯滤波器相比于均值滤波器对图像个模糊程度较小。

一幅图像基本都是连续的，这也意味着越相邻的像素点之间的关系越密切，权重应该越高，越疏远的像素点之间的关系也越疏远，权重应该越低。因此我们应该使用加权平均的方法进行模糊。正态分布是一种钟形曲线,那么越接近中心，取值越大，反之越小。

由于图像是二维的，高斯模糊则将二维正态分布作为权重分配的模式。将中心点作为原点，所以二维正态分布的密度函数，即二维高斯函数公式如下：

$$G(x,y)=\frac{1}{2\pi\sigma^2}e^{-(x^2+y^2)/2\sigma^2}$$

其中(x,y)为点坐标，在图像处理中可认为是整数； σ 是标准差。
高斯滤波器宽度(决定着平滑程度)是由参数 σ 表征的,而且 σ 和平滑程度的关系是非常简单的. σ 越大，高斯滤波器的频带就越宽，平滑程度就越好. 通过调节平滑程度参数 σ ,可在图像特征过分模糊(过平滑)与平滑图像中由于噪声和细纹理所引起的过多的不希望突变量(欠平滑)之间取得折衷。

就结果而言，如未滤波图片图 2.8 和经过高斯模糊后的蹄片图 2.9 所示，可以看出高斯模糊对于后续步骤边缘识别消除了锯齿，避免了错误识别。

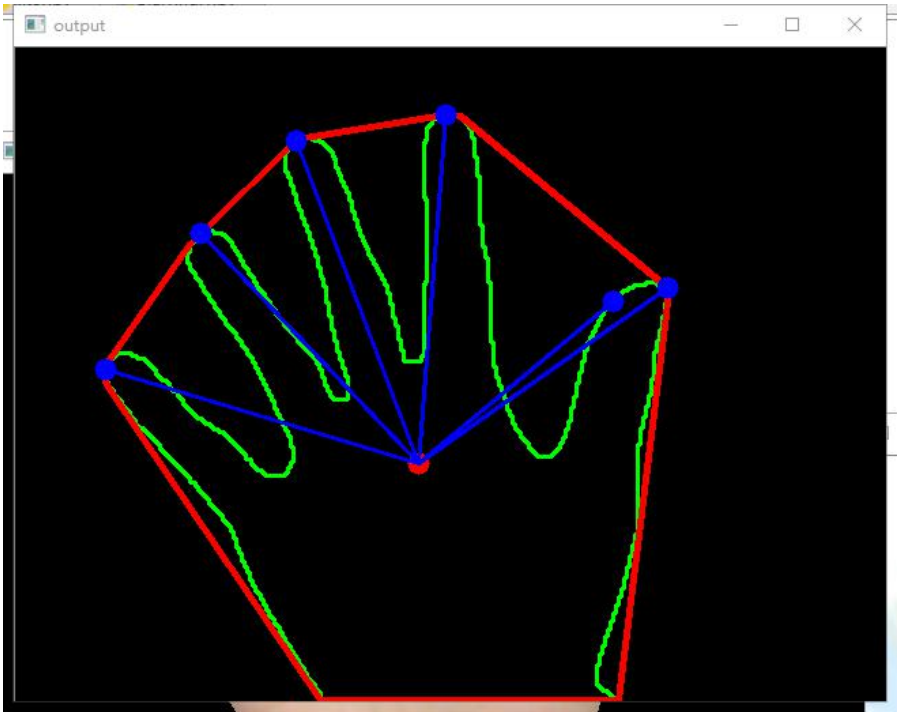

图 2. 8 无滤波处理结果

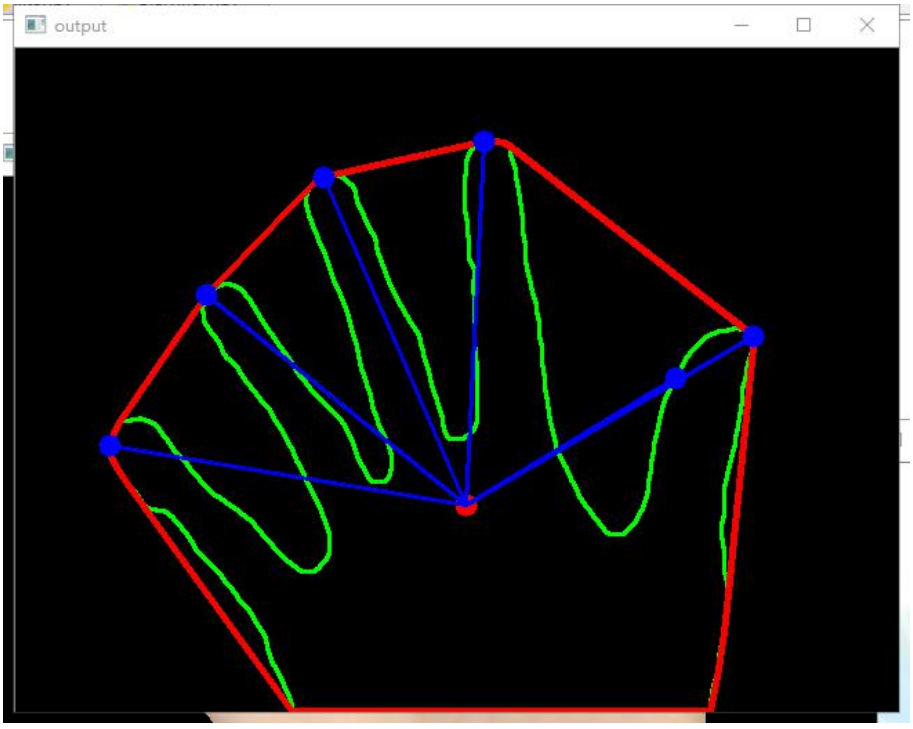

图 2. 9 经过高斯模糊处理结果

2.5 肤色分割并二值化

根据在 PS 软件中查看手掌的颜色区间，选取的感兴趣区域为：

$0 \leqslant H \leqslant 29$;

$25 \leqslant S \leqslant 100$;

$0 \leqslant V \leqslant 255$

效果如图 2.10 所示

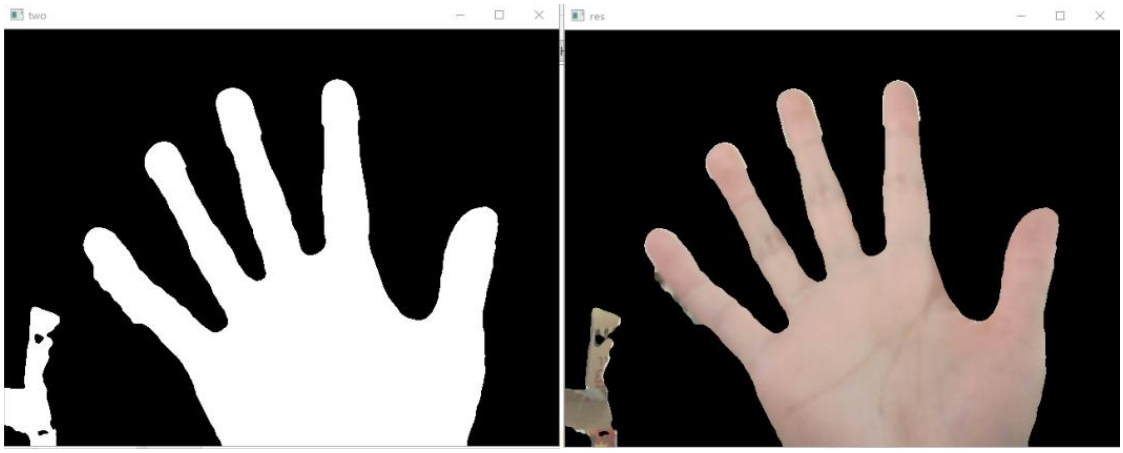

图 2.10 提取感兴趣区域

2.6 闭运算排除小型黑洞

2.6.1 腐蚀

就像土壤侵蚀一样，这个操作会把前景物体的边界腐蚀掉（但是前景仍然是白色）。这是怎么做到的呢？卷积核沿着图像滑动，如果与卷积核对应的原图像的所有像素值都是 1，那么中心元素就保持原来的像素值，否则就变为零。根据卷积核的大小靠近前景的所有像素都会被腐蚀掉（变为 0），所以前景物体会变小，整幅图像的白色区域会减少。这对于去除白噪声很有用，也可以用来断开两个连在一块的物体等。

2.6.2 膨胀

与腐蚀相反，与卷积核对应的原图像的像素值中只要有一个是 1，中心元素的像素值就是 1。所以这个操作会增加图像中的白色区域（前景）。一般在去噪声时先用腐蚀再用膨胀。因为腐蚀在去掉白噪声的同时，也会使前景对象变小。所以我们再对他进行膨胀。这时噪声已经被去除了，不会再回来了，但是前景还在并会增加。膨胀也可以用来连接两个分开的物体。

2.6.2 闭运算

闭运算是对图像先膨胀再腐蚀，可以排除小型黑洞。对实验图像进行闭运算，消除手掌中可能出现的小型黑洞，防止下一步获取轮廓时在手掌内识别到轮廓。闭运算操作前后效果如图 2.11 所示。

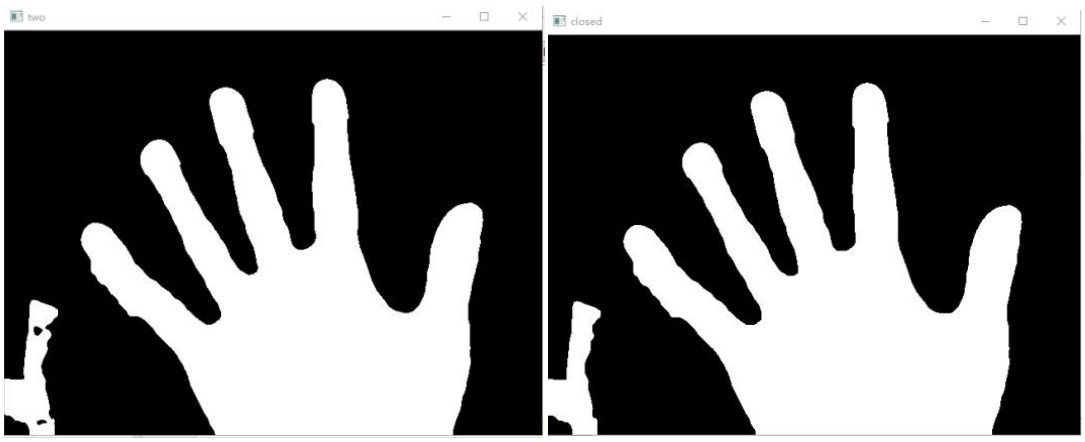

图 2.11 闭运算前后对比

2.7 获取轮廓并显示

使用 OpenCV 中的 findContours()函数寻找轮廓，并使用 drawContours()函数画出轮廓。函数 cv2.findContours()有三个参数。第一个是输入图像，第二个是轮廓检索模式，第三个是轮廓近似方法。第二个参数选用 RETR_TREE， 这种模式下会返回所有轮廓，并且创建一个完整的组织结构列表。第三个参数选用设为 cv2.CHAIN_APPROX_SIMPLE ，压缩垂直、水平、对角方向，只保留端点。

函数 cv2.drawContours()被用来绘制轮廓。第一个参数是一张图片，可以是原图或者其他。第二个参数是轮廓，也可以说是 cv2.findContours()找出来的点集，一个列表。第三个参数是对轮廓（第二个参数）的索引，当需要绘制独立轮廓时很有用，若要全部绘制可设为-1。接下来的参数是轮廓的颜色和厚度。

结果如图 2.12 所示。

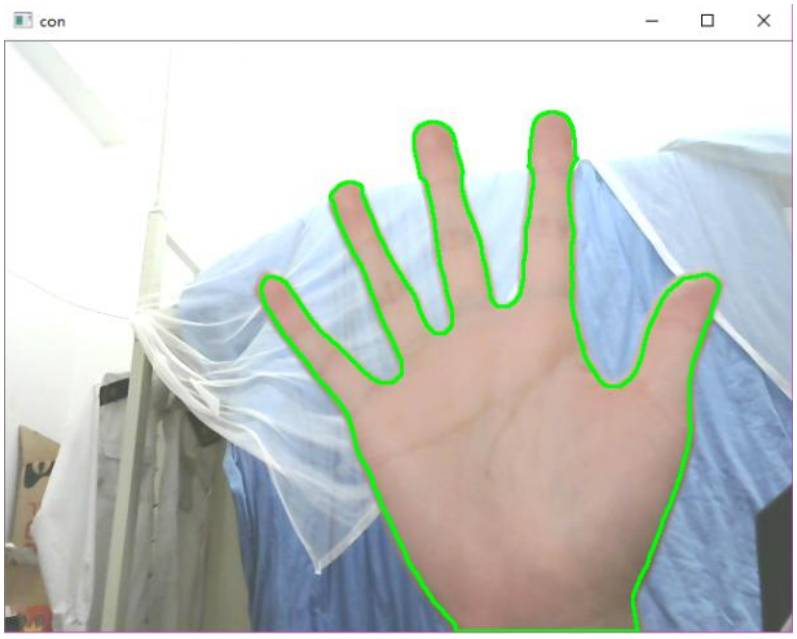

图 2.12 轮廓获取与显示

2.8 根据轮廓几何特征进行指尖检测

2.8.1 图像的几何矩

图像的几何矩是图像平面上每个像素点的值 看成该处的概率密度，对某点求期望，就是图像在该点处的矩，图像矩一般都是原点矩，可以通过一阶矩和零阶矩计算形状的重心，通过二阶矩计算形状的方向。其计算公式为：

$$\text{零阶矩}\quad M_{00}=\sum_{I}\sum_{J}V(i,j)$$

$$\text{一阶矩}\quad M_{10}=\sum_{I}\sum_{J}i\bullet V(i,j)$$

$$M_{01}=\sum_{I}\sum_{J}j\bullet V(i,j)$$

$$\text{那么图像的重心坐标，}\quad x_c=\frac{M_{10}}{M_{00}},\quad y_c=\frac{M_{01}}{M_{00}}$$

在 OpenCV 可以使用 moments（）函数计算，其中‘m00’表示面积，计算出的重心坐标可认为是掌心。使用 convexHull（）函数可找到最大外接凸多边形。

2.8.2 指尖检测

计算出面积最大的区域，将检测范围固定在此区域。计算重心到轮廓的距离，若有超过连续 120 个像素点减少，则认为这是一跟手指。再加上若干限制条件，去掉不合理的可能会被误认为手指的情况(如离掌心距离过短，低于手心的点，相邻距离太近的点等等)。

使用自带的 circle 函数和 line 函数画出手指。效果如图 2.13 所示。

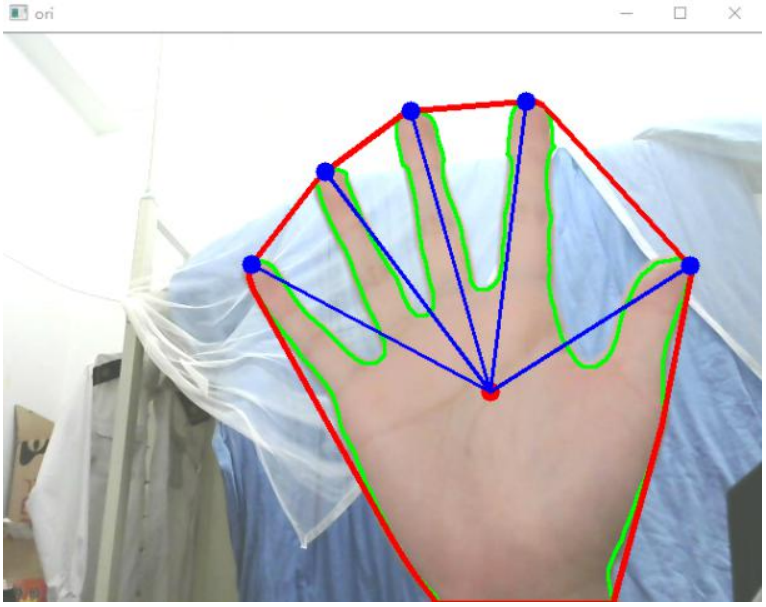

图 2. 13 指尖检测结果

3. 本实验的设计性和创新性体现

改进了手指检测部分处理技术，传统的手指检测采用了凸缺陷部分来检测手指，如图 3.1 与图 3.2 所示。

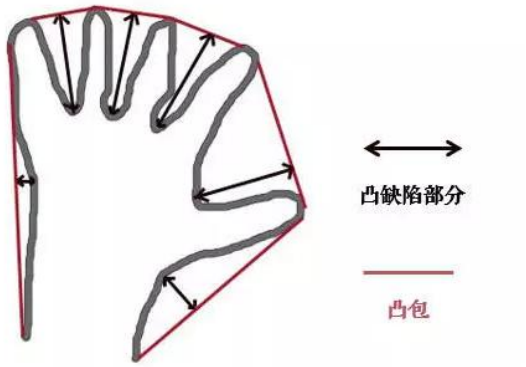

图 3. 1 凸缺陷部分与凸包

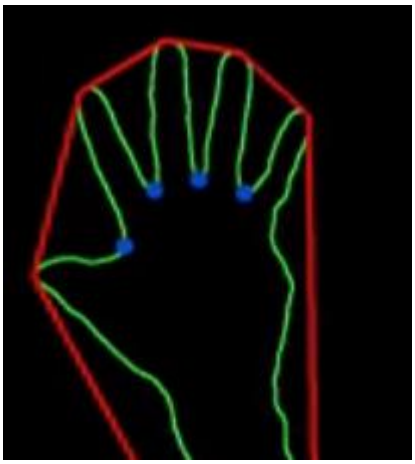

图 3. 2 检测效果

但是其在检测一根手指与没有手指的情况时，容易进行混淆，如图 3.3 所示。

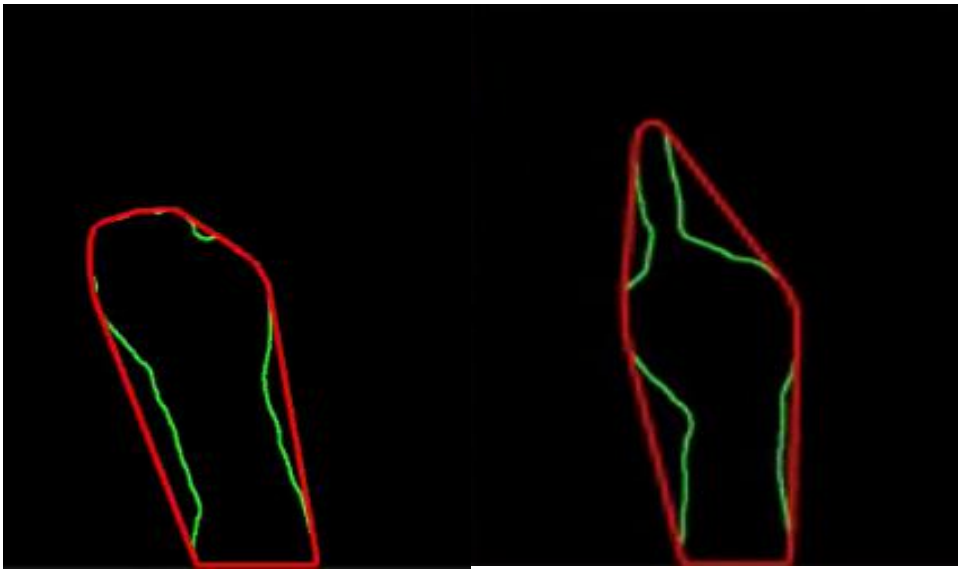

图 3. 3 无法进行区分

于是采用计算重心到轮廓的距离，若有超过连续 120 个像素点减少，则认为这是一跟手指的方法。再加上若干限制条件，去掉不合理的可能会被误认为手指的情况：离掌心距离过短排除便于识别没有手指的情况，排除低于手心的点去掉边框位置的可能被误认为指尖的点，排除相邻距离太近的点防止一根手指重复识别。具体效果如图 3.4 所示。

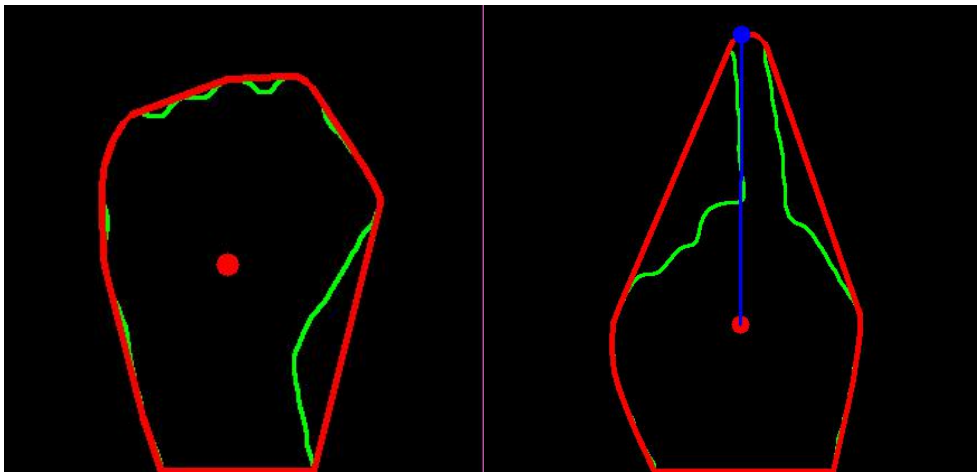

图 3.4 成功区分两种状态

4. 实现步骤

4.1 摄像头获取图像

采用 OpnenCV 的 cv2.VideoCapture(0)函数。

4.2 颜色空间转换

RGB 颜色空间转换到 HSV 颜色空间的算法:

```
max=max(R,G,B)
min=min(R,G,B)
V=max(R,G,B)
S=(max-min)/max
if  (R = max):  H =(G-B)/(max-min)* 60
if  (G = max):  H = 120+(B-R)/(max-min)* 60
if  (B = max)  :H = 240 +(R-G)/(max-min)* 60
if  (H < 0)  :H = H+ 360
```

4.3 高斯模糊处理

```
blur0 = cv2.GaussianBlur(hsv, (21, 21), 0)  # 加高斯模糊
```

4.4 肤色分割并二值化

选取的 HSV 颜色区间为:

```
lower_skin = np.array([0, 25, 0])
upper_skin = np.array([27, 100, 255])
```

使用 inRange()函数进行分割

```
mask0 = cv2.inRange(hsv, lower_skin, upper_skin)
```

4.5 闭运算排除小型黑洞

```
def opencloes(img):
   kernel = np.ones((9, 9), np.uint8)
    closed = cv2.morphologyEx(img, cv2.MORPH_CLOSE, kernel)
```

```
        cv2.imshow('closed', closed)
        return closed
```

4.6 获取轮廓并显示

使用了 OpenCV 的 cv2.findContours()函数，并用 cv2.drawContours 将其画出。

4.7 指尖检测

```
moments = cv2.moments(res)  # 求最大区域轮廓的各阶矩
center = (int(moments['m10'] / moments['m00']), int(moments['m01'] /
moments['m00']))
#moment.m00 是零阶矩,可以用来表示图像的面积,moment.m10、moment.m01 为一
阶矩；Xc = m10/m00;Yc = m01/m00 用来表示图像的重心.
cv2.circle(drawing, center, 8, (0, 0, 255), -1)  # 画出重心
cv2.circle(img, center, 8, (0, 0, 255), -1)  # 画出重心
fingerRes = []  # 寻找指尖
max = 0;
count = 0;
notice = 0;
cnt = 0
num = 0
for i in range(len(res)):
    temp = res[i]
    dist = (temp[0][0] - center[0]) * (temp[0][0] - center[0]) +
(temp[0][1] - center[1]) * (
                temp[0][1] - center[1])  # 计算重心到轮廓边缘的距离
    if dist > max:
        max = dist
        notice = i

    if dist != max:
        count = count + 1
        if max < 25000:
            continue
        if count > 120:
            count = 0
            max = 0
            flag = False  # 布尔值
            if center[1] < res[notice][0][1]:  # 低于手心的点不算
                continue
            if dist < 900:
                continue

            for j in range(len(fingerRes)):  # 离得太近的不算
```

```
                if abs(res[notice][0][0] - fingerRes[j][0]) < 40:
                    flag = True
                    break
            for j in range(len(fingerRes)):  # 离得太远的不算
                if abs(res[notice][0][0] - fingerRes[j][0]) > 500:
                    flag = True
                    break
            if flag:
                continue

            fingerRes.append(res[notice][0])
            cv2.circle(drawing, tuple(res[notice][0]), 8, (255, 0, 0),
-1)  # 画出指尖
            cv2.circle(img, tuple(res[notice][0]), 8, (255, 0, 0), -1)
# 画出指尖
            cv2.circle(img, tuple(defect), 8, (255, 0, 0), -1)  # 画出
指尖

            cv2.line(drawing, center, tuple(res[notice][0]), (255, 0,
0), 2)
            cv2.line(img, center, tuple(res[notice][0]), (255, 0, 0), 2)

            cnt = cnt + 1

print("手指根数:")
print(cnt)
```

四、 实验效果

运行过程：转换模型--提取感兴趣区域--二值化--闭运算--寻找边缘--找出重心并识别指尖，依次如图 4.1-4.5 所示.

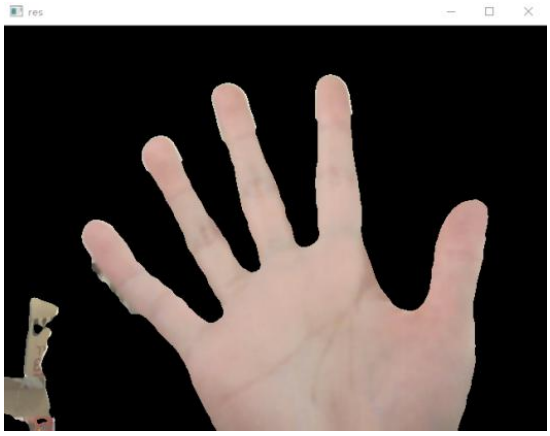

图 4.1 提取感兴趣区域

图 4.2 二值化

图 4.3 闭运算

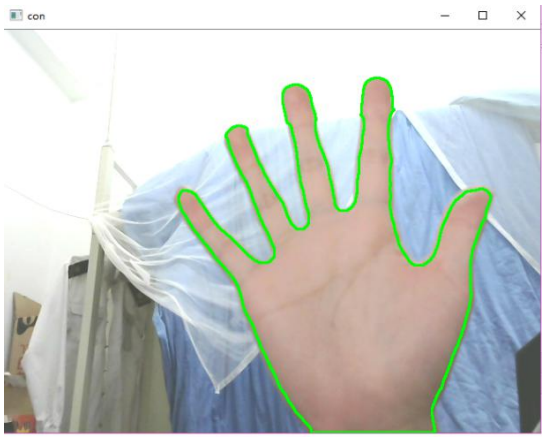

图 4.4 寻找边缘

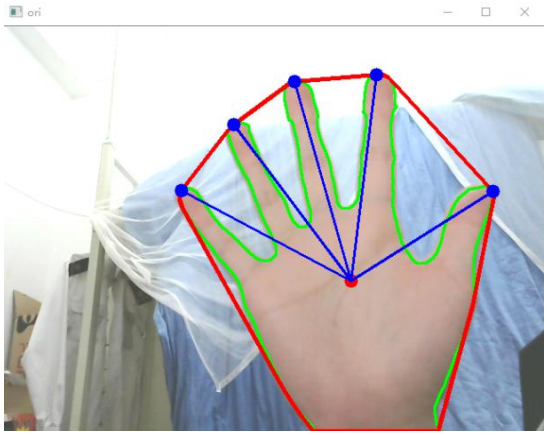

图 4.5 找出重心并识别指尖

运行结果：分别对 **0-5** 根手指的情况进行测试，结果如图 **4.6-4.12** 所示。

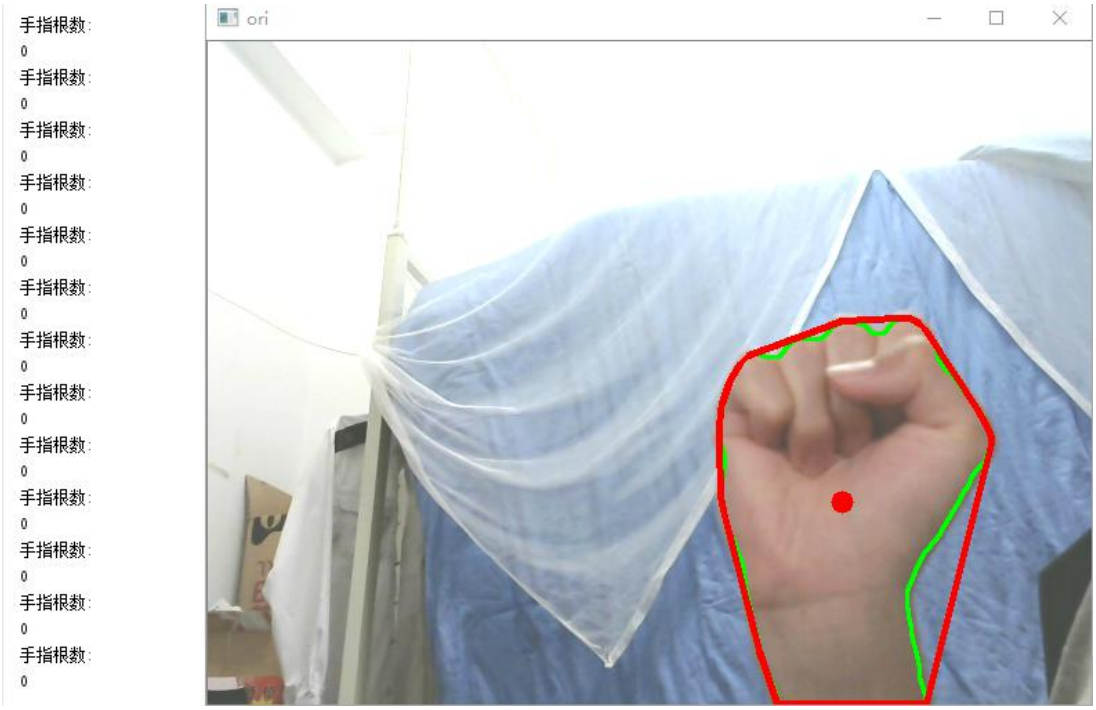

图 4.6 “0”

手指根数:
1
手指根数:
1
手指根数:
1
手指根数:
1
手指根数:
1
手指根数:
1
手指根数:
1
手指根数:
1
手指根数:
1
手指根数:
1
手指根数:
1
手指根数:
1
手指根数:
1
手指根数:
1
手指根数:
1

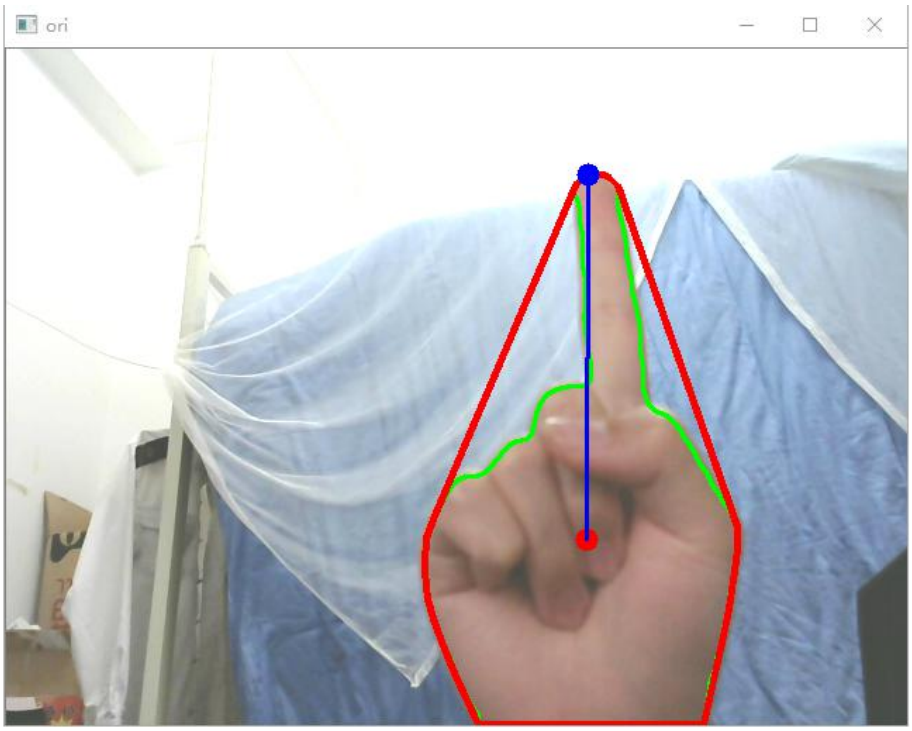

图 4.7 “1”

手指根数:
2
手指根数:
2
手指根数:
2
手指根数:
2
手指根数:
2
手指根数:
2
手指根数:
2
手指根数:
2
手指根数:
2
手指根数:
2
手指根数:
2
手指根数:
2
手指根数:
2
手指根数:
2

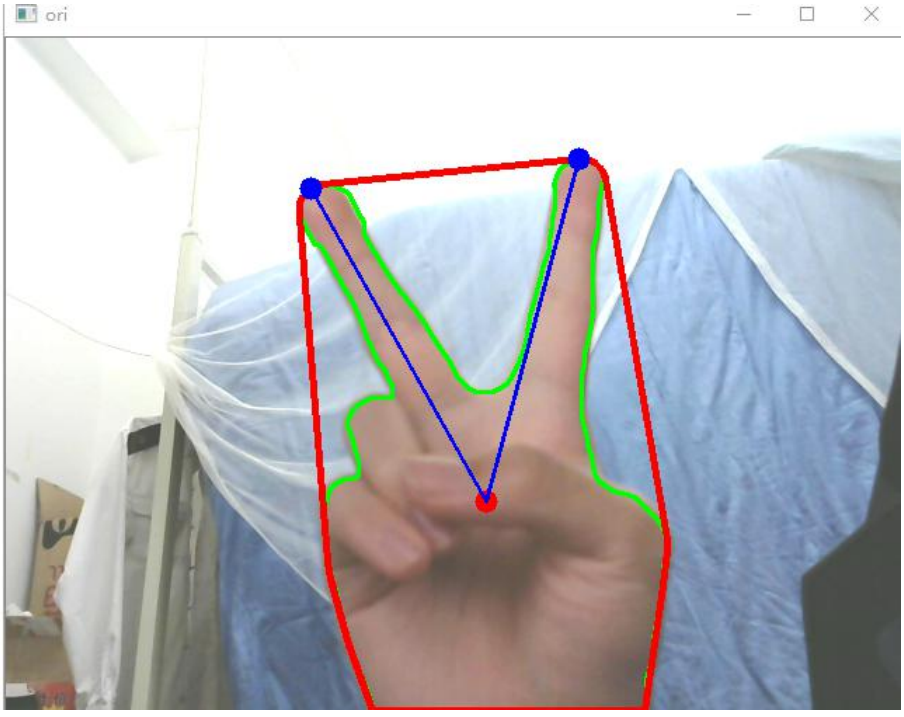

图 4.8 “2”

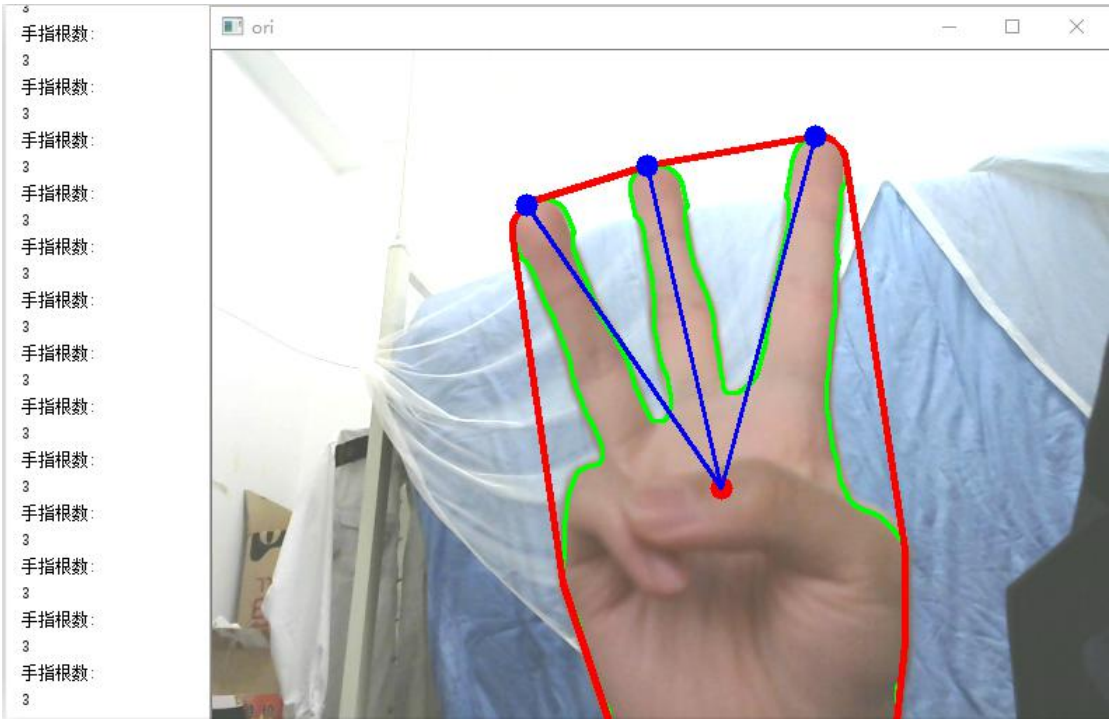
手指根数:
3
手指根数:
3
手指根数:
3
手指根数:
3
手指根数:
3
手指根数:
3
手指根数:
3
手指根数:
3
手指根数:
3
手指根数:
3
手指根数:
3
手指根数:
3
手指根数:
3
手指根数:
3
手指根数:
3
ori

图 4. 9 "3"

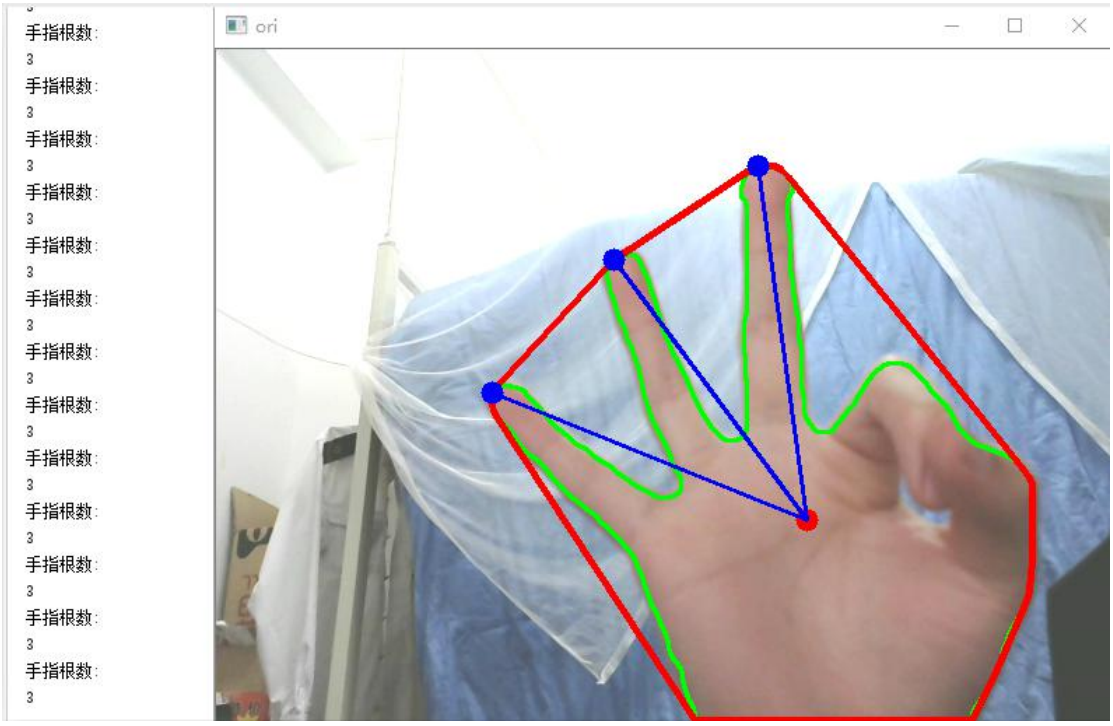
手指根数:
3
手指根数:
3
手指根数:
3
手指根数:
3
手指根数:
3
手指根数:
3
手指根数:
3
手指根数:
3
手指根数:
3
手指根数:
3
手指根数:
3
手指根数:
3
手指根数:
3
手指根数:
3
手指根数:
3
ori

图 4. 10 "3"

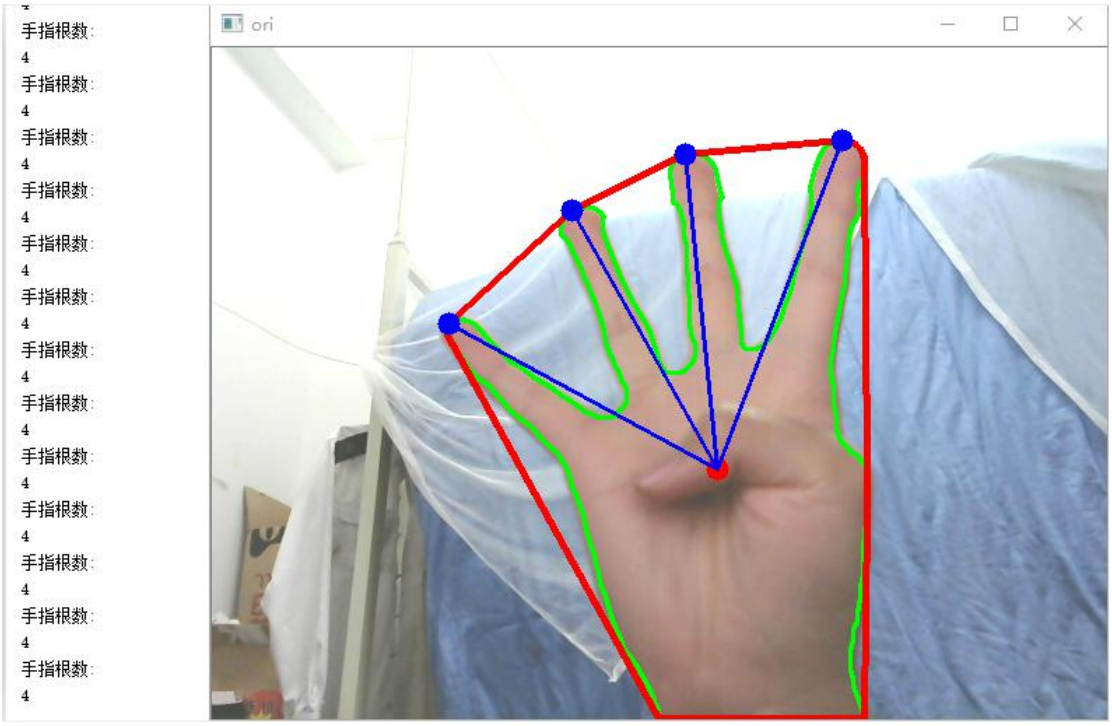
手指根数:
4
手指根数:
4
手指根数:
4
手指根数:
4
手指根数:
4
手指根数:
4
手指根数:
4
手指根数:
4
手指根数:
4
手指根数:
4
手指根数:
4
手指根数:
4
手指根数:
4
手指根数:
4
手指根数:
4
ori

图 4. 11 "4"

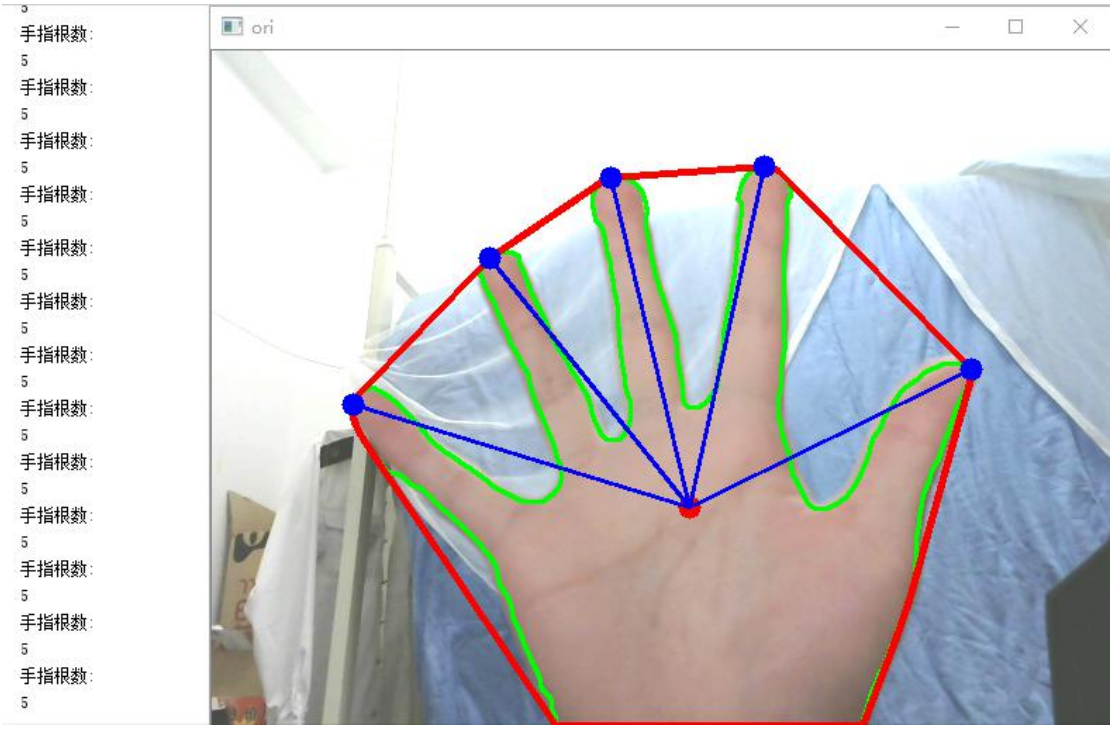
手指根数:
5
手指根数:
5
手指根数:
5
手指根数:
5
手指根数:
5
手指根数:
5
手指根数:
5
手指根数:
5
手指根数:
5
手指根数:
5
手指根数:
5
手指根数:
5
手指根数:
5
手指根数:
5
手指根数:
5
手指根数:
5
ori

图 4. 12 "5"

五、 实验结论

(1) 实验结果正常，能够有效的将手掌与背景分割开来，基本符合对指尖的识别的要求，达到了实验目的。

(2) 尽管使用了 HSV 颜色空间模型来减少光照的影响，但受到光照的影响还是存在的，环境变化较大时，要重新修改 HSV 区间来达到最好的效果。未来可以使用 HS-CrCb 空间的方式(HSV 与 YCrCb 相结合)来进一步消除光照影响，或者使用背景相减的办法来实现更好的肤色分割。

(3) 各种图片的预处理确实对后续的轮廓识别起到了关键的作用，如高斯模糊减少了轮廓边缘的锯齿，闭运算消除了手掌内的小型黑洞。

(4) 关于手势识别仍然停留在对于手指几何形状的判断，未来可以结合机器学习，进行更进一步的深化，识别更多的手势。