计算机视觉 实验一 报告

张俊华

16030199025

一、实验内容

实验目标是编写一个图像滤波函数，并用它基于 Oliva、Torralba 和 Schyns 在 SIGGRAPH 2006发表的题为 “Hybrid images” 的论文的简化版本创建混合图像。混合图像是静态图像，其解释随着观看距离的变化而变化。其基本思想是，高频往往在感知中占主导地位，但在远处，只能看到信号的低频(平滑)部分。通过将一幅图像的高频部分与另一幅图像的低频部分混合，可以得到一幅混合图像，在不同的距离产生不同的解释。你将使用你自己的解决方案来创建你自己的混合图像。

二、实验环境

Python 3.7.1 (v3.7.1:260ec2c36a, Oct 20 2018, 14:05:16) [MSC v.1915 32 bit (Intel)] on win32

PyCharm 2018.2.4 Build #PY-182.4505.26, built on September 19, 2018 Windows 10 10.0

三、实验过程

本实验要求自己实现卷积函数，并通过自己实现的卷积函数对图像进行高通和低通滤波，滤波后的图像进行叠加，就可以创建出混合图像。

这个项目需要实现5个函数，每个函数都建立在前面函数的基础上:

- `cross_correlation_2d`

  这个函数是互相关函数，其传入两个参数：img，kernel，img为待卷积的图像，kernel 为卷积核，当处理彩色图像时，img 为 height * width * 3 的三维 numpy 数组。

  只需要对彩色图像的 RGB 三个通道，分别进行卷积，即可得到卷积后的彩色图像。

  因此，定义单通道卷积函数:

```
    def cross_correlation_2d_channel(channel, kernel):
        G = np.array(channel)
        size = channel.shape
        for i in range(size[0]):
            for j in range(size[1]):
                sum = 0.0
                ksize = kernel.shape
                for u in range(ksize[0]):
                    for v in range(ksize[1]):
                        try:
                            assert (i + u - ksize[0] // 2)>=0 and (j + v - ksize[1] // 2)>=0
                            sum += kernel[u][v] * channel[i + u - ksize[0] // 2][j + v - ksize[1] // 2]
```

```
                        except Exception:
                            sum+=0
                G[i][j] = sum
        return G
```

当处理到图片外像素时，`assert (i + u - ksize[0] // 2)>=0 and (j + v - ksize[1] // 2)>=0` 触发异常，设定其值为 0

通过判断 `img.ndim` 值，以确定图片的通道数，调用通道卷积函数完成整幅图像的卷积处理

```
    if img.ndim is 3:
        newimg = np.array(img)
        for channel in range(img.ndim):
            newimg[:,:,channel] =
cross_correlation_2d_channel(img[:,:,channel],kernel)
    else :
        newimg = cross_correlation_2d_channel(img,kernel)
    return newimg
```

- `gaussian_blur_kernel_2d`

  该函数实现自定义长宽比的高斯核数组的生成

  按照定义，根据像素距离高斯核中心的距离，计算出高斯核中每个点的权值，最后，对整个矩阵进行归一化操作

```
def gaussian_blur_kernel_2d(sigma, height, width):
    height_center = height//2.0
    width_center = width//2.0
    def gaussian(x,y):
        x = x- height_center
        y = y - width_center
        rs = np.exp(-(x ** 2 + y ** 2) / (2 * (sigma ** 2)))
        return  rs
    gaussian = np.fromfunction(gaussian,(height,width))
    sum_matrix = np.sum(np.reshape(gaussian, (gaussian.size,)))
    return gaussian/sum_matrix
```

- `low_pass`

  通过 sigma 和size 值生成对应的高斯核，然后调用卷积函数，生成图片的高斯滤波版本

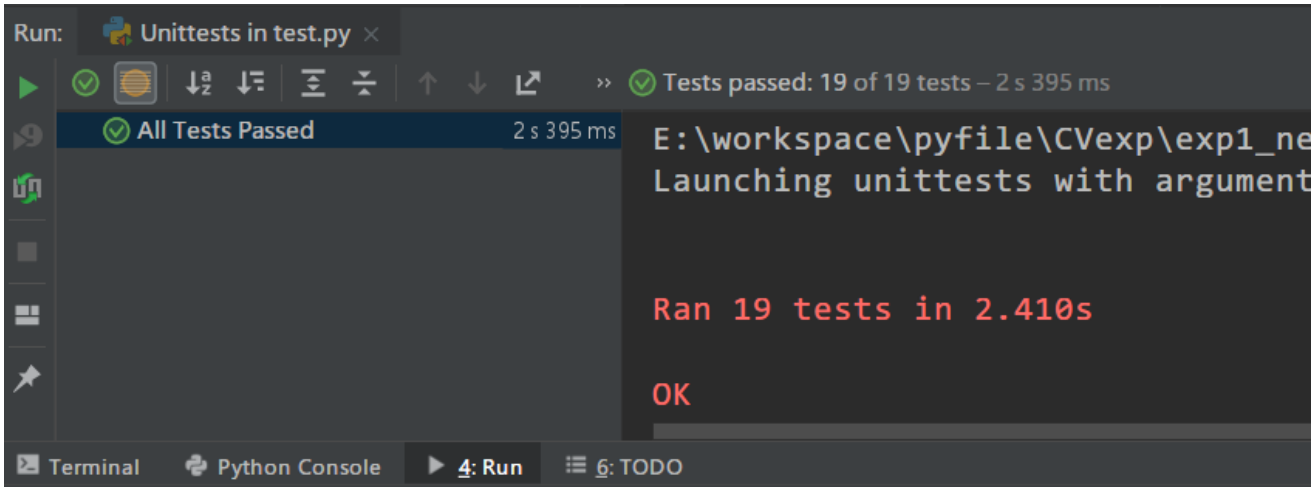
```
def low_pass(img, sigma, size):
    return convolve_2d(img,gaussian_blur_kernel_2d(sigma,size,size))
```

- `high_pass`

  用原图像与图像的低通版本作差，得到的即为图像的高通版本

```
def high_pass(img, sigma, size):
    return img - low_pass(img,sigma,size)
```

最终运行 `test.py` 对编写的函数进行测试：

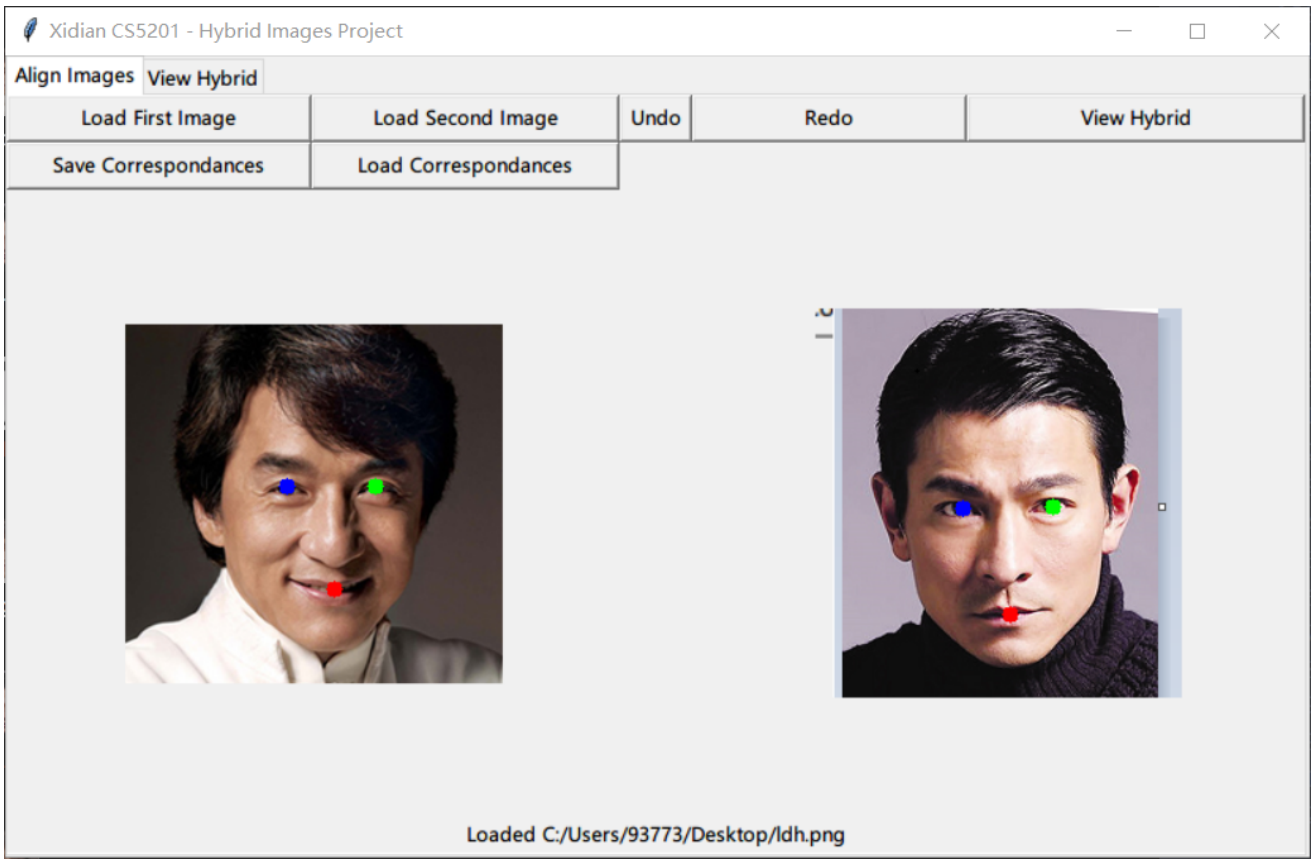


可以看到，通过了所有的测试样例

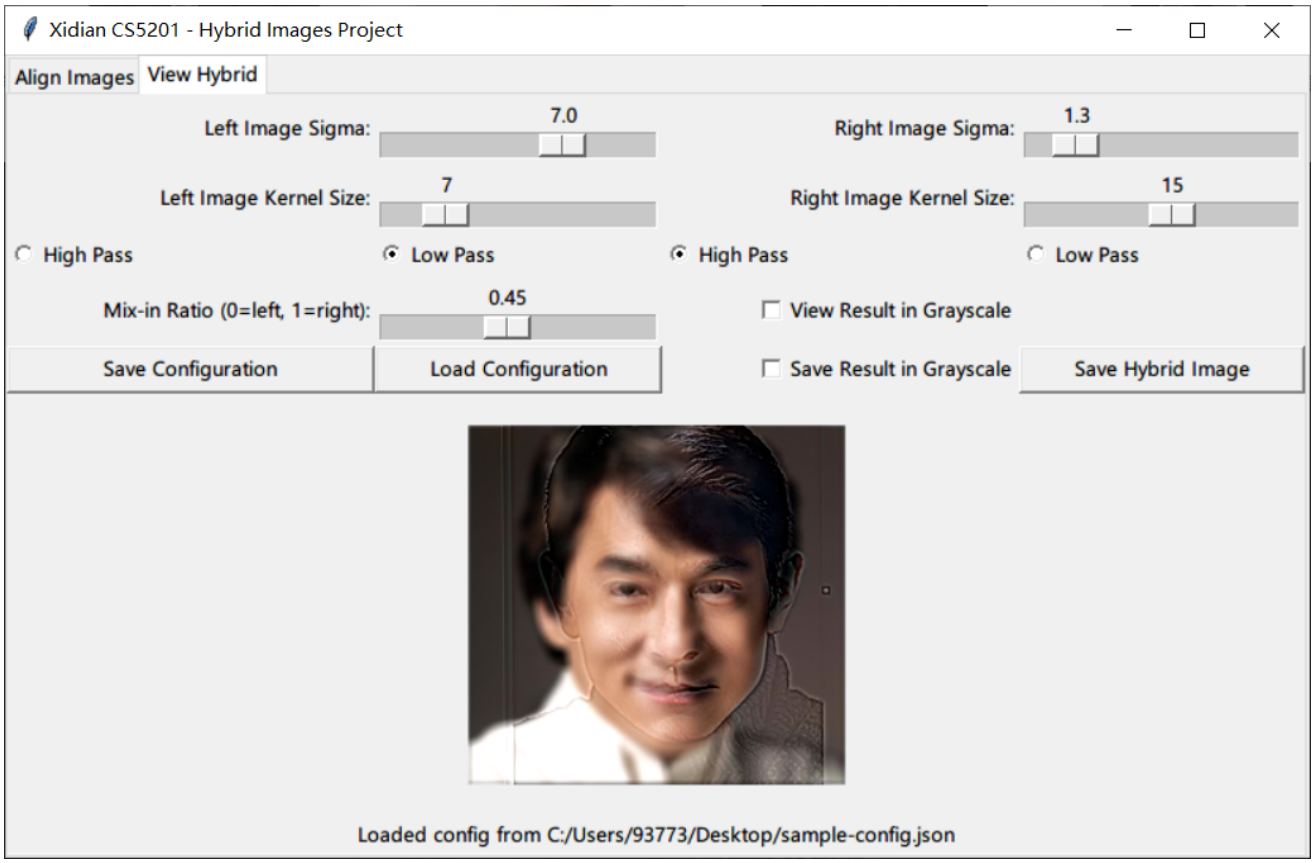
四、实验结果

选用 chenglong.png 作为 Left Image， 选用 ldh.png 作为Right Image

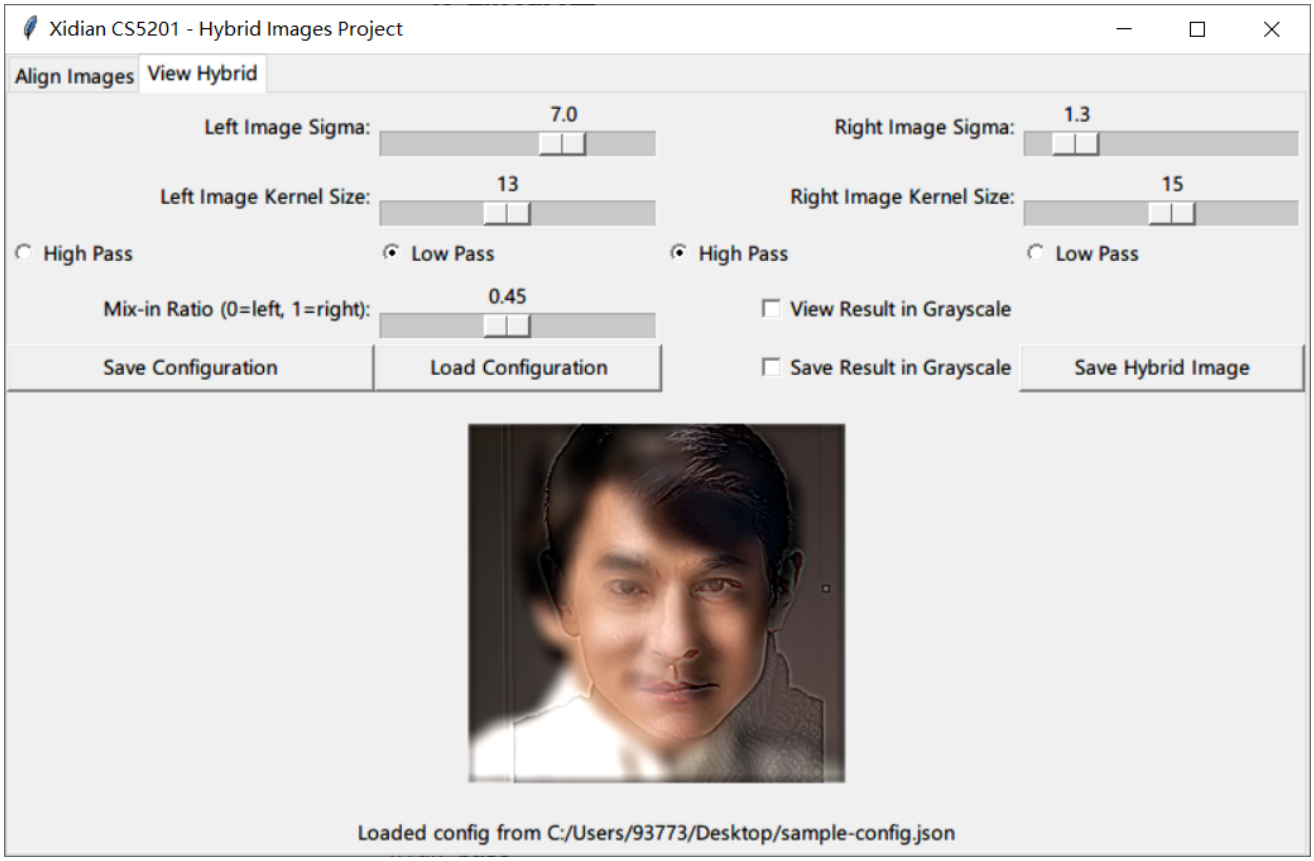
执行 gui3.py 文件，导入图像，并标记三个匹配点进行图像配准：



载入参数配置文件，观察合成效果



调整参数获得最佳合成效果



实验参数：

Left Image （成龙）：

- **低通滤波**
- **Sigma：7.0**
- **Kernel Size: 13**

right Image (刘德华)：

- **高通滤波**
- **Sigma：1.3**
- **Kernel Size：15**

混合比：0.45

最终合成的图片：

五、实验心得

本次实验是我第一次接触计算机视觉相关的实验。与以往我做过的实验不同，本次实验提供了完善的配套文档，测试样例和前端界面，这让我在实验过程中更能清楚的认识到错误之处。最后生成了可视化结果也很有成就感，使我对课堂所学的卷积、高斯滤波等知识有了更深的认识。

在实验过程中发现，由于卷积函数没有得到优化，大量的浮点运算导致图像处理过程十分缓慢。性能远不如 openCV 或 numpy 提供的卷积函数高效。查阅相关资料，发现 OpenCV 的卷积函数使用了 FFT 、多线程等其他方法进行了大量的优化，实现了快速卷积。可见，看似简单的卷积，其实现也可以非常复杂，还有很多需要我们去学习。