科学计算通识实验课-大作业

班级：暑期班 姓名：涂磊 学号：22377211 指导教师：冯成亮

1. 实验课题与内容：（列出大作业所选做的课题，内容及其目标）

第二题：图像处理

第一步：对灰度图像做最优二值变换：

1、将灰度图像读取，为矩阵，其中每个元素：

2、取两个整数，对矩阵A做数值转换：

（策略自定义）

3、寻找最小的整数组（或实数组），使变换矩阵与原矩阵A差的2范数最小；

4、对矩阵做图像输出（先转为整数形）；

第二步：对RGB图像分别做三色的最优二值变换：

1、将RGB图像读取，为3个矩阵，其中每个元素：

，其中每个元素：，其中每个元素：

2、取三组整数，，，对矩阵R,G,B做数值转换：

（分别使用两种策略赋值：（1）三色互不相关；（2）（）=() or ()）;

5、寻找最小的整数组（或实数组），，，使（，，）与原矩阵（，，）差的2范数最小；

6、对矩阵（，，）做图像输出（先转为整数形）；

7、分析查看使用两类赋值策略处理不同图片时的效果。

1. 课题分析：
2. 拟解决的关键科学问题；（为实现课题目标，需解决哪几方面的问题）
3. 0~255的三重for循环所需运行时间极高，需要想办法降低时间复杂度
4. 采取何种赋值策略可以使得赋值结果与原图相近，并更多反映出人类更在意的特征
5. 在提高对比度与刻画轮廓过程中，有多种算法可供选择，该如何抉择
6. 拟采用的研究方案；（为解决以上关键科学问题，将采用的研究方法和技术路线）
7. 为降低时间复杂度，并可以准确找到所需的最小值点，尝试使用二分法找到一个区域后再遍历
8. 由于人观察物体时更多的在意轮廓以及色块之间的对比度，所有一个直观的想法就是：用提高对比度后的图像，与原图像轮廓混合成新的图像，然后对其用适当方法二值化
9. 在提高对比度与刻画轮廓的方法选择上，我们将常用的多种算法进行分别尝试，找到了结果相对较好的算法
10. 课题解决与实现过程：
11. 理论推导；
12. 1, 首先使用OTSU阈值法，找到近似的阈值c，后以10为间隔，两层for循环遍历0~255，找到在给定c下，使得前后两矩阵差的二范数最小的a，b，再在该a,b,下计算使二范数最小的c

2, 迭代使用二分法，重复上述步骤，初始的迭代区间为（i-step,i+step）每次间隔为 step/5 或 step/10,知道step小于0.5后停止（step\_0 = 80,i = a,b,c）

3, 迭代完成后，在上一步得到的a,b,c，周围，遍历找到使得二范数最小的min\_a,min\_b,min\_c

1. 1, 找到适合的提高图像对比度的方法func\_1,与适合的刻画图像轮廓的方法func\_2,对原图像做处理得到 img\_trans = (0.8\*func\_1(img) + 0.2\*func\_2(img)),后对img\_trans使用A)中的方法，求出使其与img差的二范数最小的a,b,c,后进行二值化
2. 经过尝试，我们发现，提高色块对比度时，自适应直方图均衡化这一算法有较好的效果，在绘制轮廓时，我们使用sobel方法和scharr方法对img进行处理后，让img\_lunkuo = （0.5\*sobel(img) + 0.5\*scharr(img)) 可得到结果相对较好的轮廓
3. 程序实现；

import cv2

import numpy as np

import matplotlib.pyplot as plt

## sobel算子

def sobel\_operator(img, ksize=3):

    #img\_gray = cv2.cvtColor(img, cv2.COLOR\_BGR2GRAY)  # 灰度化

    img\_x = cv2.Sobel(img, cv2.CV\_16S, 1, 0, ksize=ksize)  # x方向sobel算子

    img\_y = cv2.Sobel(img, cv2.CV\_16S, 0, 1, ksize=ksize)  # y方向sobel算子

    img\_absx = cv2.convertScaleAbs(img\_x)  # 取绝对值

    img\_absy = cv2.convertScaleAbs(img\_y)  # 取绝对值

    img\_sobel = cv2.addWeighted(img\_absx, 0.5, img\_absy, 0.5, 0)  # 合并x,y方向的结果

    return cv2.bitwise\_not(img\_sobel)  # 取反

## scharr算子

def scharr\_operator(img):

    #img\_gray = cv2.cvtColor(img, cv2.COLOR\_BGR2GRAY)  # 灰度化

    img\_x = cv2.Scharr(img, cv2.CV\_16S, 1, 0)  # x方向scharr算子

    img\_y = cv2.Scharr(img, cv2.CV\_16S, 0, 1)  # y方向scharr算子

    img\_absx = cv2.convertScaleAbs(img\_x)  # 取绝对值

    img\_absy = cv2.convertScaleAbs(img\_y)  # 取绝对值

    img\_scharr = cv2.addWeighted(img\_absx, 0.5, img\_absy, 0.5, 0)  # 合并x,y方向的结果

    return img\_scharr

## laplacian算子

def laplacian\_operator(img, ksize=3):

    #img\_gray = cv2.cvtColor(img, cv2.COLOR\_BGR2GRAY)  # 灰度化

    img\_laplacian = cv2.Laplacian(img, cv2.CV\_16S, ksize=ksize)  # laplacian算子

    img\_abs = cv2.convertScaleAbs(img\_laplacian)  # 取绝对值

    return cv2.bitwise\_not(img\_abs)  # 取反

##对比度拉伸

def contrast\_stretch(img):

    p2, p98 = np.percentile(img, (2, 98))  # 计算图像的2%和98%分位数

    img\_rescale = cv2.convertScaleAbs(img, alpha=255.0/(p98-p2), beta=-p2\*255.0/(p98-p2))  # 对比度拉伸

    img\_rescale = img\_rescale.astype(np.uint8)  # 转换数据类型

    return img\_rescale

## 自适应直方图均衡化

def adapthist\_equalization(img):

    clahe = cv2.createCLAHE(clipLimit=2.0, tileGridSize=(8,8))  # 创建自适应直方图均衡化对象

    img\_equalized = clahe.apply(img)  # 自适应直方图均衡化

    return img\_equalized

def convert\_value(x, a, b, c):

    if x < c:

        return a

    else:

        return b

def bianli\_find\_min\_array(img,A,A\_0):

    min\_norm = float('inf')

    norm = float('inf')

    min\_array = None

    min\_a = 0

    min\_b = 255

    #初值

    ret3, th3 = cv2.threshold(img, 0, 255, cv2.THRESH\_BINARY + cv2.THRESH\_OTSU)

    median = int(ret3)

    #min\_a, min\_b, median = 0, 255, np.median(A)

    min\_norm = float('inf')

    step = 80

    print(min\_a, min\_b, median)

    for a in range(int(0), min(median,int(median+step)), 10):

        for b in range(max(median,int(median-step)), int(255), 10):

            converted\_A = np.vectorize(convert\_value)(A, a, b, median)

            norm = np.linalg.norm(converted\_A - A\_0)

            if norm < min\_norm:

                min\_norm = norm

                min\_array = converted\_A

                min\_a, min\_b = a, b

    for c in range(max(min\_a,int(median-step)), min(min\_b,int(median+step)), max(1,int(step/8))):

            converted\_A = np.vectorize(convert\_value)(A, min\_a, min\_b, c)

            norm = np.linalg.norm(converted\_A - A\_0)

            if norm < min\_norm:

                min\_norm = norm

                min\_array = converted\_A

                median = c

    print(min\_a, min\_b, median)

    while step > 0:

        for a in range(int(min\_a-step), min(median,int(median+step)), max(1,int(step/5))):

            for b in range(max(median,int(min\_b-step)), int(min\_b+step), max(1,int(step/5))):

                converted\_A = np.vectorize(convert\_value)(A, a, b, median)

                norm = np.linalg.norm(converted\_A - A\_0)

                if norm < min\_norm:

                    min\_norm = norm

                    min\_array = converted\_A

                    min\_a, min\_b = a, b

        for c in range(max(min\_a,int(median-step)), min(min\_b,int(median+step)), max(1,int(step/10))):

            converted\_A = np.vectorize(convert\_value)(A, min\_a, min\_b, c)

            norm = np.linalg.norm(converted\_A - A\_0)

            if norm < min\_norm:

                min\_norm = norm

                min\_array = converted\_A

                median = c

        print(min\_a, min\_b, median)

        # 如果当前norm和之前的norm相差小于1，则退出循环

        step //= 2

    for a in range(int(min\_a-3), int(min\_a+3), 1):

        for c in range(int(median-3), int(median+3), 1):

            for b in range(int(min\_b-3), int(min\_b+3), 1):

                converted\_A = np.vectorize(convert\_value)(A, a, b, c)

                norm = np.linalg.norm(converted\_A - A\_0)

                if norm < min\_norm:

                    min\_norm = norm

                    min\_array = converted\_A

                    min\_a, min\_b, median = a, b, c

    print(min\_a, min\_b, median)

    return min\_array,min\_a,min\_b,median,min\_norm

def transfer(img):

    img\_con\_stretch = contrast\_stretch(img)  # 对比度拉伸

    img\_adp\_equalized = adapthist\_equalization(img)  # 自适应直方图均衡化

    img\_sobel\_adp = sobel\_operator(img\_adp\_equalized)  # sobel算子

    img\_scharr\_adp = cv2.bitwise\_not(scharr\_operator(img\_adp\_equalized))  # scharr算子

    img\_laplacian\_adp = laplacian\_operator(img\_adp\_equalized)  # laplacian算子

    img\_sobel\_con = sobel\_operator(img\_con\_stretch)  # sobel算子

    img\_scharr\_con = cv2.bitwise\_not(scharr\_operator(img\_con\_stretch))  # scharr算子

    img\_laplacian\_con = laplacian\_operator(img\_con\_stretch)  # laplacian算子

    img\_lunkuo = cv2.addWeighted(img\_sobel\_adp, 0.5, img\_scharr\_adp, 0.5, 0)  # 融合sobel和laplacian算子

    img\_jieguo = cv2.addWeighted(img\_lunkuo, 0.2, img\_adp\_equalized, 0.8, 0)  # 融合融合后的图像和对比度拉伸后的图像

    # 显示结果

    #cv2.imshow('Original', img)

    #cv2.imshow('lunkuo', img\_lunkuo)

    #cv2.imshow('laplacian\_adp', img\_laplacian\_adp)

    #cv2.imshow('sobel\_adp', img\_sobel\_adp)

    #cv2.imshow('scharr\_adp', img\_scharr\_adp)

    #cv2.imshow('laplacian\_con', img\_laplacian\_con)

    #cv2.imshow('sobel\_con', img\_sobel\_con)

    #cv2.imshow('scharr\_con', img\_scharr\_con)

    #cv2.imshow('jieguo', img\_jieguo)

    cv2.waitKey(0)

    return img\_jieguo

# Read image

#img\_clr = cv2.imread('lena.png')

img\_clr = cv2.imread('tian.jpg')

img\_clr =  cv2.resize(img\_clr, (int(img\_clr.shape[1]\*0.25),int(img\_clr.shape[0]\*0.25)),interpolation=cv2.INTER\_CUBIC)

#img\_clr = cv2.imread('d.png')

#img\_clr =  cv2.resize(img\_clr, (int(img\_clr.shape[1]\*0.25),int(img\_clr.shape[0]\*0.25)),interpolation=cv2.INTER\_CUBIC)

#通道分割

b\_0,g\_0,r\_0=cv2.split(img\_clr)    #b = img[:, :, 0]

b = transfer(b\_0)

g = transfer(g\_0)

r = transfer(r\_0)

print("\n开始遍历b\_0")

min\_b,min\_b\_a,min\_b\_b,min\_b\_c,min\_b\_norm = bianli\_find\_min\_array(b\_0,b\_0,b\_0)

print("\n开始遍历g\_0")

min\_g,min\_g\_a,min\_g\_b,min\_g\_c,min\_g\_norm = bianli\_find\_min\_array(g\_0,g\_0,g\_0)

print("\n开始遍历r\_0")

min\_r,min\_r\_a,min\_r\_b,min\_r\_c,min\_r\_norm = bianli\_find\_min\_array(r\_0,r\_0,r\_0)

print('\n开始遍历b')

min\_b\_trans,min\_b\_a\_trans,min\_b\_b\_trans,min\_b\_c\_trans,min\_b\_norm\_trans = bianli\_find\_min\_array(b,b,b\_0)

print('\n开始遍历g')

min\_g\_trans,min\_g\_a\_trans,min\_g\_b\_trans,min\_g\_c\_trans,min\_g\_norm\_trans = bianli\_find\_min\_array(g,g,g\_0)

print('\n开始遍历r')

min\_r\_trans,min\_r\_a\_trans,min\_r\_b\_trans,min\_r\_c\_trans,min\_r\_norm\_trans = bianli\_find\_min\_array(r,r,r\_0)

min\_b = np.array(min\_b, dtype=np.uint8)

min\_g = np.array(min\_g, dtype=np.uint8)

min\_r = np.array(min\_r, dtype=np.uint8)

min\_b\_trans = np.array(min\_b\_trans, dtype=np.uint8)

min\_g\_trans = np.array(min\_g\_trans, dtype=np.uint8)

min\_r\_trans = np.array(min\_r\_trans, dtype=np.uint8)

min\_array\_rgb = cv2.merge((min\_b,min\_g,min\_r))

min\_array\_rgb\_trans = cv2.merge((min\_b\_trans,min\_g\_trans,min\_r\_trans))

print('\n结束遍历')

#min\_array,norm = bianli\_find\_min\_array(A)

#min\_array = np.array(min\_array, dtype=np.uint8)

#print(norm)

cv2.imshow('Original', img\_clr)

#cv2.imshow('Min Array', min\_array)

cv2.imshow('Min Array RGB', min\_array\_rgb)

cv2.imshow('Min Array RGB Trans', min\_array\_rgb\_trans)

""" cv2.imshow('min\_array r', min\_r)

cv2.imshow('min\_array g', min\_g)

cv2.imshow('min\_array b', min\_b) """

cv2.waitKey(0)

color = ['b','g','r']

images = [img\_clr[:,:,[2,1,0]], 0, min\_array\_rgb[:,:,[2,1,0]], 0, min\_array\_rgb\_trans[:,:,[2,1,0]], 0]

# 绘制图像

titles = ['Original', 'Histogram',

          'Min RGB', 'Histogram',

          'Min RGB Trans', 'Histogram']

medians = [min\_b\_c, min\_g\_c, min\_r\_c]

for i in range(3):

    # 绘制原图

    plt.subplot(3, 2, i \* 2 + 1)

    plt.imshow(images[i \* 2], 'gray')

    plt.title(titles[i \* 2], fontsize=8)

    plt.xticks([]), plt.yticks([])

    """ # 绘制直方图 plt.hist,ravel 函数将数组降成一维

    plt.subplot(3, 3, i \* 3 + 2)

    plt.hist(images[i \* 3].ravel(), 256)

    plt.title(titles[i \* 3 + 1], fontsize=8)

    plt.xticks([]), plt.yticks([]) """

    plt.subplot(3, 2, i \* 2 + 2)

    for j,col in enumerate(reversed(color)):

        histr = cv2.calcHist([images[i\*2]],[j],None,[256],[0,256])

        plt.plot(histr,color = col)

        plt.axvline(medians[j],color = color[j],linestyle='--')

        plt.xlim([0,256])

    plt.title(titles[i \* 2 + 1], fontsize=8)

plt.show()

""" for i,col in enumerate(color):

    histr = cv2.calcHist([img\_clr],[i],None,[256],[0,256])

    plt.plot(histr,color = col)

    plt.xlim([0,256])

plt.show()

for i,col in enumerate(color):

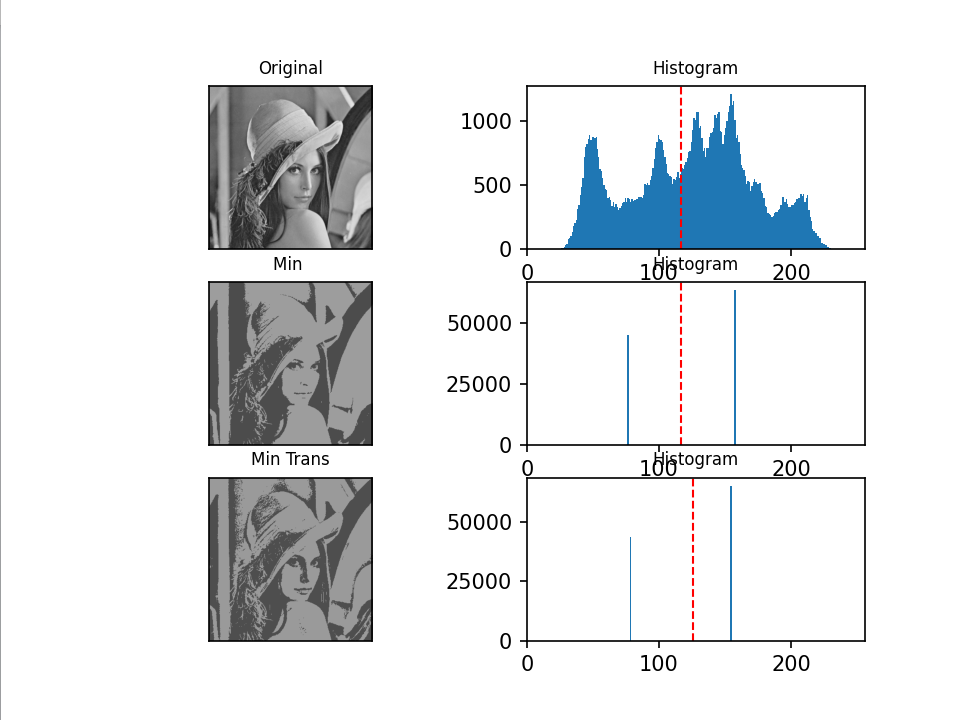
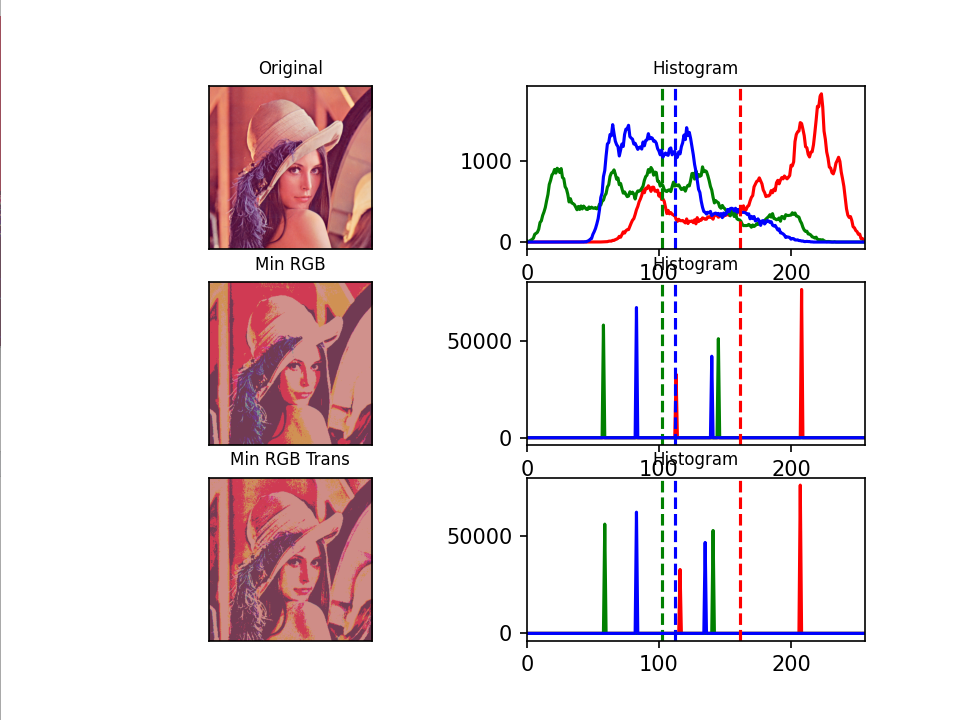
    histr = cv2.calcHist([min\_array\_rgb],[i],None,[256],[0,256])

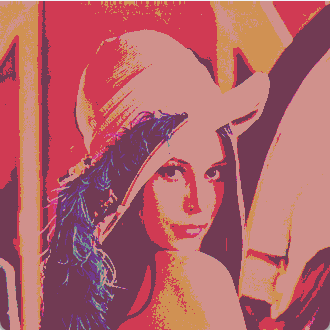
    plt.plot(histr,color = col)

    plt.xlim([0,256])

plt.show() """

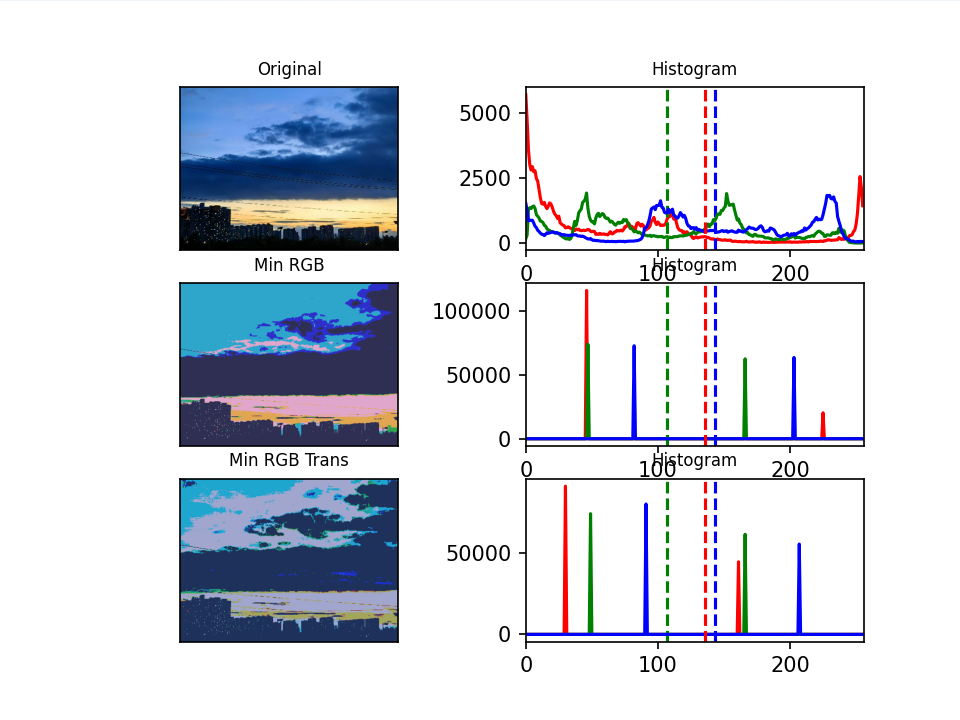
1. 结果呈现与对比；

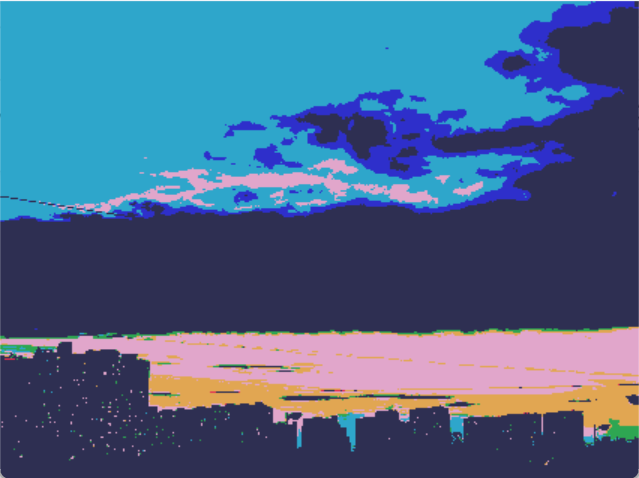


（左上为原图，右边为遍历最小结果，下边为自己设置的函数结果）



（左上为原图，右边为遍历最小结果，下边为自己设置的函数结果）







（左上为原图，右边为遍历最小结果，下边为自己设置的函数结果）

1. 课题总结：
2. 课题目标的实现情况；（问题是否解决？效果如何？有哪些结论？）

问题基本得到解决，输出的二值图像较遍历结果来说更加清晰，对人类的辨识度更高，轮廓教之更加清晰，

相比于机器来说，人判断图像是否相似并不是依靠公式计算的结果是否相近，而更多是依赖于，轮廓线是否清晰，色块之间的对比是否强烈

1. 课题成果与收获；（通过该课题的研究，实现了哪些成果？训练了哪些方面的能力？有哪些收获？）

实现了图像的二值处理，使得结果相较于遍历求解的结果更加符合人类辨识图像的习惯，应用二值化的方法，可以实现很可观的图像压缩技术，大大节约了图像保存所占用的内存

训练了快速学习的能力：锻炼了如何在短时间内快速学习一些知识并加以应用的能力。锻炼了代码能力：与最开始相比，可以更快更清晰的向计算机表达自己的需求，更少出错，提高效率。提升了计算相关的思维能力：了解到了很多计算相关的，降低时间复杂度或者提高计算准确度的方法，在遇到问题时就知道可以从哪些方向下手。

1. 经验、问题、建议及其他需要说明的情况；（介绍课题研究中的难点和经验。课题研究是否达到预期目标，如未达到，请分析原因和可能的解决途径。后续进一步研究的建议和设想）

研究中的难点有：

1. 如何在降低三重for循环计算的时间复杂度时，尽可能保证结果的准确（可以先大致定位，后在定位结果附近遍历寻找）
2. 如何选取降低对比度的算法以及刻画轮廓的算法，使得复合图像二值化后的结果更加有辨识度且接近原图（提高对比度时，自适应直方图均衡化这一方法有很好的效果，并且从直方图可以看出，他并不会造成严重的色差）
3. 如何避免色差（在进行轮廓刻画的时候，由于轮廓线条为黑线加白色底色，在与原图叠加时可能产生色差，我们对自适应直方图均衡化后的图像使用两种轮廓刻画方法是由于，均衡化后，图像轮廓刻画的结果更接近与原灰度图的颜色，并且，两种方法一个线条清晰，只刻画主要线条，另一种更加敏感，可以对图像细碎的线条进行加强）

后续问题：

1. 由于Rgb图像在合成为彩色图像前我们比不清楚他会合成为什么颜色，我们只是对矩阵在进行操作，我知道我的算法可以将灰度图更清晰的表达出来，（由于对比度于轮廓实际上就是灰度图所可以表达的关于图像的信息了）但分别应用于rgb图像后再融合为彩色图像，并不一定会达到最好的近似结果，未来的研究可以集中于：如何构建一种算法，可以保证对rgb图像的处理后，融合为彩色图像时结果也可以达到很好