目录

**[一．设计对象 1](#_Toc26111)**

[1.1设计任务 1](#_Toc7039)

[1.2实现功能 1](#_Toc12042)

[1.3人员分工 1](#_Toc22194)

**[二．相关知识 1](#_Toc32143)**

[2.1半色调算法 1](#_Toc8556)

（1）抖动法 （颜宇康） 1

（2）误差扩散法 （张子琪） 2

[2.2相关python函数 （夏思俊） 3](#_Toc22182)

[2.3广度优先搜索 3](#_Toc22005)

[2.4 PyQt5库相关 4](#_Toc14022)

[2.5 canny算子进行边缘检测抠图 4](#_Toc1288)

**[三．源程序 5](#_Toc24915)**

[3.1 GUI界面（夏思俊，梁伟斌） 5](#_Toc27564)

[3.2 边缘检测抠图（王智豪） 10](#_Toc20903)

[3.3 颜色抠图（杨俊曦） 11](#_Toc10389)

**[四．实验结果 13](#_Toc345)**

**[五．心得体会 17](#_Toc13777)**

**[参考文献 18](#_Toc5585)**

**一．设计对象**

**1.1设计任务**

（1）图像的半色调技术：打印机打印图像时，只能是落墨或不落墨两种状态。灰度图像的打印需要利用图像的半色调技术完成。因此需要对一幅0～255灰度范围的灰度图像用半色调技术实现。

(2)实现类似Photoshop的抠图功能，需要有GUI。

**1.2实现功能**

针对（1），通过抖动法和误差扩散法两种方法实现半色调技术，并对比二者的不同。

针对（2），实现了对鼠标拖动选定区域的颜色抠图和边缘检测抠图。

**1.3人员分工**

夏思俊：统筹，收集资料，代码编写

杨俊曦：代码编写，颜色抠图

梁伟斌：界面GUI，收集资料

张子琪：半色调误差扩散法

王智豪：边缘检测抠图

颜宇康：半色调抖动法

**二．相关知识**

**2.1半色调算法**

半色调技术是 指的是用少量的色彩将一幅连续色调图像量化为一幅二值图像或只有少数几种色彩的图像，并使得量化过后，图像在一定距离的视觉效果和原始图像相似的技术。

半色调图像如常见的印刷品图像，其由浅到深或由淡到浓的变化，是靠网点面积大小或网点覆盖率来表现的。一般用于复制诸如照片之类的连续色调图像时，会采用半色调技术，它会将图像分成许多点，通过点的不同大小来表现深浅。

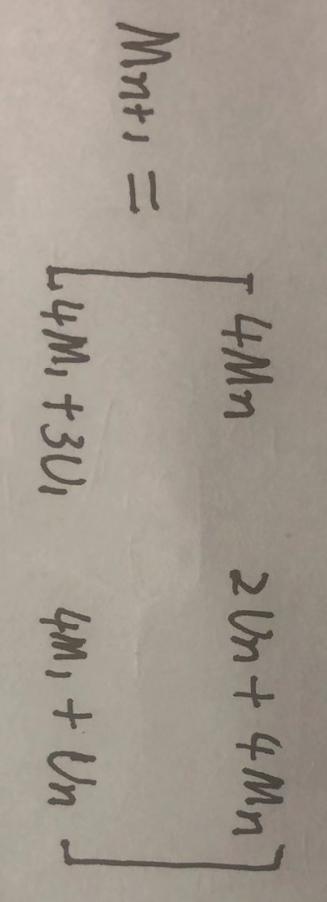
目前半色调技术最普遍的分类法是按照它的处理方式分为：**抖动法**，**误差扩散法**，**迭代法**三大类。

**（1）抖动法**

抖动法分为随机抖动和有序抖动，此处只介绍有序抖动当中的基于Bayer矩阵的有序抖动。抖动法都需要一个阈值矩阵，它决定了当灰度值减小时网点变成黑点的顺序，也因此决定半色调图像的质量。

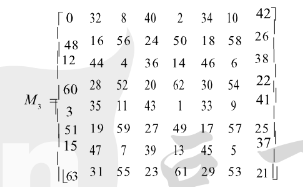
抖动法的原理即**将输入连续色调图像的像素点灰度值与与阈值矩阵当中对应的阈值进行比较，如果前者大于后者则半色调后的图像中的对应灰度值为1，否则为0，最后即得到半色调后的二值图像。**

阈值矩阵的构造是抖动法的关键，阈值矩阵当中每个阈值都在输入的连续色调图像的最大灰度值和最小灰度值之间。此处介绍用到的Bayer矩阵，它的构造原理如下：给定一个2×2的矩阵1623517341(1)，递推关系为



**图2-1**

其中Mn和Un都是2^n\*2^n的矩阵，Un的所有元素都是1，则M3，即8×8矩阵，即为Bayer有序抖动矩阵：



**图2-2**

**有序抖动简便而且输出的半色调图像质量良好，但是却有明显的周期性人工纹理，无论如何设计矩阵，都会存在此类问题。**

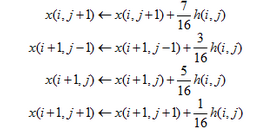
**（2）误差扩散法：**

误差扩散法的原理即对整幅连续色调图像的像素点，从左到右从上至下逐点执行两步操作：

1. 阈值化输出y（i，j）：将原图像的像素点进行量化，并计算其量化误差，其中输入为x（i，j），

输出y（i，j），量化误差h（i，j）。

1. 将量化误差扩散到邻近的未被处理的点，按7：3：5：1的比例转移并叠加到邻近的像素点：



**图2-3**

对每个像素点做如此处理之后即可得到对应的半色调后的图像。

误差扩散法相当于将中值阈值法在每个点上产生的误差再加到周围的点上，以此来保持局部区域总体灰度基本不变。

**误差扩散法处理后的图像像素分布各异而且无规律性，是目前比较简易且效果较好的半色调技术之一，被广泛使用。**

**2.2相关python函数**

（1）**canny算子**：输出图像的边缘。

1.**lowThreshold**：低阈值。

2.**ratio**：高低阈值比为3：1。

3.**kernel\_size**：内核尺寸为3。

（2）**GaussianBlur函数**：高斯滤波，一种线性平滑滤波，用于消除高斯噪声。

（3）**findcontours（）：**轮廓检测函数，输入必须为二值图像，否则需要先灰度化和二值化。**有四种检索模式，其中cv2.RETR\_EXTERNAL表示只检测外轮廓。有四种轮廓近似办法，其中 cv2.CHAIN\_APPROX\_NONE 存储所有的轮廓点，相邻的两个点的像素位置差不超过1。其返回值有两个，一个是轮廓本身contours，一个是每条轮廓对应属性。**

（4）**getStructuringElement（）：**返回指定形状和尺寸的结构元素，如矩形：MORPH\_RECT。

（5）**cv2.inRange（）：**中值阈值法，共有三个参数，第一个参数是原图，第二个和第三个参数对应下限灰度值和上限灰度值。

**2.3广度优先搜索**

广度优先搜索算法是最简便的搜索算法之一，它属于一种盲目搜寻法，目的是系统地展开并搜索图中的所有节点，从而搜索到结果，它不考虑结果的可能位置，而是彻底搜索直到找到为止。

本次设计在实现PS抠图的时候，由于需要寻找轮廓，则要用到广度优先搜索，其原理如下：**在寻找轮廓的时候，对于鼠标选定的需要寻找轮廓的区域进行中值阈值法运算，超过灰度上限值和低于灰度下限值的像素点都会被置为0，而在范围内的则会被置为1，得到了一个二值图像。由于在一个区域当中，一般都是灰度值比较相近的同一种颜色的像素点，而灰度值不相近的其他颜色即为轮廓，因此，只要对该区域进行一次广度优先搜索，在中值阈值法处理过后，原本为其他颜色的如今为0的像素点即为标记为轮廓，而原本颜色相近如今为1的像素点标记为轮廓内部，最后给轮廓标上不一样的颜色即完成抠图。**

**2.4 PyQt5库相关**

·QWidget：所有用户界面对象的基类；用于接收用户输入（如图像），并在界面上绘制自己。

·QPushButton:界面的按钮类，用于设置按钮事件。

·QHBoxLayout、QVBoxLayout：水平布局、垂直布局。用于组织各种界面元素（如图片、元素）布局。

·QApplication：用于设置界面风格。

·QLabel：提供一个文本或图像的显示，用于图像标签化。

·QLineEdit：一个单行文本编辑控件，使用者可以通过很多函数，输入和编辑单行文本，如撤销、恢复、粘贴、剪切、施放等。

·QPixmap、QImage：QPixmap用于绘制图片。QImage用于图片的像素级访问。

·QTimer：提供重复和单次触发的定时器：创建一个QTimer，链接timeout（）信号到适当的槽函数，并调用start，即可在恒定时间间隔发送timeout（）信号，以此来每隔一段时间执行以此界面的操作。用于定义每次执行函数操作之间的时间间隔。

**2.5 canny算子进行边缘检测抠图**

我们抠图的过程其实就是找图像中某个实例（例如人）的边缘的过程，所以，我们的边缘检测，其实就是检测图像中的实例的边缘。那现在就有一个问题了，我们人是怎么区分边缘的呢？我们发现，每一个实例，它的边缘跟其周围的像素差距一般是比较大的。我们的抠图，就是根据明显的像素差距，来区分实例的边缘。所以抠图的根据，就是图像像素的明显变化。图像的边缘，一般都是图像上像素有明显变化的位置。

本次设计采用canny算子，canny算子是一个多级边缘检测算法，它主要包括以下五个步骤：

①图像平滑：图像平滑就是让图像之间的像素差距更小一些，能够更好地提取特征更加明显的边缘，所以图像平滑的目的就是去除部分噪声。我们这里采用高斯滤波的方法。

②寻找图像强度梯度：Canny算法的基本思想是寻找一幅图像中灰度强度变化最强的位置。所谓变化最强，即指梯度方向。平滑后的图像中每个像素点的梯度可以由Sobel算子（一种卷积运算）来获得（opencv中有封装好的函数，可以求图像中每个像素点的n阶导数）。

③消除边误检：这一步的目的是将模糊的边界变得清晰（sharp）。通俗的讲，就是保留了每个像素点上梯度强度的极大值，而删掉其他的值。

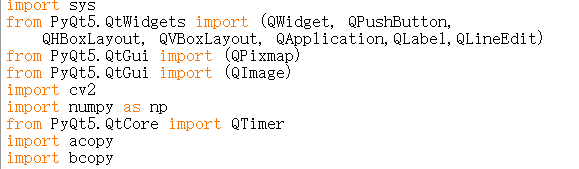
④双阈值求可能边：经过非极大抑制后图像中仍然有很多噪声点。Canny算法中应用了一种叫双阈值的技术。即设定一个阈值上界和阈值下界（opencv中通常由人为指定的），图像中的像素点如果大于阈值上界则认为必然是边界（称为强边界，strong edge），小于阈值下界则认为必然不是边界，两者之间的则认为是候选项（称为弱边界，weak edge），需进行进一步处理。

⑤边界跟踪：较高的亮度梯度比较有可能是边缘，但是没有一个确切的值来限定多大的亮度梯度是边缘多大又不是，所以 Canny 使用了滞后阈值。

滞后阈值（Hysteresis thresholding）需要两个阈值，即高阈值与低阈值。假设图像中的重要边缘都是连续的曲线，这样我们就可以跟踪给定曲线中模糊的部分，并且避免将没有组成曲线 的噪声像素当成边缘。所以我们从一个较大的阈值开始，这将标识出我们比较确信的真实边缘，使用前面导出的方向信息，我们从这些真正的边缘开始在图像中跟踪 整个的边缘。在跟踪的时候，我们使用一个较小的阈值，这样就可以跟踪曲线的模糊部分直到我们回到起点。

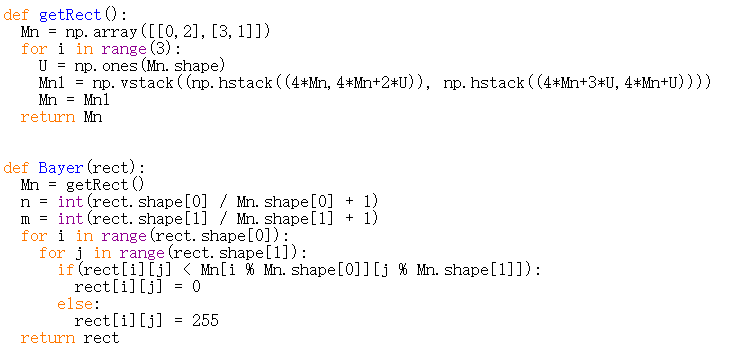
**三．源程序**

**3.1 GUI界面（test.py）**

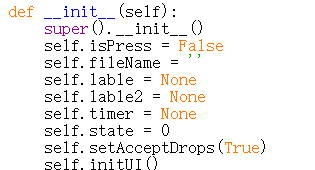
****

**图3-1**

如图3-1所示，包含相关的头文件以及调用其中的类，如PyQt5库中QtWidgets文件的五个类，QtGui文件中的两个类，以及QtCore文件当中的QTimer，它们的作用在相关知识中有详细介绍。此外还有numpy库当中的np类，还有cv2、sys等基础配置，以及包含两个写好的头文件acopy.py以及bcopy.py。

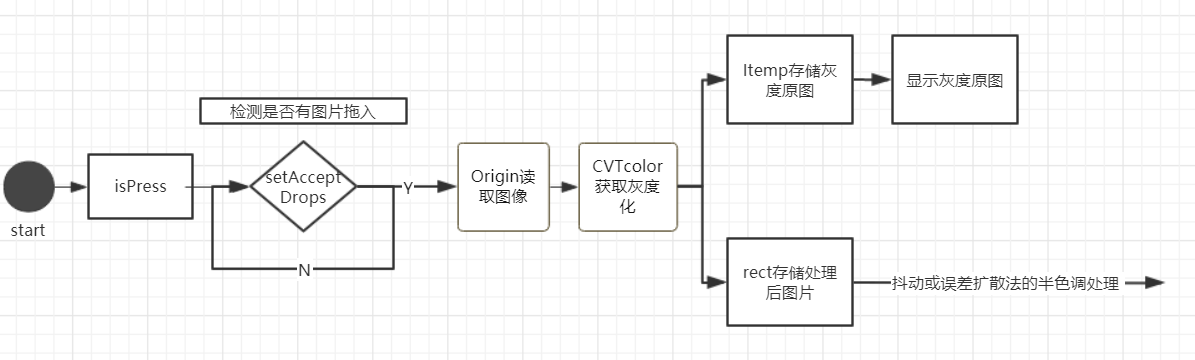
**图3-2**

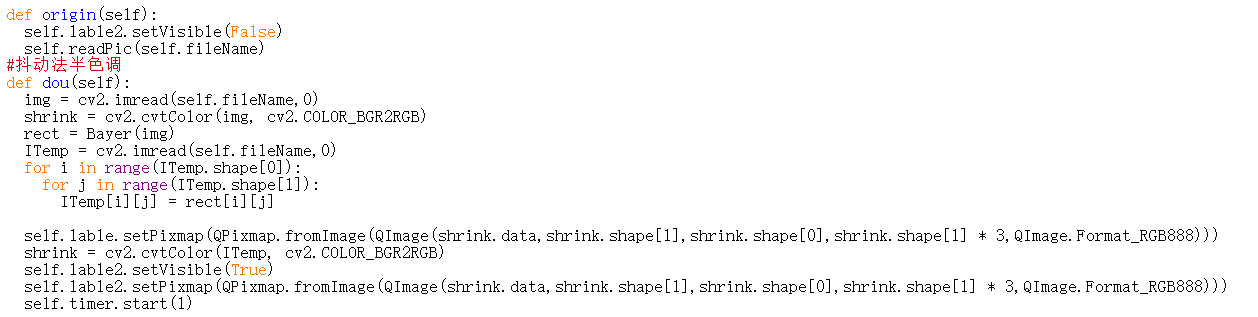
如图3-2所示，getRect函数用于生成Bayer有序抖动矩阵，其生成方法详见相关知识：语句中，Mn为初始的给定2维矩阵，Mn1为Mn+1的生成方式；本次实践中使用了2^3\*2^3矩阵，因此更新了3次，最后输出8\*8的阈值矩阵。Bayer函数用于有序抖动的比较并根据比较结果输出0或255，以此定义了一个有序抖动图像处理函数，每个像素点都与getRect函数生成的阈值矩阵比较，输出一个二值图像。

****

**图3-3**

如图3-3为初始化，isPress检测鼠标是否点击，setAcceptDrops则是启动接收拖拽图片，将标签、定时、状态全部置0，初始化UI后让GUI准备接收图片。读取图片流程图如下：

**图3-4**

**图3-5**

如图3-5,origin函数的功能是读取拖拽的图片，在上方label当中显示原图，且关闭下方label2的GUI组件显示。

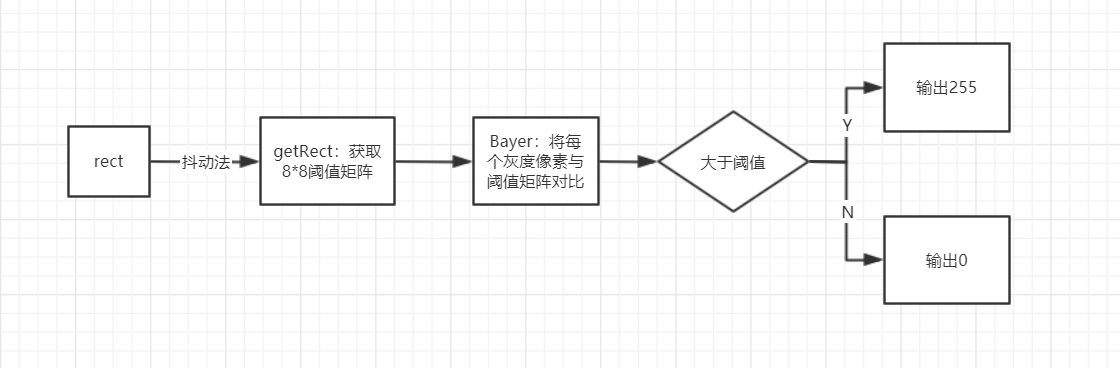
每进行一次操作，显示图片时：

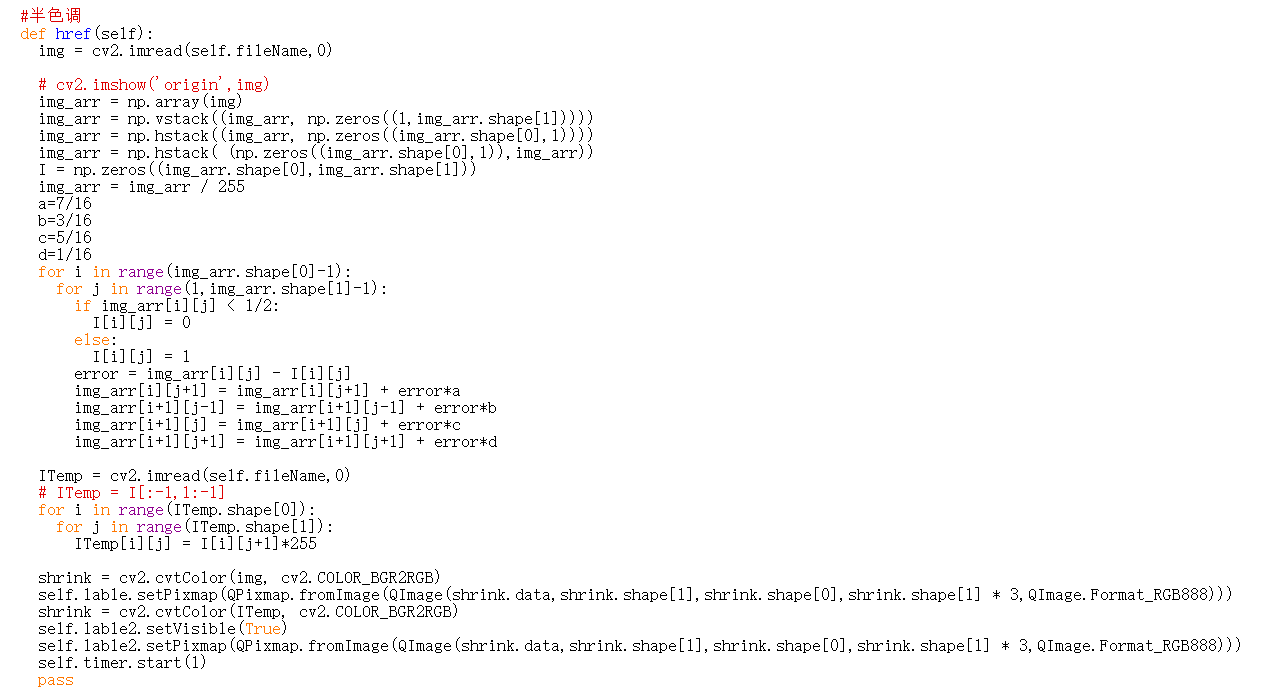
1.用setVisible函数决定是否开启对应域的组件显示。（此处label已在原图按钮事件中开启，所以无需再写）

2.用setpixmap函数显示在对应图片区域（label、label2）。显示之前需要用cvtColor函数进行颜色空间转换。

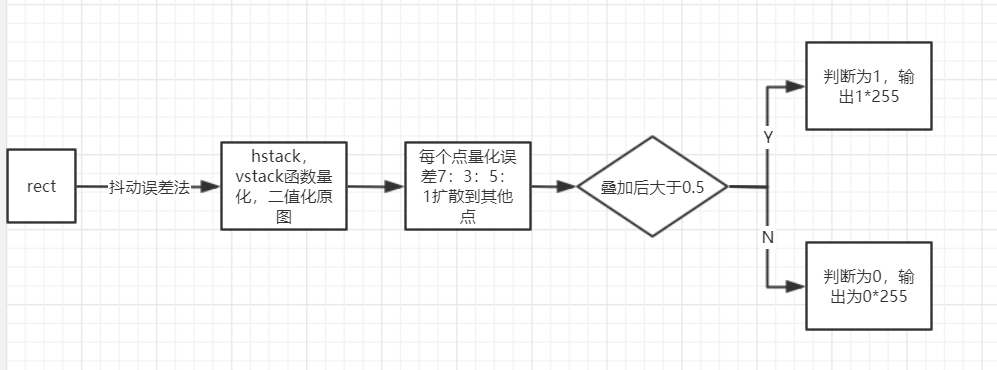
3.timer.start()写入每次操作的时间间隔。

dou（）函数功能即实现基于Bayer矩阵的有序抖动半色调处理图像，先将原图用cvtcolor函数转换格式，并分别用Itemp和rect进行记录，之后对rect调用Bayer（）函数进行有序抖动半色调处理，然后在GUI上方的label中显示Itemp作为原图，在下方label2当中显示处理过后的rect的半色调化图像。显示时label2的组件要用SetVisible开启显示，图像都用setPixmap函数进行显示。抖动法流程图如下所示：

**图3-6**

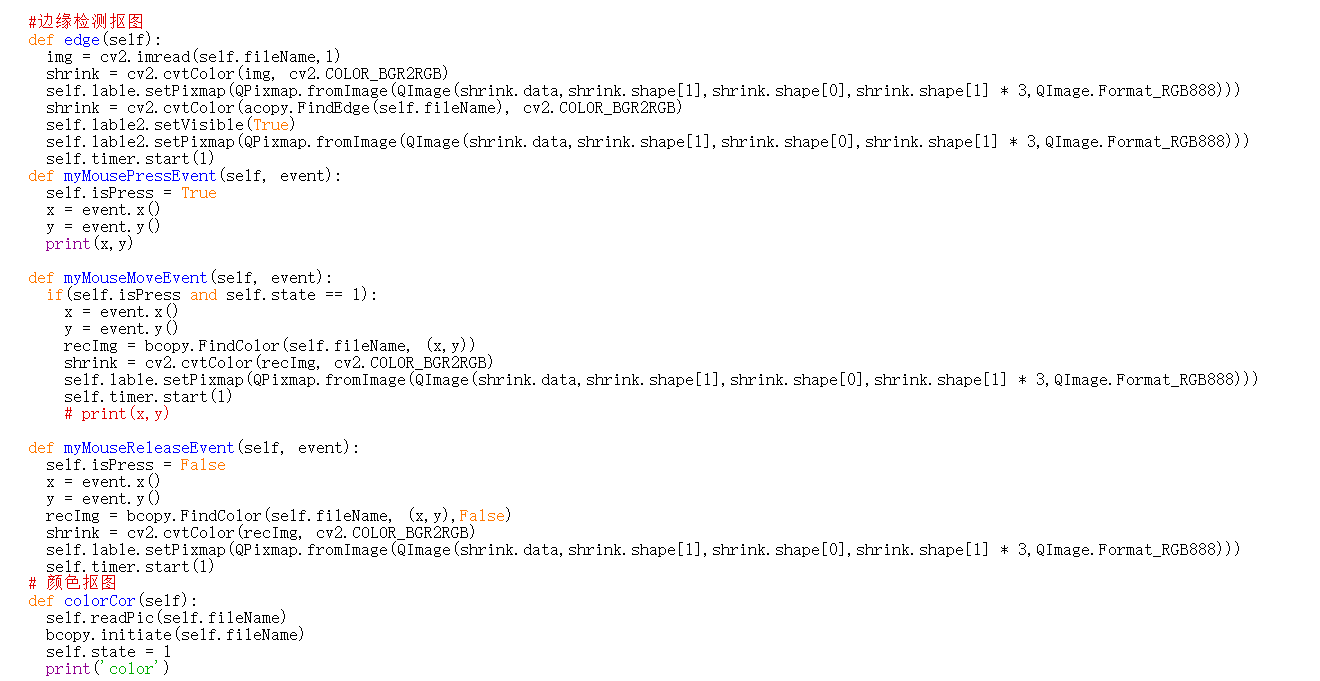
****

**图3-7**



**图3-8**

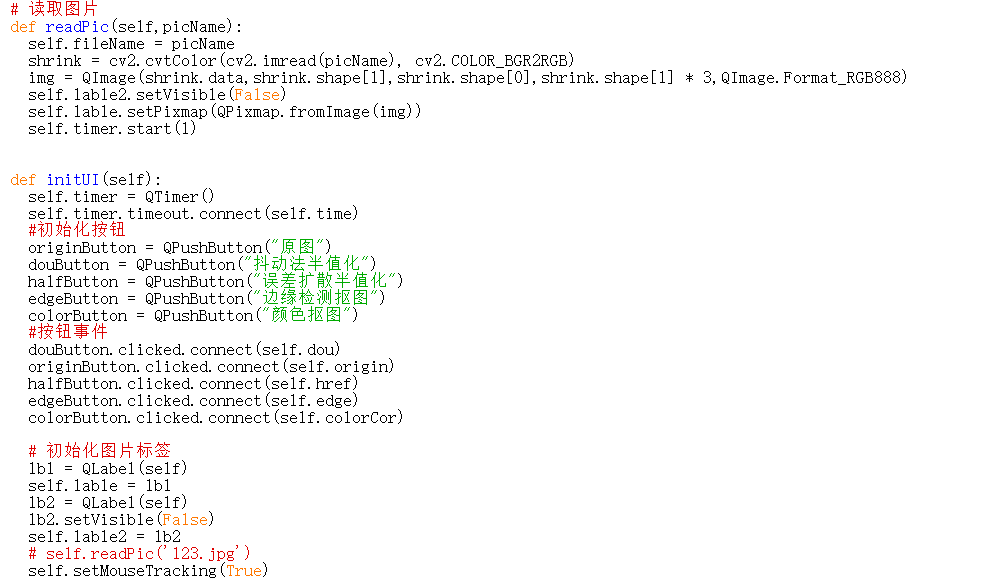
如图3-7所示，href函数用于误差扩散法的半色调处理图像，首先先用hstack和vstack函数填充输入进来的原图，然后将原图的每个像素点进行量化，即除以255，如果小于0.5的直接置0，否则置1，如此得到了一幅二值化图像。之后对该二值化图像进行误差扩散法的半色调处理，即每个点的量化误差按7：5：3：1的比例扩散到其他点，然后再进行叠加。最后再让每个像素点重新乘255，用Itemp记录，如此则得到半色调处理后的图像。最后的四行为显示方法，同dou（）函数。抖动误差法流程图如下所示：

**图3-9**

如图3-9为边缘检测抠图与颜色抠图部分，edge函数为边缘检测抠图的初始化部分，即上方显示原图而下方显示原图的边缘，用到了acopy文件中的寻找边缘函数FindEdge（），显示方法同上。之后的三个函数myMousePressEvent（）、myMouseMoveEvent（）、myMouseReleaseEvent（）分别检测图像的点击、拖动和释放，用于确定需要抠图的区域并在选定区域后对该区域调用bcopy文件当中的Findcolor函数进行寻找轮廓，并显示在原图上。

colorCor（）函数实现的功能时颜色抠图，它是先调用bcopy当中的initiate函数进行初始化并和边缘抠图一样在选定区域的寻找轮廓并显示在原图上。

值得一提的是，这几个函数都用了QTimer进行定时检测，让它们每隔一段时间就检测以上的行为是否发生，以此来执行相关操作。

****

**图3-10**

如图3-10所示，这几个函数功能一目了然。readPic函数为之前多次调用的读取函数，用于读取拖拽进入的图像。initUI函数用于初始化UI界面，并且定义按钮、检测是否发生按钮事件（是否被点击按钮），以及初始化上下两个图像的标签label、label2，并用setMouseTracking函数检测开启检测鼠标。

****

**图3-11**

如图3-11所示，承接上面的初始化函数，之后开启检测鼠标的点击、移动、释放，以此检测显示区域，另外还有相关的按钮布局、图片布局、整体布局，按钮和图片用到的是垂直布局QVBoxLayout，而整体布局用到的是水平布局QHBoxLayout，之后设置窗口信息。最后两个函数用于打开图片拖拽、并接收拖拽进入的图片。

**3.2 边缘检测抠图（acopy.py）**

**图3-12**

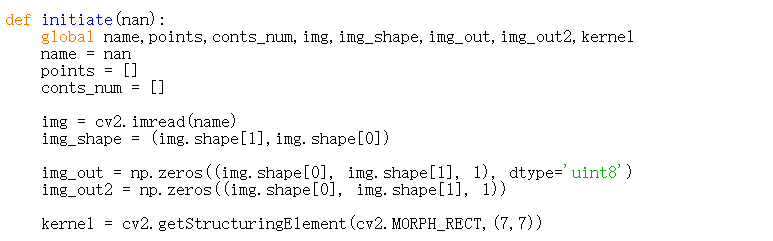
这一文件存放边缘检测抠图部分的代码。

如图3-12所示，这是acopy文件当中的FindEdge函数，用于寻找边缘，进行边缘检测抠图的第一步，用到的是canny算子，先设置低阈值为60，高低阈值比为3：1，内核尺寸为3。

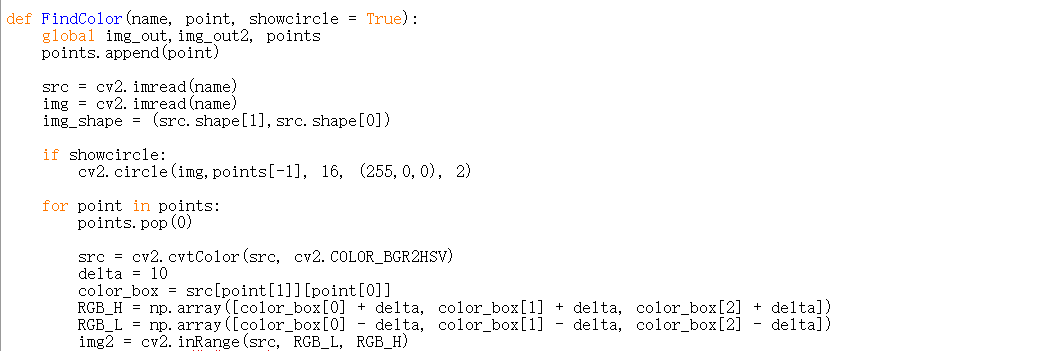
之后读取图片放在img、img2中将img转换格式放入gray当中，然后对gray进行高斯滤波放在detected\_Edges当中，之后对该图片进行canny算子检测边缘，并用findContour函数寻找边缘，检测外轮廓并存储所有点，记录在contours与hierarchy当中，当边缘超过5条，说明找到的边缘足够，则停止。

将用边缘检测处理过的图像canny\_img记录在detected\_Edges当中，将其检测区域用getStructuringElement函数定义为矩形，对detected\_Edges再用findContour函数用morphologyEx函数进行形态学滤波，然后再用FincContour函数对其进行寻找边缘，检测外轮廓但只存储矩形区域的元素，并记录再contours当中，之后在原图img2上画出边缘，如此则实现了边缘检测抠图。

**3.3 颜色抠图（bcopy.py）**

**图3-13**

如图3-13所示，该函数即颜色抠图的初始化函数，对进入的图片进行填充，并用getStructuringElement函数定义kernel，即检测区域为矩形。

****

**图3-14**

****

**图3-15**

如图3-14、3-15，即寻找轮廓函数FindColor（）。其中，图3.11功能如下：读取图像放在img、img2，用circle函数定义画出来的轮廓有多粗，并定义画出来的轮廓形状为圆形。在图像转化完成后对图像的每个像素点进行一个加delta、减delta，以此定义上限阈值和下限阈值RGB\_H与RGB\_L，并以此调用inRange函数进行中值阈值法，即灰度值在阈值中间的像素点为1，否在为0.

图3-15则是利用广度优先搜索算法，寻找出为0的像素点并标记为轮廓，最后以此画出轮廓完成抠图，此处利用广度优先搜索算法的原理见相关知识。

While pos\_list ! = []此处，是程序的核心，先创建一副黑色图像。然后对遍历列表里面的每个元素进行遍历，遍历的方式是：

判断列表中某个点P0在二值化图片中的像素是1还是0，假如是1，说明这个点在阈值范围内，于是将黑白图像中该点置1。然后以P0为中心，将四周的点加入遍历列表中（此处加入便利列表时附带条件，需要确定新加的点没有被遍历过且在图像范围内）假如是0，说明这个点不在阈值范围内，因此跳过并进行下一个点的遍历。

最后在遍历结束后，我们得到一张和鼠标点击坐标颜色相同并联通的图片，那么我们将该图片进行canny边缘检测，就得到所需要的边界了。最后将边界在图像中画出，返回到GUI程序中。

在实现过程中我遇到了一点问题：

运算速度实在太慢了，因此我考虑了我算法上的问题，并提出了解决方案

原来的方案是：从鼠标点击的点开始，向四周遍历，每次遍历一个点就计算这个点的RGB值与标准RGB值之间的差距，若差距在某个范围内，就确定该点和我们期望的颜色相近，在边缘范围内。

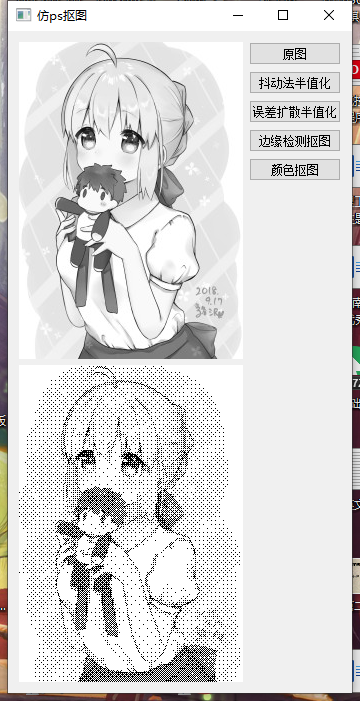
但是这个方法对每个像素点都要计算距离，在分辨率低的时候还好，分辨率高了计算量就巨大。因此用了第二种方法。

现在的方案是：先用inrange函数对整张图片进行某个范围内的二值化，，之后对每个像素点就判断该点在二值化图像中是0或者1来判断是否在范围内。遍历的过程中仅仅有一个判断，运算速度大幅提升。

**四．实验结果**

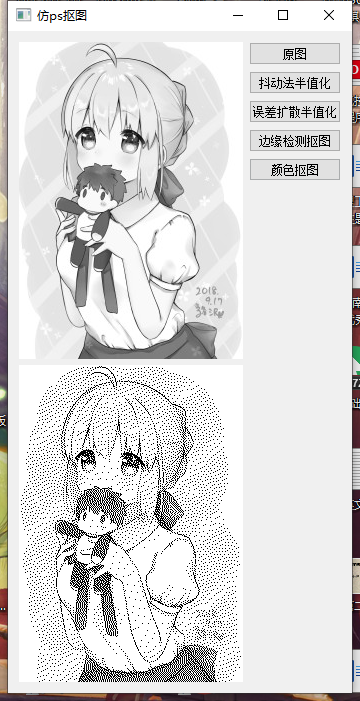
**4.1实验结果分析**

抖动法半值化实验结果:

 ****

**图4-1 图4-2**

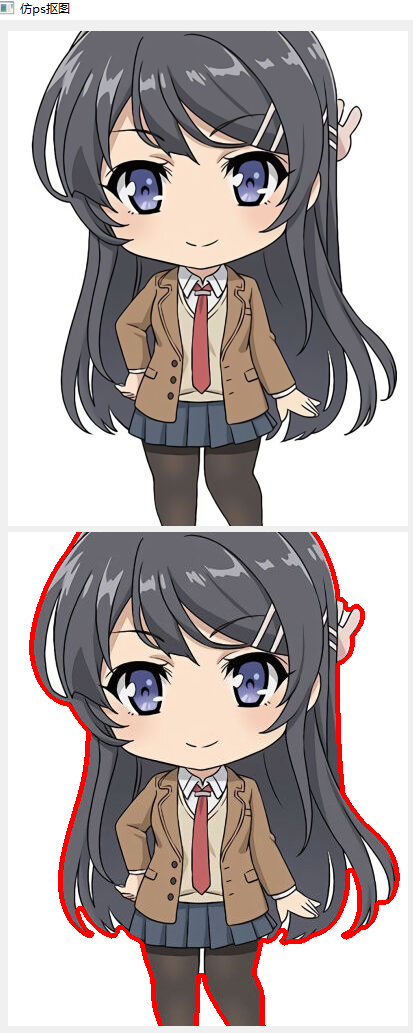
误差扩散半值化实验结果:



**图4-3 图4-4**

经对比两种半值化效果演示图，误差扩散半值化的效果较好，并且相较于抖动法半值化像素分布无规律，而有序抖动简便而且输出的半色调图像质量良好，但是却由明显的周期性人工纹理。

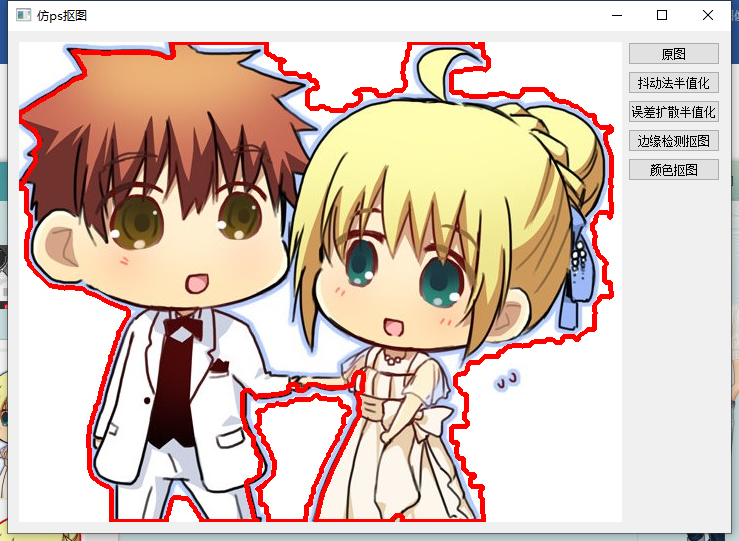
边缘检测抠图效果图:



**图4-5 图4-6**

边缘检测抠图能大致扣除目标图片中的人物图，但相对复杂的图片所扣出的人物，可能会扣出人物的一部分，比如眼睛鼻子头发等。

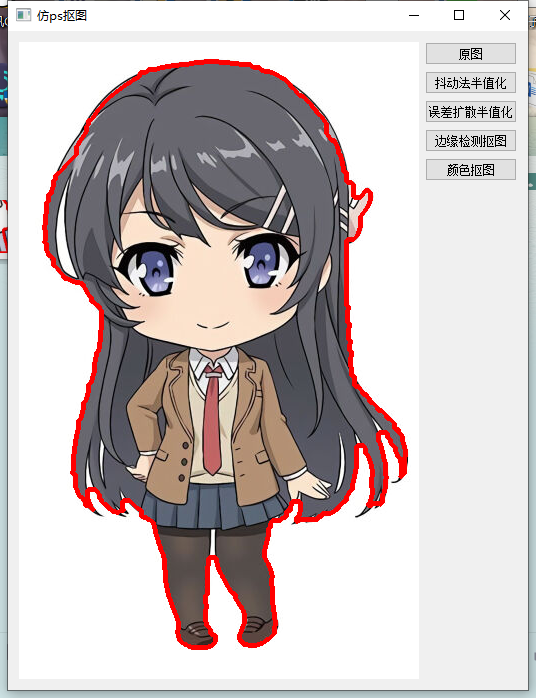
颜色抠图效果图：



**图4-10**

****

**图4-11**



**图4-12**

颜色抠图可以大致扣出图片中的所需扣出的部分，如实验图片中的动漫人物。

**五．心得体会**

本次课程设计在设计的过程当中遇到了很多的困难，这是由于我们小组里面几个人都是初次接触这些算法和函数，甚至是初次接触python的。在设计的过程当中，我们一个一个查找函数功能，查找了不少文献，大家都找了不少关于半色调算法和实现抠图的算法，最后才决定选用基于Bayer矩阵的有序抖动、误差扩散法进行半色调，并用canny算子和广度优先搜索算法进行抠图，可以说是十分的不容易了。

本次课程设计我们用的时间还是不少的，虽然设计过程当中十分的艰难，并且做出来的效果有些差强人意，但是我们小组几个人在这设计的过程当中学到了很多，包括python的相关基础知识以及一些基础的算法步骤，可以说获益良多，总体效果也还算可以，还是比较令人满意的。

参考文献

［1］冈萨雷斯 编著．《数字图像处理（第三版）》.北京.电子工业出版社，2003.

［2］半色调技术

https://blog.csdn.net/songzitea/article/details/40832565

［3］一种新的基于Bayer矩阵抖动的灰度图像密码学方法https://www.docin.com/p-1446417455.html