# 数字图像处理和模式识别综合实验报告

## 课题名称：豆子分类计数

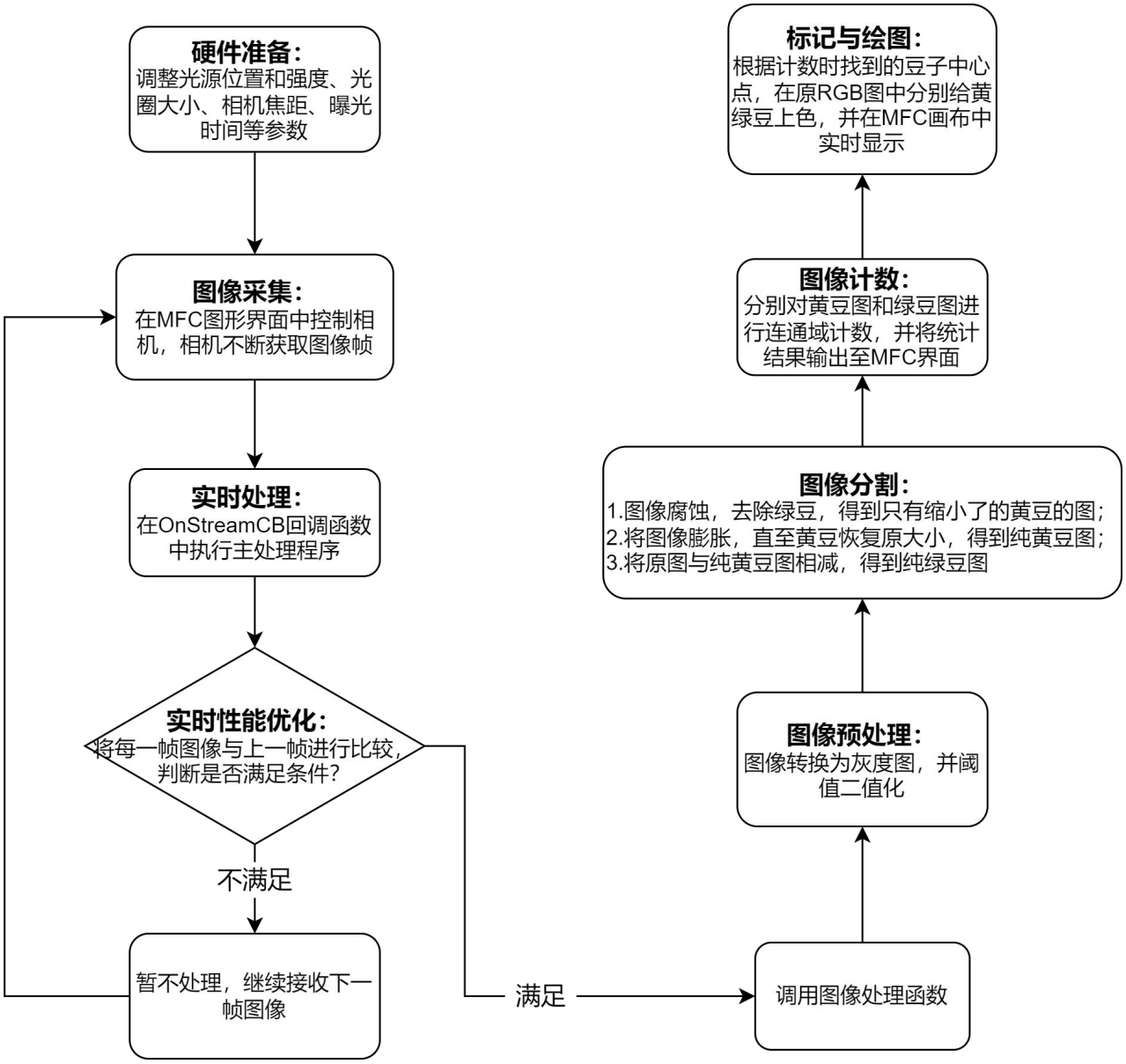

院　　系：　人工智能与自动化
班　　级：　自动化 1905
姓　　名：　钱力晖、郑义泽
学　　号：U201914690、U201914701
指导老师：　马 杰、郑定富



自动化学院
2022 年 3 月 30 日

# 目　　录

# 一、总体方案设计


硬件准备：
调整光源位置和强度、光圈大小、相机焦距、曝光时间等参数
图像采集：
在MFC图形界面中控制相机，相机不断获取图像帧
实时处理：
在OnStreamCB回调函数中执行主处理程序
实时性能优化：
将每一帧图像与上一帧进行比较，判断是否满足条件?
不满足
暂不处理，继续接收下一帧图像
满足
调用图像处理函数
图像预处理：
图像转换为灰度图，并阈值二值化
图像分割：
1.图像腐蚀，去除绿豆，得到只有缩小了的黄豆的图；
2.将图像膨胀，直至黄豆恢复原大小，得到纯黄豆图；
3.将原图与纯黄豆图相减，得到纯绿豆图
图像计数：
分别对黄豆图和绿豆图进行连通域计数，并将统计结果输出至MFC界面
标记与绘图：
根据计数时找到的豆子中心点，在原RGB图中分别给黄绿豆上色，并在MFC画布中实时显示

## 二、关键技术

### 1.腐蚀膨胀算法：

腐蚀算法的核心思想是，逐个像素点遍历二值化后的灰度图（豆子为灰色，灰度值为 128；背景为黑色，灰度值为 0）。若遍历到豆子点，则检测其邻域是否有背景点，若有则认为该豆子点为边界点，立刻将其置为背景黑色点；膨胀算法原理与之相同，只是将背景和豆子的判断反了过来，相当于是“反向的腐蚀”。

### 2.图像分割算法：

主要由四部分完成：

一是将阈值二值化后得到的灰度图像腐蚀一定次数，直至完全去除绿豆，得到只有缩小了的黄豆的图像；

二是将图像膨胀，直至黄豆恢复原大小，得到纯黄豆图；

三是将原图与纯黄豆图相减，得到纯绿豆图。其中相减算法的原理为同时遍历原图和纯黄豆图的对应像素点，若其灰度值之和刚好等于 128（只有绿豆点满足），则将绿豆图像内存区的相应点的灰度值设为 128，其余皆设为 0，由此得到纯绿豆图；

四是再进行一定次数的腐蚀，以去除相减后的残余边界和粘连。

### 3.连通域计数算法：

利用假想队列对连通域进行宽度优先搜索(BFS)，从而统计连通域中每个点的坐标信息，进而可算出中心点的坐标，中心点的个数即豆子个数。

### 4.（创新点）圆域涂色标记算法：

为了标记找到的豆子，常规思路是调用 MFC 画图类方法去给豆子画圆，或只是简单地给豆子中心做个标记。这两种方法皆不佳，调用 MFC 画图类方法会涉及真实坐标和画布坐标转换的问题，且会曾额外的性能消耗，不利于相机实时处理；简单地给豆子中心做个标记也会使分类结果不够清晰。

为此，我们编写了一种直接给豆子圆域涂色标记的算法，该算法基于相机原 RGB 图，能经过少数遍历就将豆子圆域上色标记，性能消耗低，结果清晰易懂。

具体思路和结果详见第四部分的源码分析。

### 5.（创新点）实时优化算法：

在 OnStreamCB 回调函数中先记录相机帧数并判断此帧图像与上一帧图像的差异，只有满足一定条件时才认为图像发生变化、需要处理。通过该方法的优化，相机实时处理性能和鲁棒性有了极大提升，在处理少量豆子、大量豆子以及外界干扰（如用手加豆子或移动光源）时都能保持较高的流畅度和较好的稳定性。

### 6.（创新点）四邻域八邻域交替检测算法：

我们在腐蚀和膨胀算法中，创新性地使用四邻域和八邻域按比例交替检测的算法，从而使腐蚀和膨胀后的豆子与原形状十分接近，有效解决了豆子分割不完全、粘连严重等问题。

具体思路和结果详见第四部分的源码分析。

## 三、图像采集系统设计

**任务分析：**

本次程序设计的目的是基于机器视觉方法，实现对平铺的黄豆、绿豆分别计数；要解决的关键问题和难点包括图像采集、区分黄豆和绿豆、正确计数、解决大量豆子紧密排列时的粘连问题以及保证程序实时性；

**文献阅读与资料准备：**

前期的文献查阅和资料准备方面，我们主要从三方面进行：

一是通过菜鸟网站了解学习 C++面向对象编程的相关知识[1]，以及通过官方文档熟悉 MFC 环境的使用[2]；

二是阅读文献了解机器视觉在工业领域的常见运用方法，如基于模板匹配的物体识别方法、基于特征点的目标识别方法以及基于机器学习的目标识别方法[3]。

三是学习了基于机器视觉对豆子进行识别分类的常见思路，如利用动态阈值灰度化和形态学腐蚀膨胀将豆子分割[4],以及提取颜色和纹理参数作为豆子、板栗等圆形物体的特征参数[5]。

**硬件方案：**

相机型号：MV-E EM 系列千兆以太网工业相机，具体型号为 EM120C
采集图像分辨率：1280×960
增益：关闭自动增益，设为定值 3
曝光时间：5us
帧率：40.05fps
伽马值：1
包大小：8996
包延迟：500
RGB 系数：R=2.12  G=1.00  B=1.79
饱和度：0
目标亮度：128

**软件方案：**

1. 使用基于 X64 版本的 Visual Studio 2019 开发环境
2. 使用维视图像(Microvision)提供的 MVGigE 等底层相机库
3. 使用基于对话框的 MFC 图形化交互界面

# 四、源程序设计和运行结果

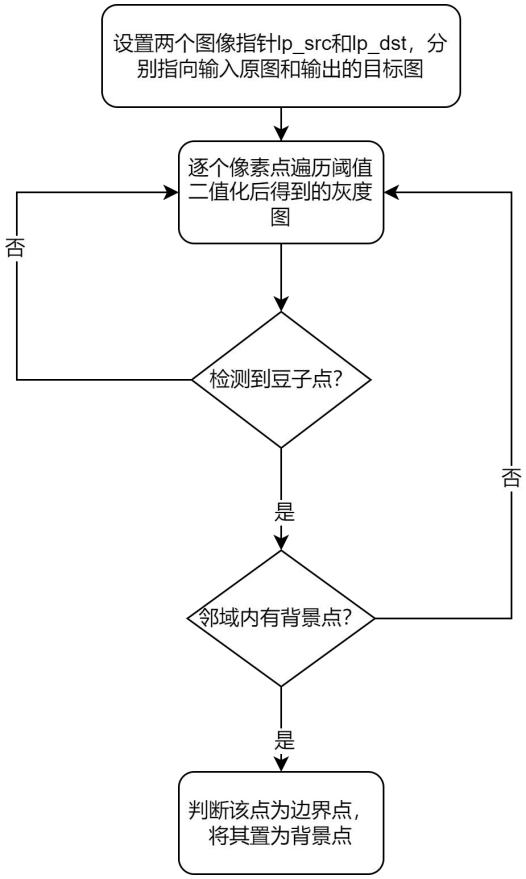
## 1. 腐蚀膨胀函数

- 函数原型：

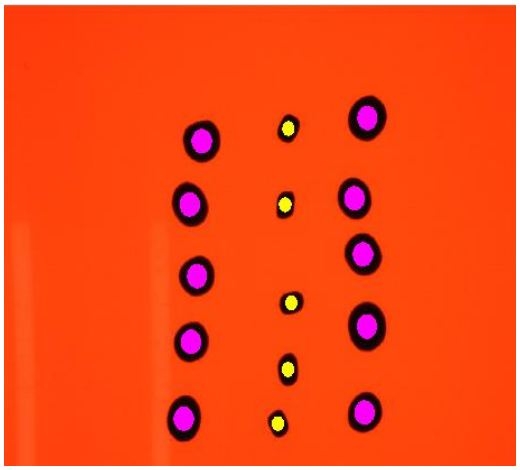
```
void another_shrink(MVImage* image_input, MVImage* image_output);

void dilation(MVImage* image_input, MVImage* image_output);
```

- 参数说明：输入图像和输出图像，皆为 MVImage 的类指针
- 功能说明：将图像中的豆子腐蚀或膨胀一次

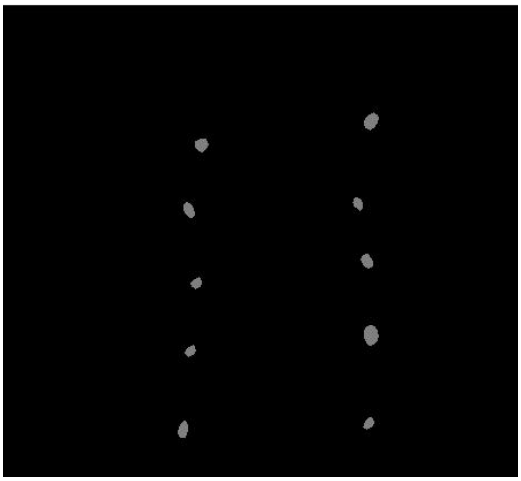
算法原理：设置两个图像指针 lp_src 和 lp_dst，分别指向输入原图和输出的目标图，逐个像素点遍历阈值二值化后得到的灰度图（豆子为灰色，灰度值为 128；背景为黑色，灰度值为 0）。若遍历到豆子点，则检测其邻域是否有背景点，若有则认为该豆子点为边界点，立刻将其置为背景黑色点，从而实现“边界收缩”和“豆子腐蚀”；膨胀算法原理与之相同，只是将背景和豆子的判断反了过来，相当于是“反向的腐蚀”



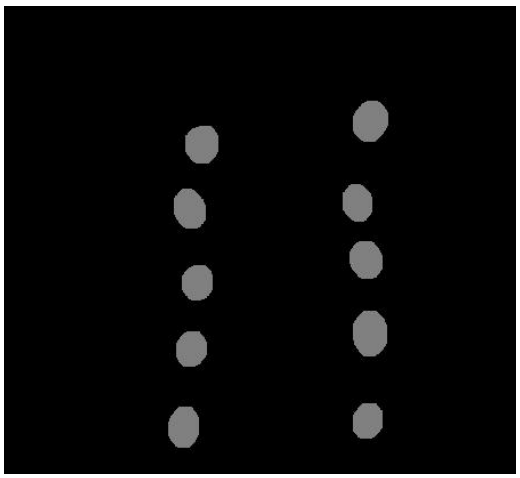
- 运行结果：



原图



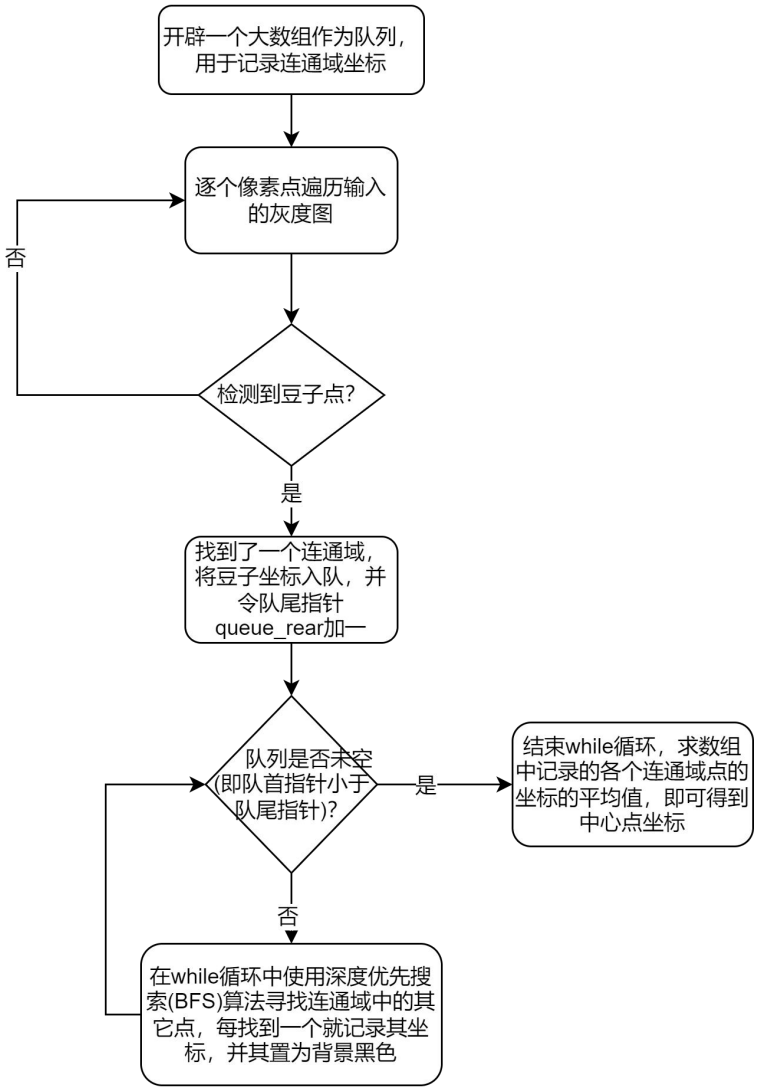
腐蚀后得到的缩小黄豆图



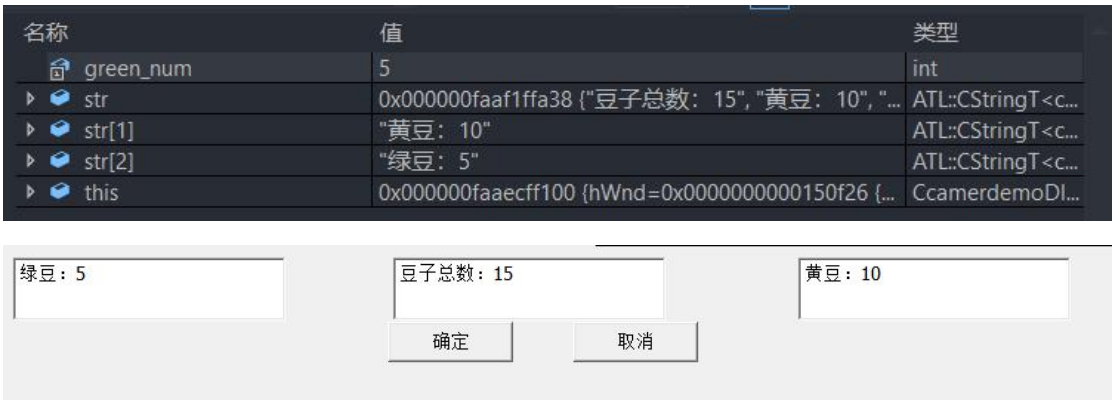
膨胀后得到的黄豆图

## 2. 连通域计数函数

- 函数原型：void count_beans(MVImage* image_input1, int mode);
- 参数说明：image_input1 为输入图像的指针和输出图像，mode 为计数模式，模式 0 代表数绿豆，模式 1 代表数黄豆
- 功能说明：记录图像中各个连通域的中心点坐标
- 算法原理：开辟一个大数组作为队列，遍历输入的灰度图的每个像素点，若为豆子点，则表示此时找到了一个连通域，将豆子坐标入队，并令队尾指针 queue_rear 加一。接下来进入 while 循环，当队列未空(即队首指针小于队尾指针)时使用深度优先搜索(BFS)算法寻找连通域中的其它点，每找到一个就记录其坐标，并其置为背景黑色，退出 while 循环时已将该连通域内的点坐标全记录在另一个数组中，并已将该连通域全涂黑。接下来求数组中记录的坐标的平均值，即可得到中心点坐标。



- 运行结果：




### 3. 圆域涂色标记函数

- 函数原型：void paint_circle(int x, int y, int r, int w, int h, unsigned char* p, int mode);

- 参数说明：

x、y 为输入的中心点坐标；

r 为要画的圆的半径；

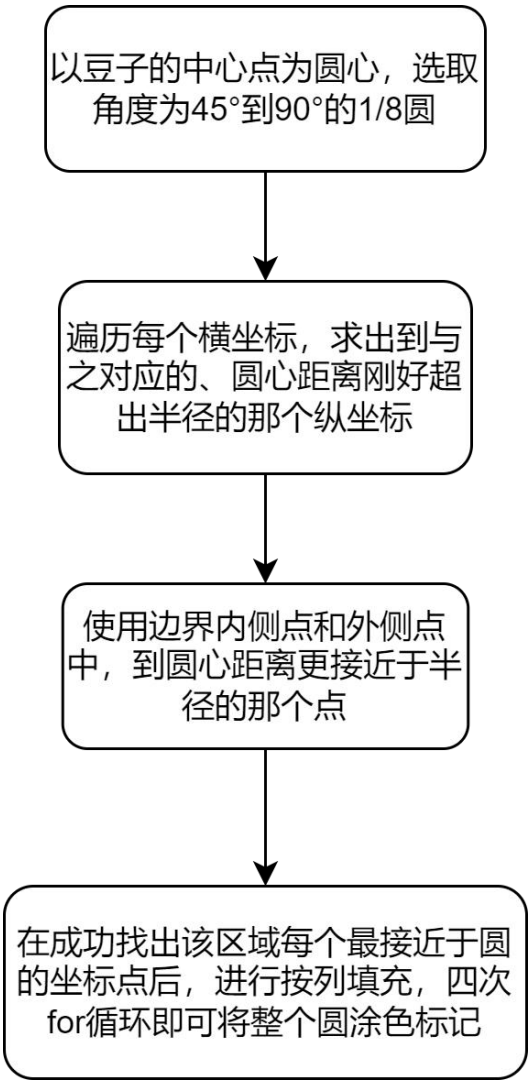
w、h 为原图的宽和高；

p 为指向原图像缓冲区的指针；

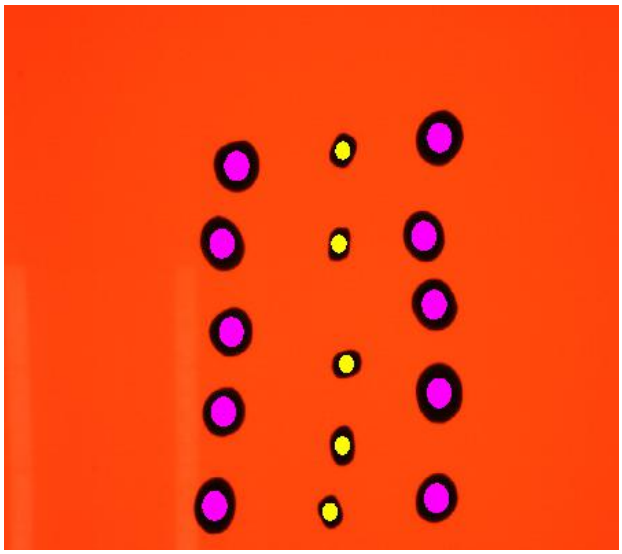
mode 为豆子类型，绿豆为 1，黄豆为 2

- 功能说明：以各个豆子的中心点为圆心，给一定范围的圆域上色，标记豆子

算法原理：将圆分成 8 等份，选取角度为 45° 到 90°（即横坐标为 0 到$\frac{\sqrt{2}}{2}r$）的那一份。对于每个横坐标，求出到圆心距离刚好超出半径的那个纵坐标(即使$\mathrm{i}^2+\mathrm{j}^2\geq\mathrm{r}^2$的 j)，接下来判断边界内侧点和外侧点哪一个更接近圆的坐标：其与半径的差值分别为$d_1=\left|\mathrm{i}^2+(\mathrm{j}-1)^2-\mathrm{r}^2\right|$和$d_2=\left|\mathrm{i}^2+\mathrm{j}^2-\mathrm{r}^2\right|$，且已知绝对值符号内$d_1$为负，$d_2$为正(因为分别位于圆的内外两侧)。我们需要他们中的较小者作为有效点坐标，因此我们可以判断他们差值的符号$d_2-d_1=2\mathrm{i}^2+2\mathrm{j}^2-2\mathrm{j}+1-2\mathrm{r}^2$如果为负就选取(i, j)，反之同理。由于一般横纵坐标都是远大于 1 的，因此上式顺理成章简化为$d_2-d_1=\mathrm{i}^2+\mathrm{j}^2-\mathrm{j}-\mathrm{r}^2$,同理判断符号即可。
在成功找出该区域每个最接近于圆的坐标点后，进行按列填充，四次 for 循环即可将整个圆涂色标记。



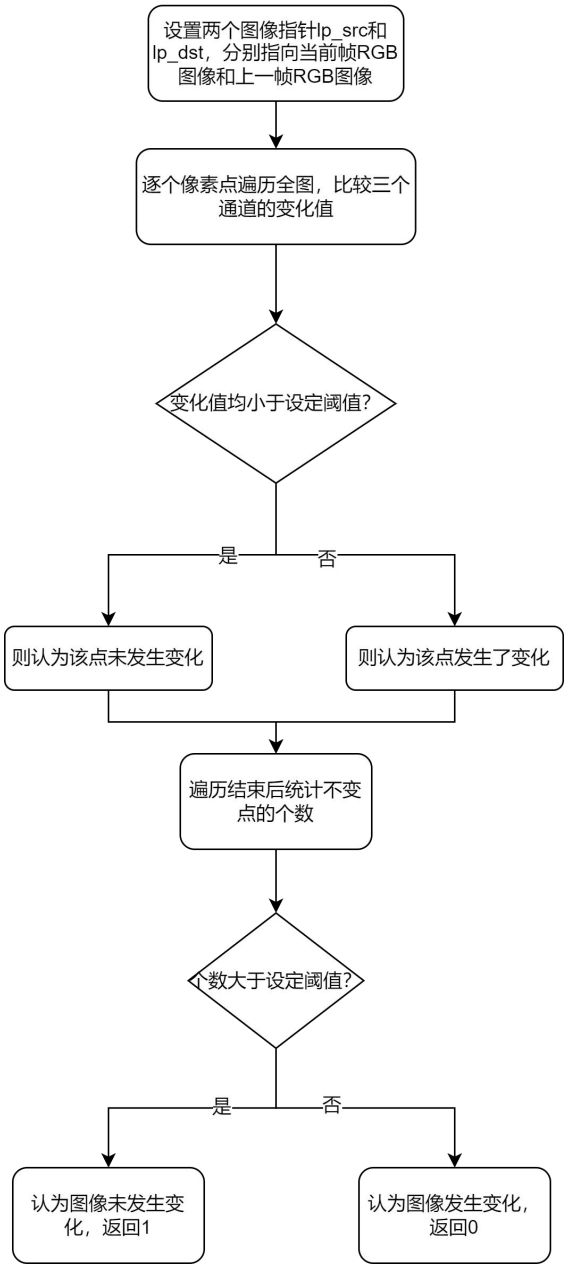

- 运行结果：



圆域涂色标记结果图

## 4. 图像帧比较函数

- 函数原型：int image_compare(MVImage* image_input1, MVImage* image_input2);
- 参数说明：image_input1 为当前帧相机获取的图像，image_input2 为上一帧获取的图像
- 功能说明：比较两张图像，返回 1 说明图像与前一刻相比没变，0 说明变了
- 算法原理：设置两个图像指针 lp_src 和 lp_dst，分别指向当前帧 RGB 图像和上一帧 RGB 图像，逐个像素点遍历并比较，若某个点三个通道的变化值均小于设定阈值，则认为该点未发生变化。最后统计不变点的总个数，大于阈值时认为图像未发生变化，返回 1；否则返回 0



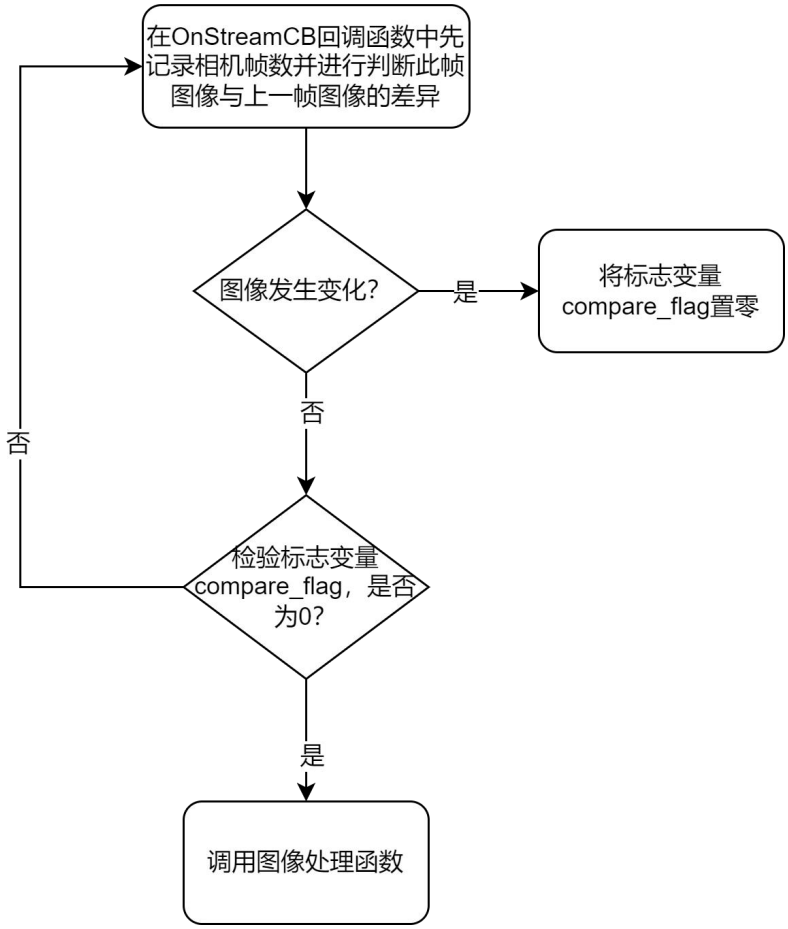
## 5.实时视频流信号优化处理算法

- 核心代码：

```
397     int compare_result = image_compare(&m_image, &m_image_last);
398     //返回1说明图像与前一刻相比没变，0说明变了
399
400     if (camera_frame == 0) //第0帧直接复制完成初始化即可
401     {
402         camera_frame++;
403         rgb_image_copy(&m_image, &m_image_last);
404         compare_flag = 0;
405     }
406     else if (compare_result == 0)
407     //图像变了，则将compare_flag置零，意味着在下次进入不变状态会触发图像处理
408     {
409         camera_frame++;
410         if (camera_frame == 30)
411             camera_frame = 1;
412
413         rgb_image_copy(&m_image, &m_image_last);
414
415         compare_flag = 0;
416     }
417     else //图像没变
418     {
419         camera_frame++;
420         if (camera_frame == 30)
421             camera_frame = 1;
422
423         rgb_image_copy(&m_image, &m_image_last);
424         if (compare_flag == 0)
425         {
426             OnBnClickedsort();
427         }
428         compare_flag = 1;
429         //图像处理完后记得置1，即只有“从变到不变时”才会触发图像处理，持续不变时不会触发
430     }
```

- 功能说明：使相机实时处理性能和鲁棒性有了极大提升，在处理少量豆子、大量豆子以及外界干扰（如用手加豆子或移动光源）时都能保持较高的流畅度和较好的稳定性。

- 算法原理：在 OnStreamCB 回调函数中先记录相机帧数并进行判断此帧图像与上一帧图像的差异。若图像发生变化，则将标志变量 compare_flag 置零；没变则检验 compare_flag，若为 0 则调用图像处理函数，意味图像稳定不变和持续剧烈变化时皆不处理，只有在图像发生变化时，才会在下一次不变时调用图像处理函数。




6. 四邻域八邻域交替检测算法

- 核心代码：

```
//大的黄豆膨胀恢复，最后得到的yellow_beans_image为复原的纯黄豆图
dilation(&m_shrink_end_image, &m_dilation_temp_image);
for (int i = 1; i <= dilation_times; i++)
{
    if (i % 2 == 0)
    {
        dilation(&m_dilation_temp_image, &m_dilation_end_image);
        image_move(&m_dilation_end_image, &m_dilation_temp_image);
    }
    else
    {
        dilation_4(&m_dilation_temp_image, &m_dilation_end_image);
        image_move(&m_dilation_end_image, &m_dilation_temp_image);
    }
}
```

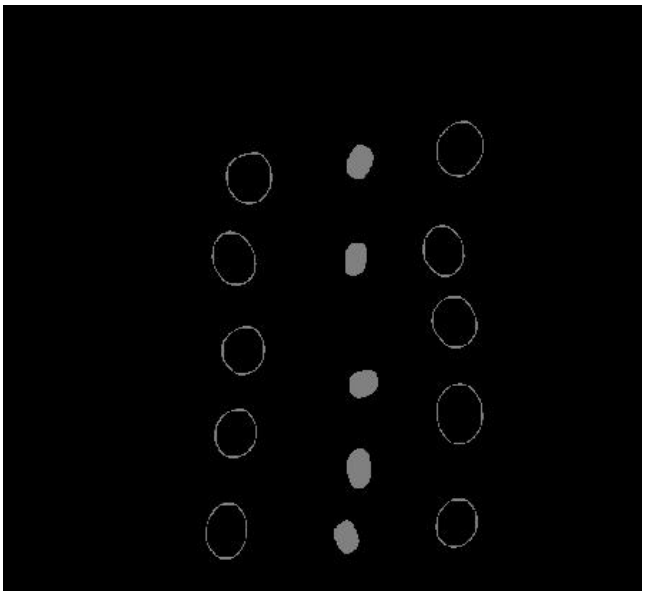
```
else if (*(lp_src - 1) == GREY ||
    *(lp_src + 1) == GREY ||
    *(lp_src - width - 1) == GREY ||
    *(lp_src - width) == GREY ||
    *(lp_src - width + 1) == GREY ||
    *(lp_src + width - 1) == GREY ||
    *(lp_src + width) == GREY ||
    *(lp_src + width + 1) == GREY )
        *lp_dst = GREY;
```

```
else if (*(lp_src - 1) == GREY ||
    *(lp_src + 1) == GREY ||
    *(lp_src - width) == GREY ||
    *(lp_src + width) == GREY )
    *lp_dst = GREY;
else *lp_dst = BLACK;
```

- 功能说明：使腐蚀和膨胀后的豆子与原形状十分接近，有效解决豆子分割不完全、粘连严重等问题

- 算法原理：无论是腐蚀还是膨胀算法，均需要检测邻域内的点，若仅使用八邻域，则腐蚀膨胀后的图像会趋于正方形；若仅使用四邻域，则图像会趋于菱形——这两种现象肯定都不好，因为会破坏豆子原有的圆形，方形凸出的角点会给图像造成严重粘连和重叠，不利于后续检验。
  为此，我们交替使用四八邻域检测，使腐蚀膨胀后的图像边缘变得平滑，接近于圆形。同时，该算法具有良好的拓展性，通过调整四邻域和八邻域的比重，理论上可使腐蚀膨胀后的图像趋于各种长宽比的椭圆。

- 结果展示：

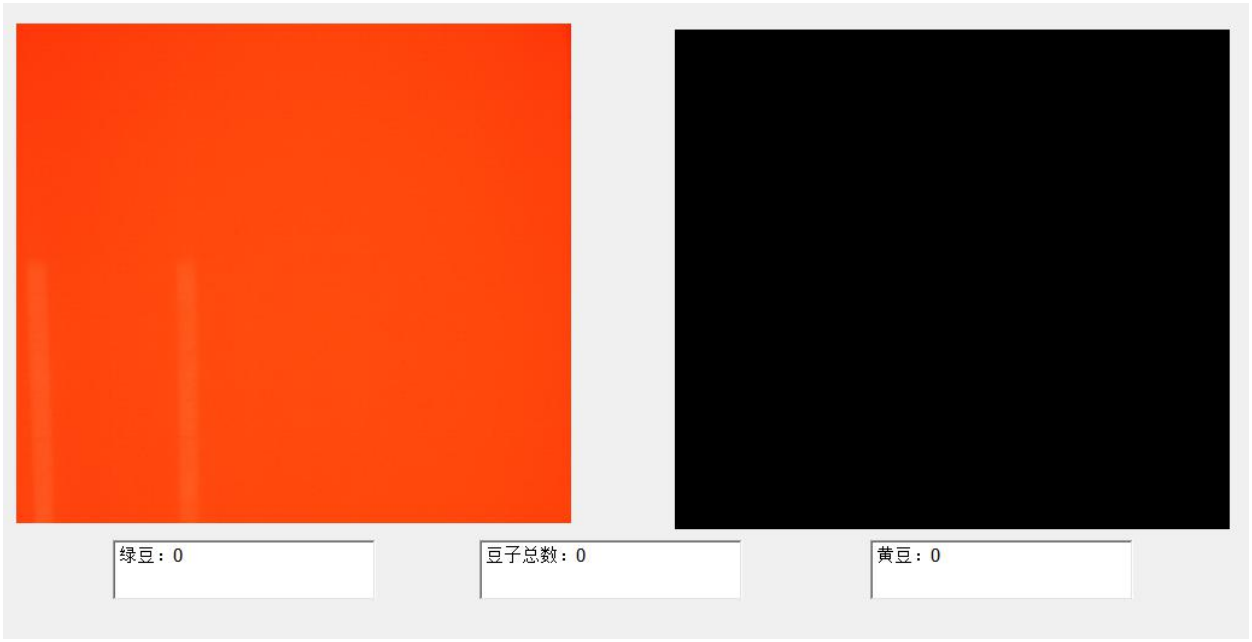


相减得到的绿豆图
（两边的灰色大圆为相减后黄豆的残余边界，
通过该边界形状可看出，四八邻域交替使用能很好地还原圆形）

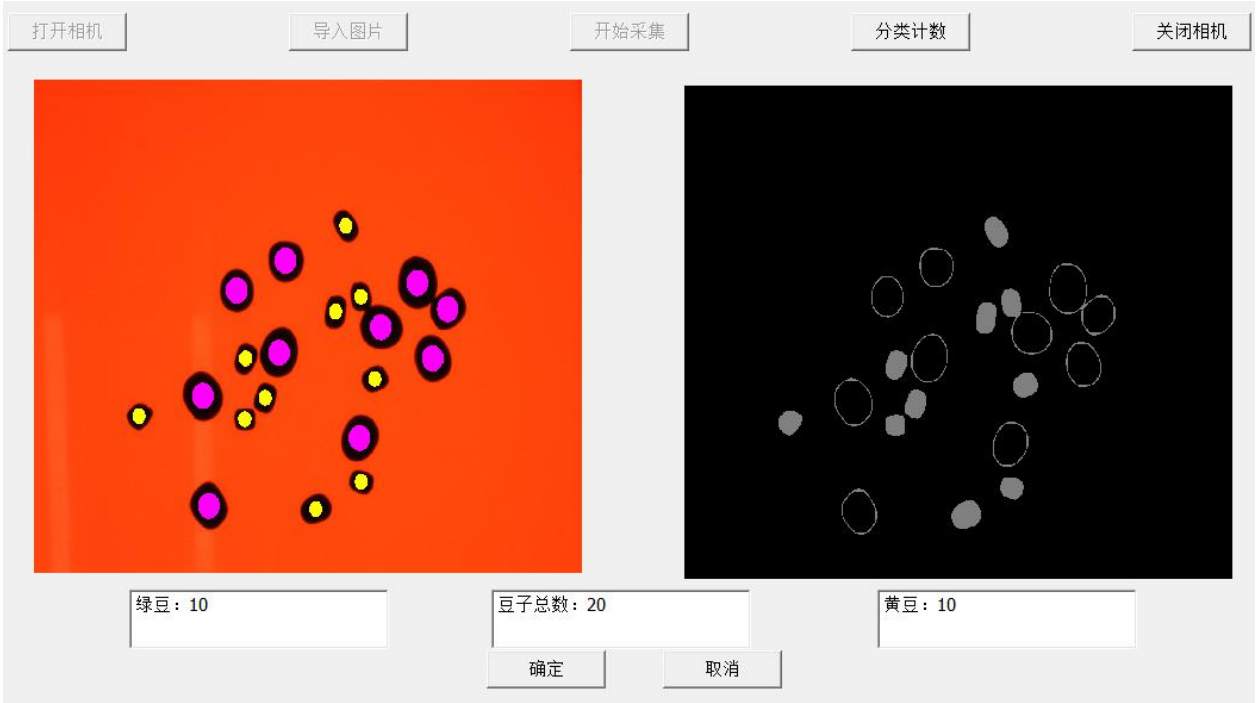
## 五、实验结果及其评价

### 1. 结果展示

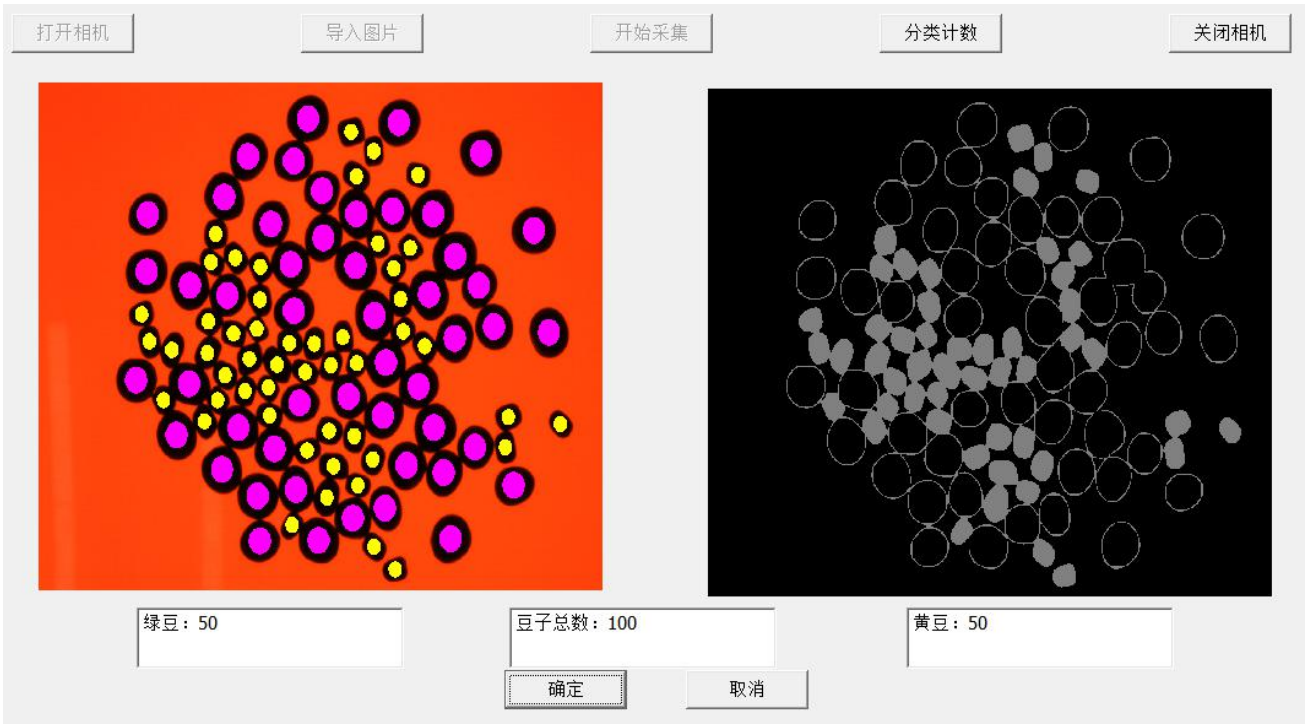
无豆子：不放豆子时程序只显示底色，计数为 0，不会崩溃或异常



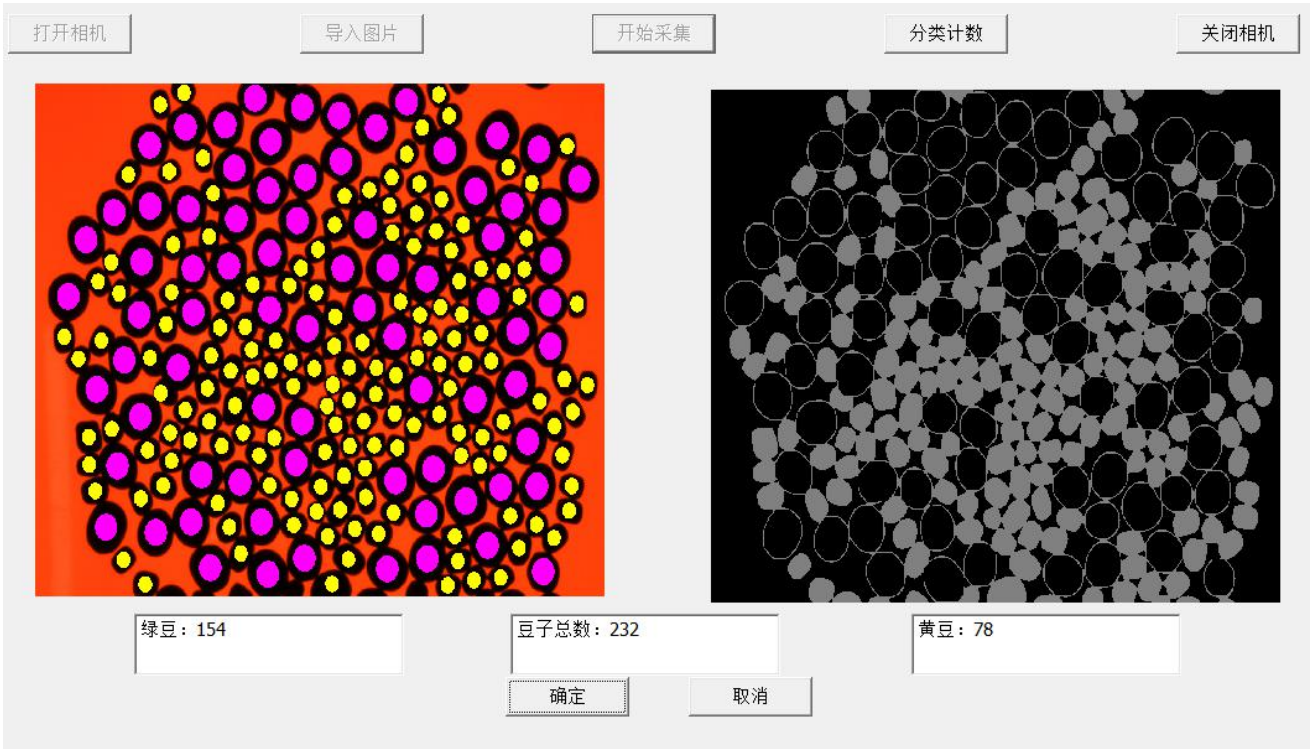

少量豆子：10 颗绿豆与 10 颗黄豆，皆能准确识别



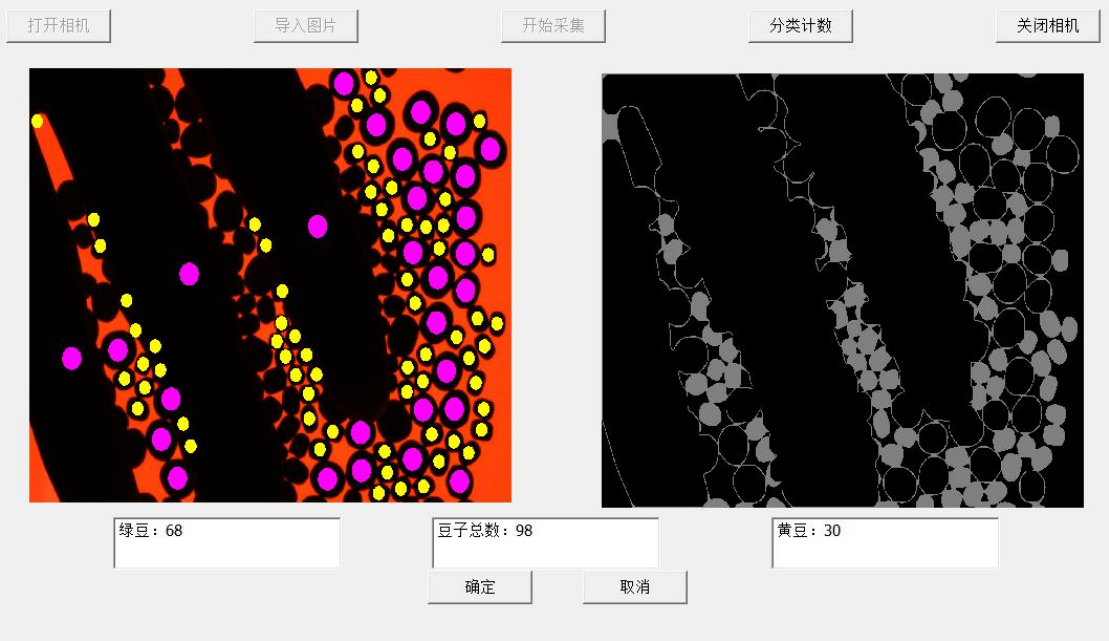
适量豆子：50 颗绿豆与 50 颗黄豆，皆能准确识别；无粘连，无异常




大量豆子：豆子几乎已将画面铺满，仍能准确将其分类计数，程序稳定无异常



外界干扰：将手遮挡于豆子之上，程序稳定运行无崩溃现象




## 2. 评价

**优点：**

(1) 建立了友好的可视化图形界面，能够实现图像连续采集。
(2) 有效解决大量粘连问题，确保能将黄绿豆正确分割。
(3) 保证目标检测的准确性，黄豆和绿豆都能被正确识别并计数。
(4) 使用基于 RGB 图的圆域涂色算法进行标记，清晰易懂且性能消耗大大降低。
(5) 创新性地使用四八邻域交替算法，有效解决图像变形、粘连等问题。
(6) 确保检验程序的实时性和流畅性，程序能对外界干扰做出鲁棒性反馈。

**缺点：**

(1) 无法处理豆子堆叠的情况。
(2) 腐蚀膨胀的效果准确度较依赖于参数整定，环境和硬件变化时可能需要重新调整参数。
(3) 应用场景有限，拓展性不强。若要识别三种及以上的豆子，则原有的图像分割算法（先腐蚀再膨胀再相减）的实际复杂度会大大增加；且连通域计数时申请了一个较大的全局数组，仅对上百颗豆子计数时尚可运行，若数量再增加一个量级，则会造成数组过大，程序内存不足。

## 六、体会和建议

本次数字图像处理和模式识别综合实验是本科期间为数不多的现代化的程序课设，且是我们第一次将机器视觉应用于实际场景，机会难得。通过长达 7 周的努力，我们成功开发出一套准确、流畅、稳定的黄绿豆分类计数程序，期间挫折与顺利并存，最终收获颇多——

首先是大型程序的开发流程中，初期应充分考虑、确定思路，而不是盲目上手！否则往往会在错误的道路上越走越远。如我们前期的算法皆基于本地文件流进行，花了许多时间在不必要的 RGB 图标记和手动进行灰度图转换上，事实上这些功能在实时处理中不过是调用一次库函数而已。

其次是要充分认识到“静态处理”和“实时处理”的巨大区别。前期由于我们的处理对象是静态的本地图片，所以未充分考虑算法的性能优化。我们初期使用了不断循环收缩边界+深度优先搜索(DFS)的方法来找中心点，该算法在静态图片的处理中表现良好，但一到相机实时检验，由于循环收缩边界的时间复杂度过高，以及 DFS 容易造成嵌套过度，程序爆栈，所以完全无法正常运行，最终我们不得不重构算法，改用队列+BFS 才得以解决问题。这对我们的代码能力，乃至未来编写更复杂、要求更严格的程序都是一个极大的教训与收获——小规模的实验性测试与大规模的真实场景应用可能是两种完全不同的思路，代码的性能优化是一件至关重要的事！

此外还有程序的 debug 调试：要善于运用断点设置、变量跟踪、性能探查器等现代工具进行调试，同时在编写程序时就要时刻注意代码规范问题，应保持良好的命名规则、宏定义和设计方法，好的代码风格是保证程序正确性和可读性的前提！

最后，真心感谢两位老师的悉心指导。马杰老师在图像处理、计算机视觉乃至摄影方面的广博知识给我留下了深刻印象，郑定富老师扎实的 C++功底和丰富的机器视觉应用经验都给我们的程序设计提供了极大的帮助，没有两位老师的倾心指导，我们不可能顺利完成此次任务，再次表示诚挚的感激和敬意！

# 七、参考文献

[1]runoob.com. C++ 类 & 对象[EB/OL]. [2022-3-30]. https://www.runoob.com/cplusplus/cpp-classes-objects.html.

[2]Microsoft. MFC 桌面应用程序[EB/OL]. [2022-3-30]. https://docs.microsoft.com/zh-cn/cpp/mfc/mfc-desktop-applications?view=msvc-160.

[3] 李政源. 基于机器视觉的工件识别与定位算法[D]. 山东:山东大学,2019.

[4] 王润涛,张长利,房俊龙,等. 基于机器视觉的大豆籽粒精选技术[J]. 农业工程学报,2011,27(8):355-359. DOI:10.3969/j.issn.1002-6819.2011.08.062.

[5] 展慧,李小昱,王为,等. 基于机器视觉的板栗分级检测方法[J]. 农业工程学报,2010,26(4):327-331. DOI:10.3969/j.issn.1002-6819.2010.04.056.

| 指导教师的评语及评分 | |
|---|---|
| | |
| 综合评分 | |