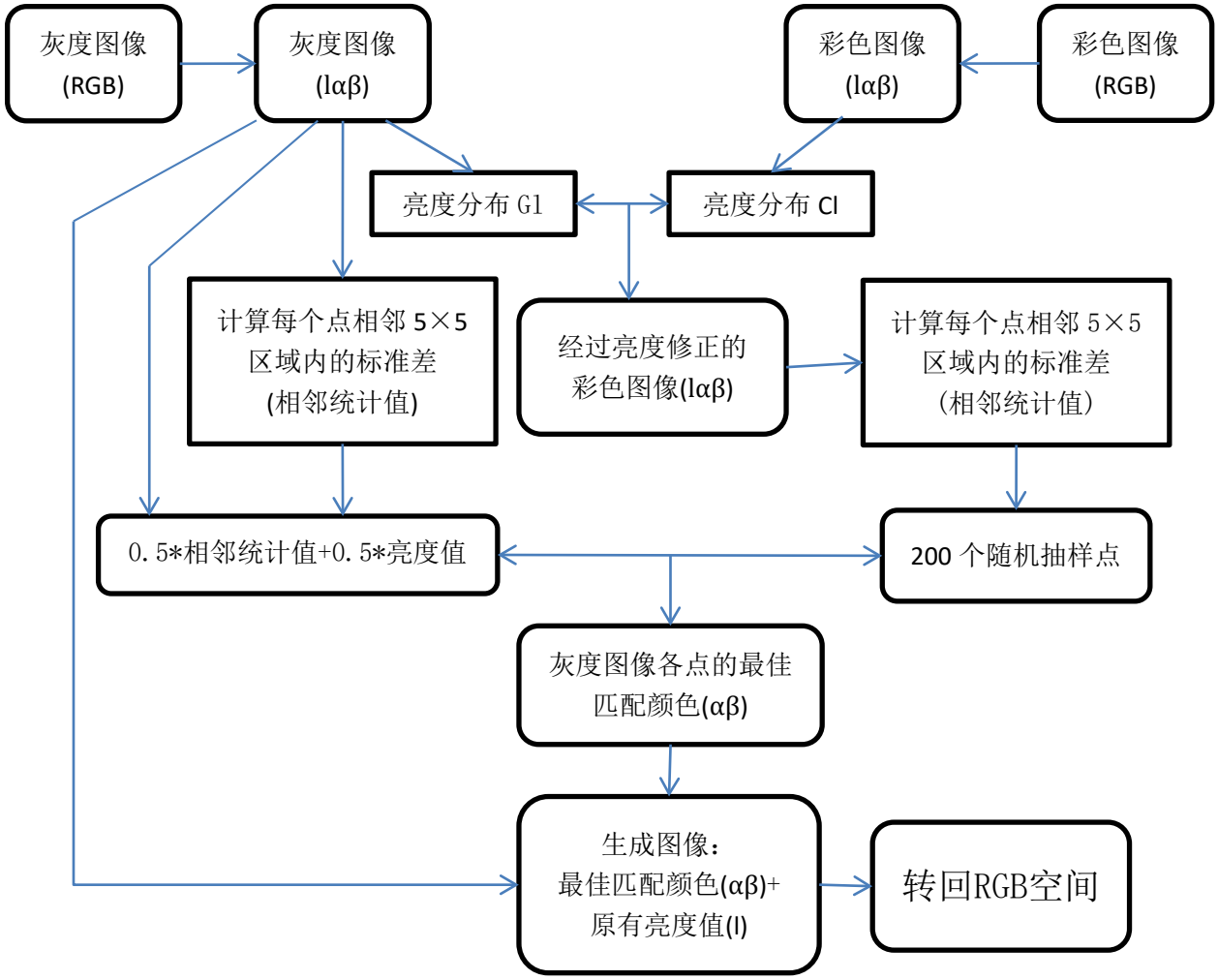
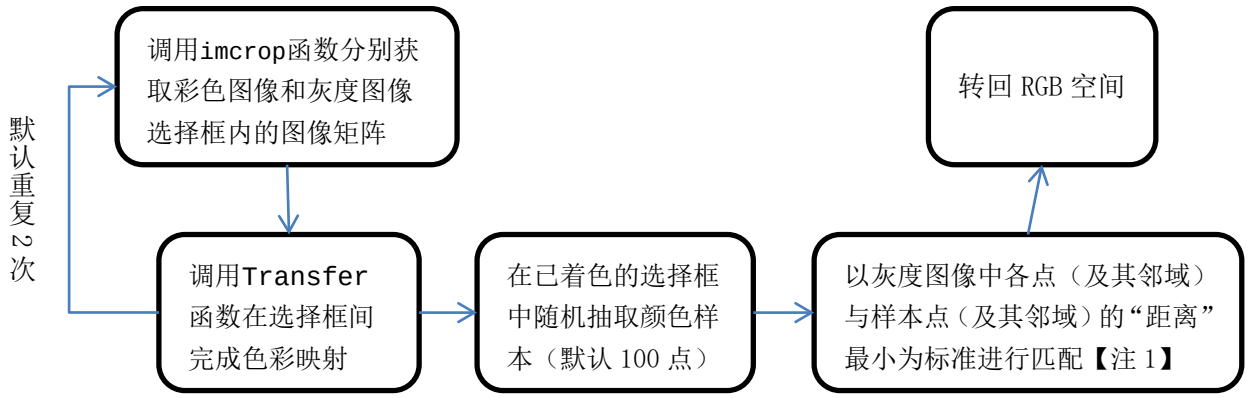
一、 算法流程

1. 全局匹配



2. 基于选择框



【注1】 此处的公式为：$\mathrm{E}(N_g, N_s) = \sum_{p \in N}[I(p) - S(p)]^2$，其中$N_g$是灰度图像中的邻近点，$N_s$是着色了的选择框中的邻近点，I 代表灰度图像，S 是被选择的彩色图像的亮度通道，p 代表邻近的像素点。

二、 程序说明

1. 全局匹配

1) 函数说明

算法见流程图。运行 go.m 后即会将 C.jpg 的色彩映射到 G.jpg（示例中预先转为灰度图像了。若非灰度图像，只需将第 5 行的注释去掉即可），生成名为 NewG.jpg 的文件。

2) 执行结果

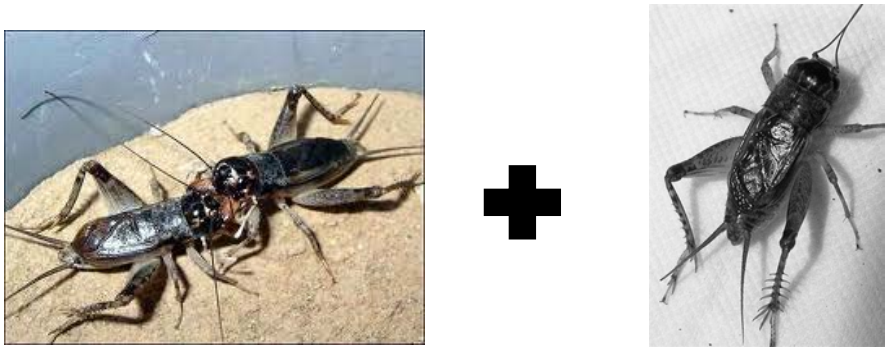
2. 基于选择框

1) 函数说明

主函数：Swatches.m，其中调用了 Transfer.m（由 go.m 修改并封装得到）。

其中，主函数的算法实现见流程图。Transfer函数的入口参数Cdata,Gdata,N分别为转换成double型的彩色图像矩阵，转换成double型的灰度图像矩阵和彩色图像中的随机撒点数；返回值是经过色彩映射得到的灰度图像的α，β值。

2) 执行结果

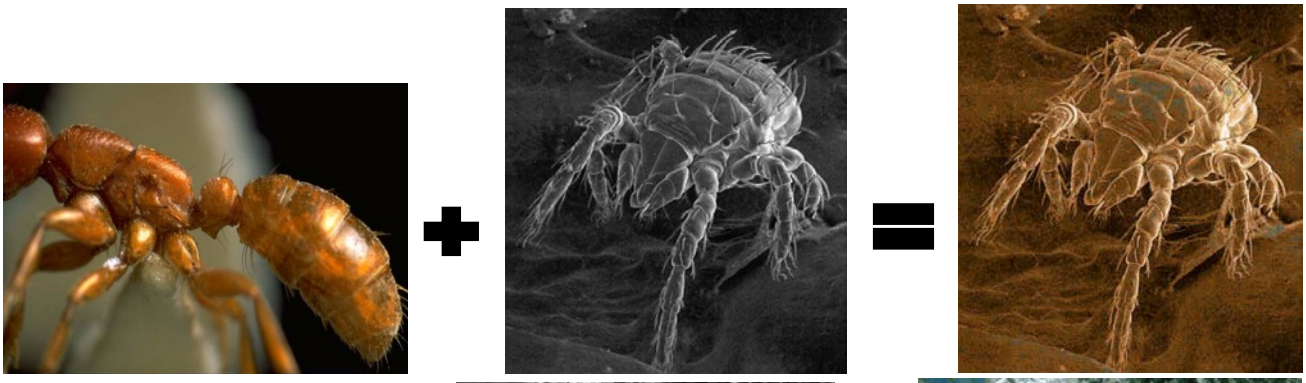


全局匹配结果　　基于选择框匹配结果

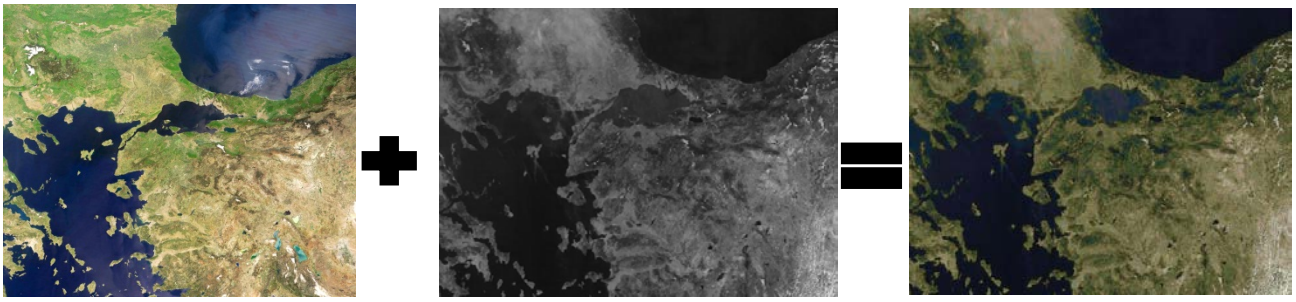
三、 结果分析

1. 测试用例分析

对于全局匹配，我测试了论文中提供的两组图像，效果很好。如下所示：



但是这种算法的鲁棒性是存在问题的。做实验的过程中我花费了一整个晚上，在网上搜索了二十组左右我觉得还算合适的的图像，仅发现了两三组能有比较好的效果。并且它们都是极其相似的。（失败的图像没有保存下来，因为实在太多太常见了…）



对于基于选择框的匹配，使用以上图像进行测试时发现性能并没有什么明显的改善，许多甚至还不如全局匹配（以上图像的全局匹配效果都太好了…）。

但是由二（2）的“蟋蟀”那幅图的试验中我们至少可以发现，当彩色图像中的色块并不全是我们想要的时候，选择框能有效地起到过滤作用——显然，左边那幅生成图像中有大面积不和谐的蓝色，右边就没有。

2. 算法的优缺点

对于算法，整体来说论文中都说明的十分清楚了。具体实现过程中，我觉得最有技术的部分就是循环的矩阵化。

在go.m中计算相邻统计值的时候，以及在Swatches.m中图像内的纹理匹配部分，我花费了大量的时间尽可能地把循环都改成了矩阵计算。这使得算法的效率相当不错，全局匹配中几万个像素的图像十秒内就可以出结果。

对于基于选择框的匹配，主观因素起了很大的作用。不同的标注会带来差别不小的结果。因此选择合适的匹配区域成为了一件很重要的事。

关于算法的改进想法：

（1） 如论文所述，可以改变邻域的范围。经过试验可知，5*5是个不错的选择。

（2） 对于最优匹配的标准可以选择不同的加权系数。时间充裕的话可以编写代码对加权系数进行参数扫描以确定相当最优的权值组合。

（3） 关于随机抽样点数的选择。需要在性能和耗时中做一个平衡。

（4） 对于选择框匹配，可以增加选择框的数量，使得映射更精细。

3. 遇到的问题及经验

（1） 在做基于选择框的匹配时，起初运行耗时过长，快一小时了还没出结果。后来发现是由于图像内的纹理匹配过程中没有进行随机取点（我当初以为选择框范围不大，全部用于计算匹配应该也不会太耗时间….），而是将已着色区域内所有的点都加入了计算。

（2） 做循环矩阵化的过程中绕晕了数次。在此就不详述了。

（3） 还是基于选择框的部分，选择框的标注位置对结果有很大的影响。我调试和试验是采取的技巧是：先将30~64行注释掉，这样生成的图像就是选择框间的映射结果。由此我们可以看看这一步的匹配是不是符合我们的期望。