

数字图像处理第二次大作业实验报告

自 42 张博文 2014011455

目录

- 1.方法原理的简要说明3
- 2.实验思路4
 - 2.1 整体实验思路4
 - 2.2 获取掩模4
 - 2.2.1 阈值分割.....4
 - 2.2.2 fillblack 对掩模进行的调整5
 - 2.3 提取光照图.....5
 - 2.4 去光照工件图6
 - 2.5 反光区上色.....7
 - 2.5.1 获取工件灰度.....7
 - 2.5.2 求取需要刷颜色的范围7
 - 2.6 灰度图转 RGB 图9
- 3.程序结果与性能10
 - 3.1 效果图与鲁棒性.....10
 - 3.2 算法复杂度.....12

1.方法原理的简要说明

我编写的去反光程序在 img_dereflect.m 文件中。其中 img_dereflect 函数为主函数，另外函数的名字和功能分别如下表

函数名	函数调用方式		函数功能
GuassianL	H=GuassianL(img,D0)		构造频域高斯低通滤波器。img 为频谱图，D0 为截至频率
getmask	ym=getmask(img,low,high)		取工件的掩模。img 为灰度图，low，high 分别为阈值。最终得到的掩模工件部分为 0，背景部分为 1
	子函数	H=divide(img,a,b)	阈值分割得到粗掩模。img 为灰度图，low,high 分别为阈值。工件部分（阈值内）为 0，背景（阈值外）为 1
		ymnew=fillblack(ymold)	处理粗掩模，将掩模工件部分中的不合理的部分重新修改
adjustcolor	newimg=adjustcolor(oldgray,newgray,mask)		将新的灰度图中工件反光区根据原灰度图重新刷成工件的颜色。oldgray 为原灰度图，newgray 为新灰度图，mask 为掩模
	子函数	highlight=gethighlight(grayimg)	取工件上反光集中的区域。值为 1 的区域代表反光强的部分
	子函数	contour=getcontour(grayimg)	取工件轮廓。值为 1 的区域代表工件轮廓
extend	newimg=extend(standard,oldimg)		根据 standard 的取值范围，扩大或缩小 oldimg 的取值范围，使二者变化范围相同
getRGB	newimg=getRGB(oldimg,newgray)		根据原 RGB 图和最终得到的灰度图及掩模，恢复原来的 RGB 图
	子函数	newrgb=gray2rgb(newgray,oldrgb)	由灰度图直接转成 RGB 图

函数 img_dereflect 是整个程序的入口，整个程序是在图像的灰度图上操作的，只在最终将灰度图转化为 RGB 图，img_dereflect 函数按照顺序完成了以下功能。

- ①读取 RGB 图并转化成灰度图
- ②根据灰度图生成工件掩模
- ③使用同态滤波获取光照图
- ④用原图减去光照图并将结果增强
- ⑤为反光强烈区刷上工件的颜色
- ⑥根据灰度图得到最终的 RGB 图

⑦使用掩模去掉背景并显示

2.实验思路

2.1 整体实验思路

整个去反光程序关键是：获取掩模，提取光照图，提取除去光照的工件图，为反光区重新上色，灰度图转 RGB 图。

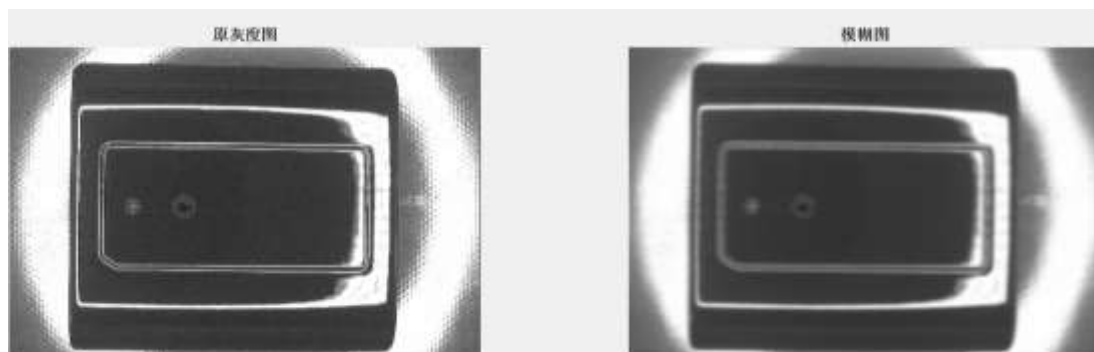
虽然提取掩模属于提高任务，但是由于在后续的操作中需要用到掩模，所以我从掩模的思路开始介绍。

2.2 获取掩模

获取掩模主要使用了函数`ym=getmask(img,low,high)`，其中包括另外两个比较重要的函数`divide`,`fillblack`。获取掩模的流程如下：

- ①对灰度图 `img` 进行高斯模糊处理
- ②根据输入的阈值，使用 `divide` 函数对模糊后的灰度图进行阈值分割
- ③使用 `fillblack` 函数对掩模进行调整

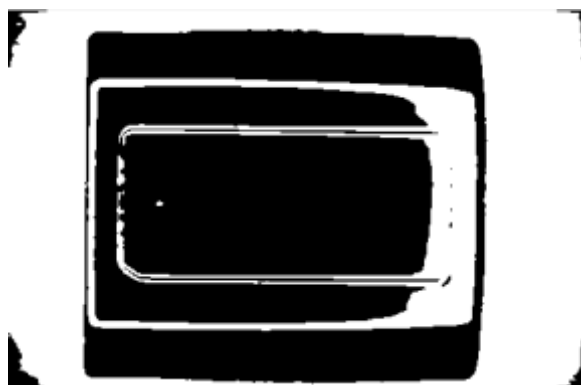
在进行阈值分割之前对灰度图高斯模糊是为了模糊边缘，以使在阈值分割时能分出比较完整的区域。高斯模糊的方法则是将图像在频域通过了高斯低通滤波器。



2.2.1 阈值分割

由于原灰度图中背景区域既含有比工件暗的背景，也含有比工件亮的反光区，所以为了将背景和工件分开，需要提供两个阈值 `low`，`high`，只有介于二者中间的才是工件。

阈值分割的过程就是遍历图像各点，凡是介于 `low` 和 `high` 之间的点，则认为是工件，使该点掩模的值为 0；若位于外部，则说明是背景区域，使该点的值为 1。最终得到的效果如下：



可以看到，此时阈值分割的结果并不能当作最终的掩模，因为图像的四角由于模糊的原因并未被识别为工件，而工件上的反光区由于灰度值较大也被归在了阈值之外，因此还需要对阈值分割的结果使用 fillblack 函数进行一些调整。

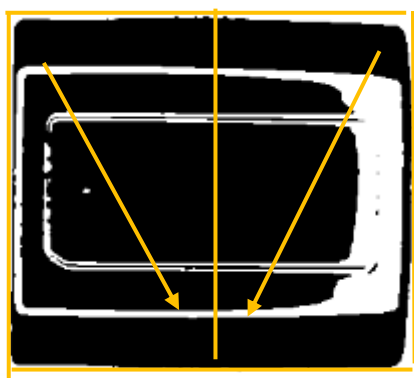
2.2.2 fillblack 对掩模进行的调整

为了获取最终的掩模，首先将得到的阈值分割图的四个角赋值为 1（白）。之后再处理工件中间的部分。

在处理工件中间的部分时，需要将被黑色（0）包围的白色（1）区域变为黑色（0），在这里我采用了以下方法：

- ① 找到黑色区域的上下左右的边界坐标。
- ② 将黑色区域左右一分为二
- ③ 对左边区域，从左上角向右下角遍历，当遇到白色点时，若白色点左侧和上方的点皆为黑色，则将白色点改为黑色点；否则保持不变。
- ④ 对右边区域，从右上角向左下角遍历，当遇到白色点时，若白色点右侧和上方的点皆为黑色，则将白色点改为黑色点；否则保持不变。

算法示意图和效果如下：



示意图



最终效果图

2.3 提取光照图

提取光照图的方法为同态滤波。

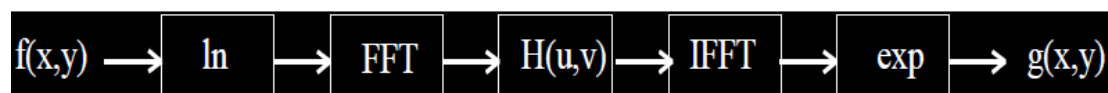
设图像的入射光照为 $i(x,y)$, 反射光为 $r(x,y)$, 则图像可表示为 $f(x,y) = i(x,y)r(x,y)$, 对图像取 \ln , 则有

$$\ln f(x,y) = \ln i(x,y) + \ln r(x,y)$$

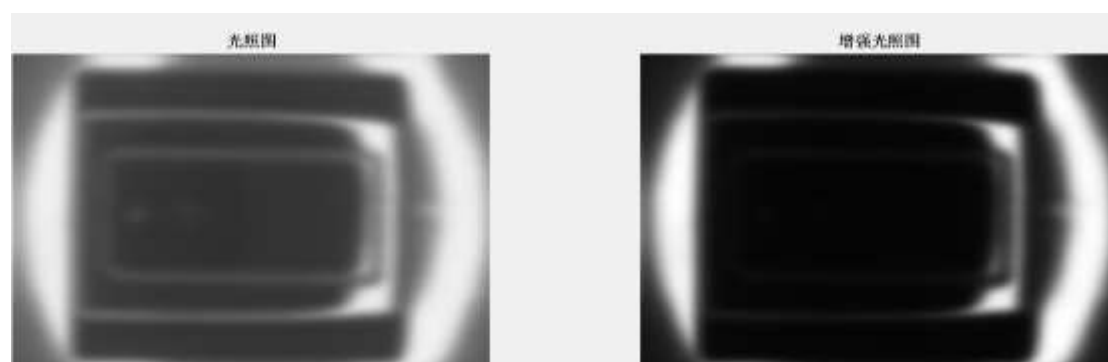
$$\ln F(u,v) = \ln I(u,v) + \ln R(u,v)$$

一般假定入射光 $i(x,y)$ 动态范围大但变化缓慢, 反射光 $r(x,y)$ 变化迅速。所以在频域使用高斯低通滤波就可以保留 $\ln i(x,y)$ 部分而滤掉 $\ln r(x,y)$ 部分, 最后再取 \exp , 即可得到入射光。

整个操作如下：



对于工件而言, 其上的反光区即为入射光, 而工件本身为反射光, 所以使用此种方法即可得到工件的光照图。但是由于在低通滤波的时候损失了一部分光照, 所以此时需要将光照图增强, 以便后续操作。



2.4 去光照工件图

在提取了光照图 $i(x,y)$ 之后, 此时需要将光照从原图上去掉, 根据

$$\ln f(x,y) = \ln i(x,y) + \ln r(x,y)$$

可得

$$r(x,y) = \exp(\ln f(x,y) - \ln i(x,y))$$

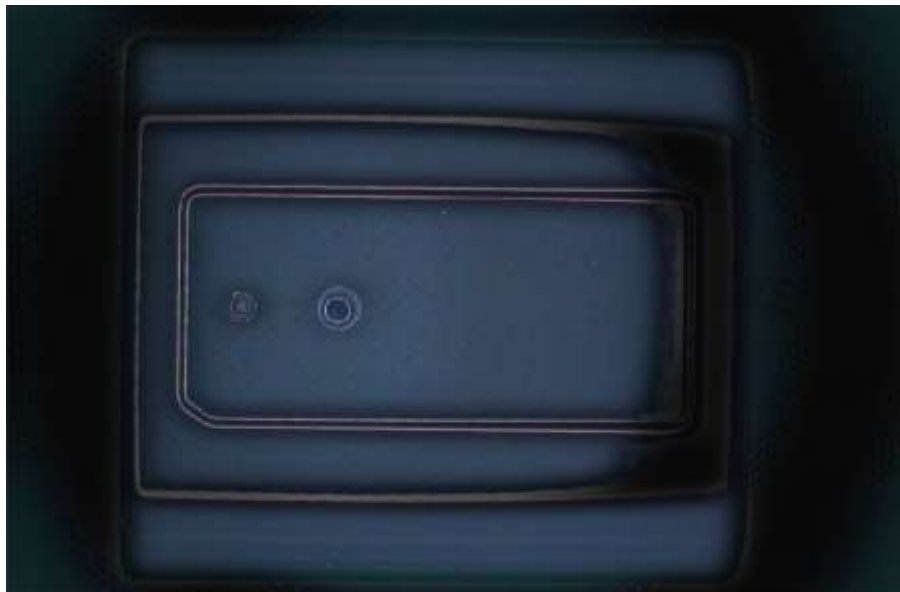
则此时可以得到除去光照图后的工件图。之后再将得到的 $r(x,y)$ 使用 `extend` 函数增强, 即可得到初步去光照的工件图。



得到了去光照的增强图之后, 其实已经可以根据此灰度图求出最终的 RGB 图。但是由于反光过于强烈, 所以在去除了反光之后, 这些区域并不具有工件的颜色, 而是比工件的颜

色更暗，略有发黑，所以此时得到的结果并不是最佳的，需要进一步将工件中没有颜色的反光区域赋值，使其和工件普通表面一致。

下图为由去光照增强的灰度图直接求出最终的 RGB 图，可以发现之前的反光区域发黑。



2.5 反光区上色

反光去上色的目的很简单，就是将反光区域刷成和工件相同的颜色。但是这其中需要处理以下几个问题：

- 工件的颜色是什么
- 需要刷颜色的反光区域有哪些

2.5.1 获取工件灰度

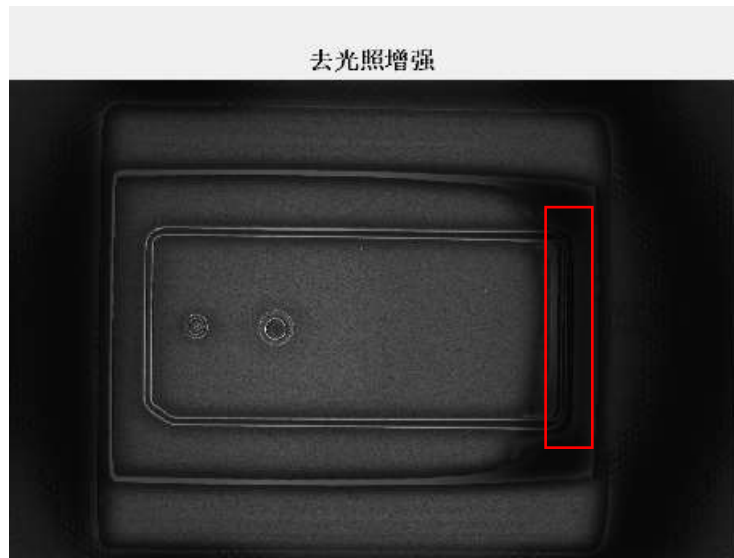
获取工件灰度其实就是取工件上无反光点的灰度值。所以需要分为两步，首先根据掩模将原来灰度图的背景部分赋值为 255，之后统计图像各个灰度值出现的次数。对灰度值在 0-200 范围的点取加权平均，即得到工件的灰度值。

之所以能得到工件的灰度值是因为此时工件的背景为 255，而工件本身反光的区域也大于 200，所以 0-200 即为除去反光和背景之后工件上的颜色变化范围。

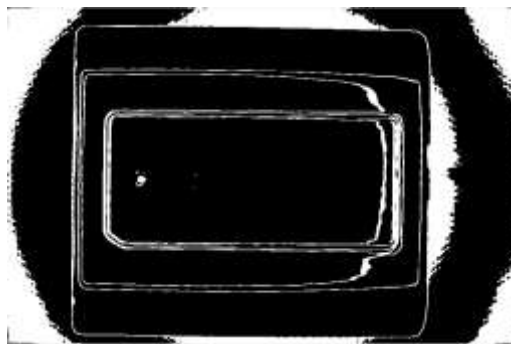
2.5.2 求取需要刷颜色的范围

首先需要刷颜色的范围肯定是工件的反光区，因此可以使用去光照图和之前的灰度图进行比较，范围变化小于给定阈值的，说明此部分非反光，所以只需要保持原灰度图不变；若变化范围大于给定阈值，则说明为反光区。

但是，根据下图可知，在前面的操作中，右侧的框线也被包括在了反光区域中，所以如果直接使用阈值得到的反光区，会将这部分框线也上色，最终导致框线消失。



所以此时需要使用 `contour` 函数，提取出图像的边框，这些部分应保持原样不能上色。由 `contour` 函数提取出的边框如下：



提取边框分为两步，首先在频域使原灰度图通过高斯高通滤波器，滤出灰度图的边框部分，之后与原灰度图进行比较，变化范围小于给定阈值的即为边框。

另外，由于在边框附近也存在着一些较为稀疏的光照，所以如果赋值就会导致边框出现模糊，所以此时应该选出光照集中的区域，找到反光最集中的区域。由 `gethighlight` 函数提取的光照集中区如下：

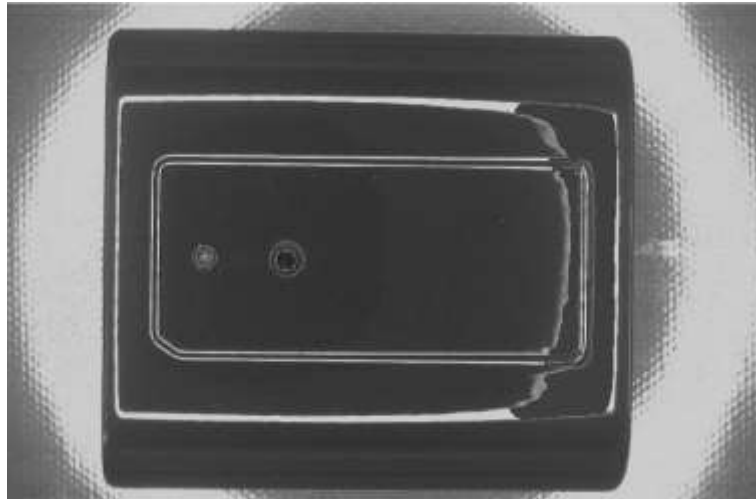


首先对原灰度图进行高斯模糊，之后对结果进行阈值分割，即可得到光照集中区。

综上，最终需要上色的区域应该满足以下条件：

- ① 位于工件之上
- ② 该区域在去反光之后与原灰度图变化大于阈值
- ③ 该区域不属于边框
- ④ 该区域属于光照集中区

最终得到的重新上色的灰度图如下：

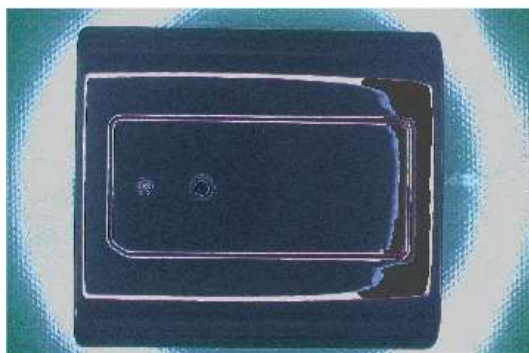


2.6 灰度图转 RGB 图

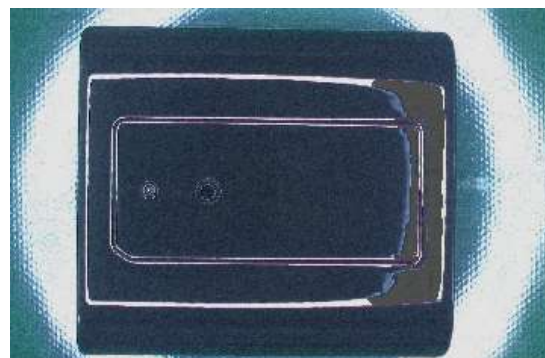
在得到了最终的灰度图之后，需要将其转化为 RGB 图，此时需要使用 `getRGB` 函数。在由灰度图转 RGB 图的过程中我采取了两种方法。

第一种方法是由最终的灰度图 and 原灰度图比较，将 RGB 图的各通道乘以比例。

第二种方法是将原 RGB 图转化到 HSV 通道，将得到的灰度图替换原来的 V 通道，之后再转化为 RGB 通道，这两种方法得到的最终结果分别如下：



第一种



第二种

观察可发现这两幅图的反光区的颜色都和工件有一定的差距，但是第一种颜色较深，第二种颜色比浅，因此我将二者取了平均得到了最终的结果。最终加上掩模，即为最终结果。



最终结果

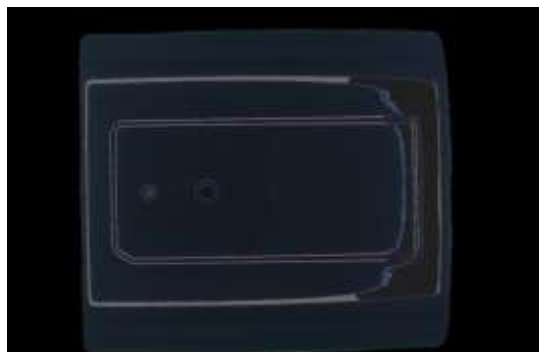
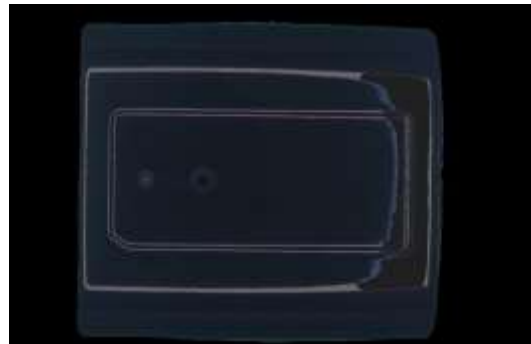


与背景分离

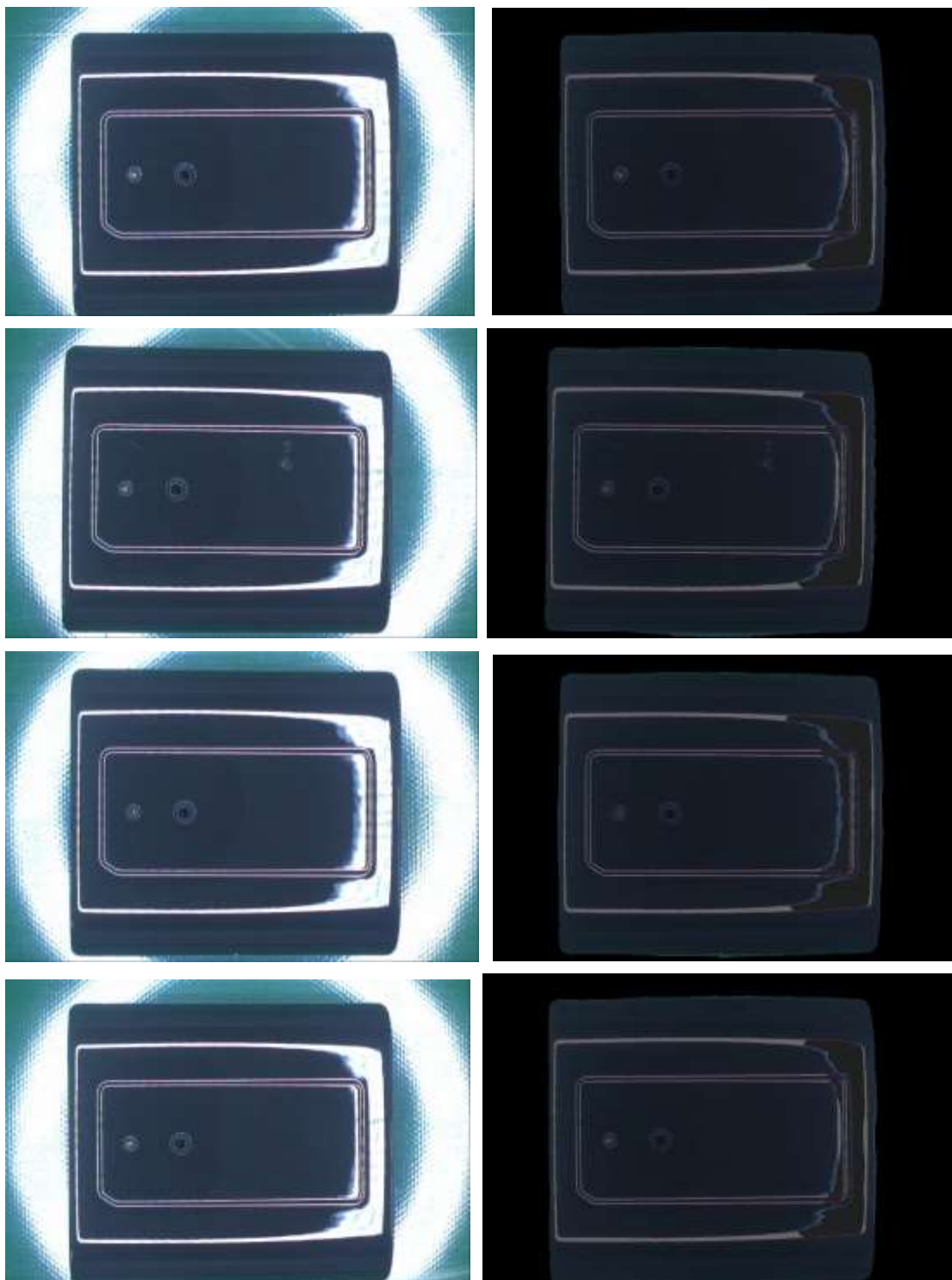
3.程序结果与性能

3.1 效果图与鲁棒性

为验证程序鲁棒性，我将测试图集集中的 10 张图缩小并且转为了 RGB 格式，通过 test.m 脚本可批量对图片进行操作，最终效果如下：



















3.2 算法复杂度

本次程序中，我尽量避免了循环，基本使用了点乘运算，只有在 fillblack 和 adjustcolor 函数中使用了局部的遍历函数，整个算法的复杂度应为 $O(m \times n)$ ，其中 m, n 为图片的尺寸。

对于像素值为 890×584 的图像，程序所花时间如下图：

Profile Summary

Generated 25-Nov-2016 19:38:04 using *cpu* time.

Function Name	Calls	Total Time	Self Time*	Total Time Plot (dark band = self time)
img_dereflect	1	0.954 s	0.075 s	
img_dereflect>adjustcolor	1	0.238 s	0.098 s	
img_dereflect>getRGB	1	0.206 s	0.011 s	
img_dereflect>getmask	1	0.170 s	0.012 s	
imshow	1	0.145 s	0.015 s	
initSize	1	0.094 s	0.007 s	
img_dereflect>fillblack	1	0.093 s	0.087 s	
ifft2	4	0.091 s	0.091 s	
rgb2hsv	1	0.089 s	0.089 s	
hsv2rgb	1	0.086 s	0.086 s	
movegui	1	0.085 s	0.081 s	
img_dereflect>getcontour	1	0.073 s	0.015 s	
m2	1	0.073 s	0.015 s	