

数字图像处理报告一：纸盘缺陷检测

姓名：鲁国锐 学号：17020021031 专业：电子信息科学与技术

2020 年 3 月 2 日

Contents

1 问题分析	2
1.1 题目描述	2
1.2 问题分析	2
2 解决方案	2
2.1 确定像素值范围	2
2.2 判断非正常像素点是否连成一个足够大的区域	3
2.2.1 算法描述	4
2.2.2 采用面向对象的方法降低时间复杂度	4
2.2.3 采用剪枝的方法降低时间复杂度	4
2.3 排除其它因素的干扰	4
3 算法设计	5
4 编程实现 ¹	6
5 结果展示与分析	8
5.1 结果展示	8
5.2 结果分析	10
6 总结体会	10

1 问题分析

1.1 题目描述

工业生产的纸盘由于产品线的技术问题会导致不同形式的损毁，比如污损，裂痕，斑点，残缺等，请结合对数字图像处理的理解设计一个进行纸盘缺陷检测的大体方案。

1.2 问题分析

本题要求从数字图像处理的角度来分析该问题。从这个角度出发，假设现在已经得到了一张关于纸带的图像，那么此时能够用来检测出其上是否有损毁的唯一标准就是像素值的大小。也就是说，需要通过分析像素值的大小来判定纸带是否有缺损。

沿着这条思路，又可以把该问题分为三个子问题：

1. 分别确定纸带正常区域及缺损区域的像素值范围；
2. 判断像素值在缺损范围的各个像素点是否连成一个区域，且该区域大到无法忽视的地步；
3. 排除其它因素的干扰（背景、阴影等）。

我们将在下一节中对这三个子问题分别进行讨论。

2 解决方案

2.1 确定像素值范围

针对1.2节中的问题1，可以直接通过实验的方法来确定。



图 1：实验样例 1

图1是一张普通卫生纸被撕开一块后，放在黑色背景下拍得的一张照片。但由于该图像分辨率较高，数据量大，不利于观察，所以将其裁剪后得到图2。



图 2: 实验样例 2 (从样例 1 中截取得到)

用 *MATLAB* 读取图2并输出其像素值，可以明显发现正常区域的像素值基本大于 150，而破损处的像素值集中在 50 以下。

同理，将墨水滴于纸上来模拟污损或斑点，得到图3和图4。



图 3: 实验样例 3



图 4: 实验样例 4 (从样例 3 中截取得到)

采用相同的方法可以得到污损区域的像素值范围。污损与破损像素值的变化有所不同：破损区域像素值的跳变明显；而污损区域的像素值由外到里变化相对平缓（由图4可以看出）。总体而言，污损区域的像素值集中在 120 以下，但后续实验的结果（见第5节）表明，将阈值设为 80 更为合适。

2.2 判断非正常像素点是否连成一个足够大的区域

针对1.2节的问题2，我们采用“深度优先搜索（DFS）”的方法。

2.2.1 算法描述

对于一张已得到的图像，从头开始遍历每一个像素，每当发现当前像素的取值不在正常范围内时，就开始进行深度优先搜索。搜索从该点开始，向其上下左右及主副对角线共 8 个方向进行递归查找。

每次递归时，首先判断该位置是否在图像范围内，其次判断该位置的像素值是否在非正常范围内。只有当上述两个条件均满足时，才继续进行递归。同时为了保证搜索不重复，需要将该点的像素值置为 255。

2.2.2 采用面向对象的方法降低时间复杂度

MATLAB 与 C 语言或 C++ 相比，一个重要的不同之处是 MATLAB 无法进行引用或指针的传递。也就是说我们无法在递归的过程中对原图像矩阵做标记，每次递归时都必须把整个矩阵复制一遍，这样增加了很多不必要的开销。

针对这种情况，我们采用面向对象的方法，将图像矩阵与 2.2.1 节中涉及的函数封装成一个类，这样就可把图像矩阵作为全局变量，从而避免“值传递”产生的额外开销。同时还可以把与之相关的其它特征参数也作为成员变量，如维度、阈值等，这样可以避免反复调用 `size` 等函数，也增加了程序的可移植性。

2.2.3 采用剪枝的方法降低时间复杂度

由于本题的要求仅仅是检测纸带是否有缺陷，所以其实无需把不合规的像素点全都遍历一遍。为此可以再增加两个成员变量：一个用于计数在当前缺陷区域中已经访问过的连续非正常像素点的数量；另一个是针对该数量的阈值，由人事先设定好。当发现连续非正常像素点的数量已经超过了阈值时，就可以判定该纸带是有缺陷的，直接结束整个程序。这样一方面可以大大降低程序运行的时间，减少图片大小及其分辨率对程序运行时间的影响；同时还可以避免当缺陷区域过大时递归层数过多，导致内存溢出的问题。

2.3 排除其它因素的干扰

针对 1.2 节中的问题 3，本文主要考虑两大干扰因素：

1. 背景因素：该问题可以在图像的获取阶段解决，只要保证在得到图像时纸带刚好充满整个画面即可；
2. 阴影因素：该问题可通过调整阈值解决——阴影部分的像素值大小要比缺陷处大。

3 算法设计

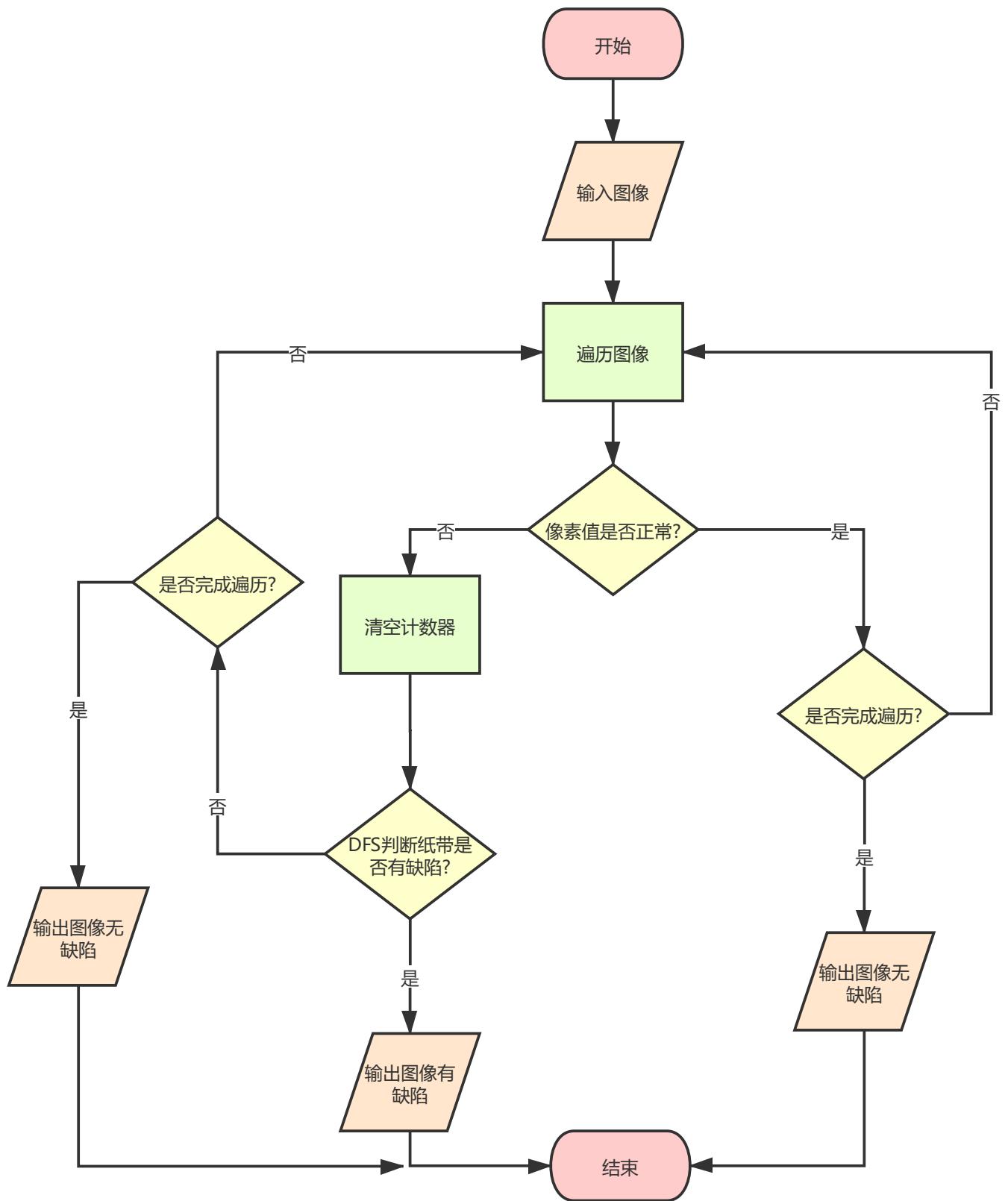


图 5: *main* 函数流程图

4 编程实现¹

Listing 1: boradcast.cpp

```
1 classdef img_detection < handle
2     properties( SetAccess = private , GetAccess = public )
3         img_orig;
4         img;
5         img_gray;
6         maxR;
7         maxC;
8         flag = 0;
9         cnt = 0;
10        thred_of_num = 1500;
11        thred_of_pix = 80;
12    end
13
14    methods
15        function obj = img_detection(im_path)
16            obj.img_orig = imread(im_path);
17            obj.img = obj.img_orig;
18            obj.img_gray = rgb2gray(obj.img);
19            [obj.maxR, obj.maxC] = size(obj.img_gray);
20            obj.flag = 0;
21            obj.cnt = 0;
22            obj.im_detect_defacement
23        end
24
25    methods ( Access = private )
26        function im_count_defacement(obj, row, col)
27            if ( row > obj.maxR || row < 1 || col > obj.maxC || col < 1 )
28
29            elseif ( obj.img_gray(row, col) <= obj.thred_of_pix )
30                %orig = img(row, col);
31                obj.cnt = obj.cnt + 1;
32                if ( obj.cnt < obj.thred_of_num )
33                    obj.img(row, col) = 255;
34                    obj.img_gray(row, col) = 255;
35                    obj.im_count_defacement(row-1, col-1);
36                    obj.im_count_defacement(row-1, col);
37                    obj.im_count_defacement(row-1, col+1);
38                    obj.im_count_defacement(row, col-1);
39                    obj.im_count_defacement(row, col+1);
40                    obj.im_count_defacement(row+1, col-1);
41                    obj.im_count_defacement(row+1, col);
42                    obj.im_count_defacement(row+1, col+1);
43                end
44            end
45        end
```

¹代码也可从该网址获取: https://github.com/chenfeng123456/CourseInOUC/blob/master/Digital_Image_Processing/codes/img_detection.m

```

46         end
47     end
48
49     methods
50         function im_detect_defacement(obj)
51             obj.img = obj.img_orig;
52             obj.img_gray = rgb2gray(obj.img);
53             obj.flag = 0;
54             obj.cnt = 0;
55
56             figure(1);
57             subplot(1, 3, 1);
58             imshow(obj.img_orig)
59             title('原始图像')
60             subplot(1, 3, 2);
61             imshow(obj.img_gray)
62             title('灰度图像')
63
64             for i = 1 : obj.maxR
65                 for j = 1 : obj.maxC
66                     if (obj.img_gray(i, j) <= obj.thred_of_pix )
67                         obj.cnt = 0;%清空计数器
68                         obj.im_count_defacement(i, j);
69                         if (obj.cnt >= obj.thred_of_num-1 )
70                             fprintf("num = %d\n", obj.cnt);
71                             obj.flag = 1;
72                             break;
73                     end
74                 end
75             end
76             if (obj.flag == 1)
77                 break;
78             end
79         end
80         fprintf('flag = %d\n', obj.flag);
81         subplot(1, 3, 3);
82         imshow(obj.img)
83         if (obj.flag )
84             title('有损毁')
85         else
86             title('无损毁')
87         end
88     end
89 end
90
91 end

```

5 结果展示与分析

5.1 结果展示



图 6: 结果 1



图 7: 结果 2

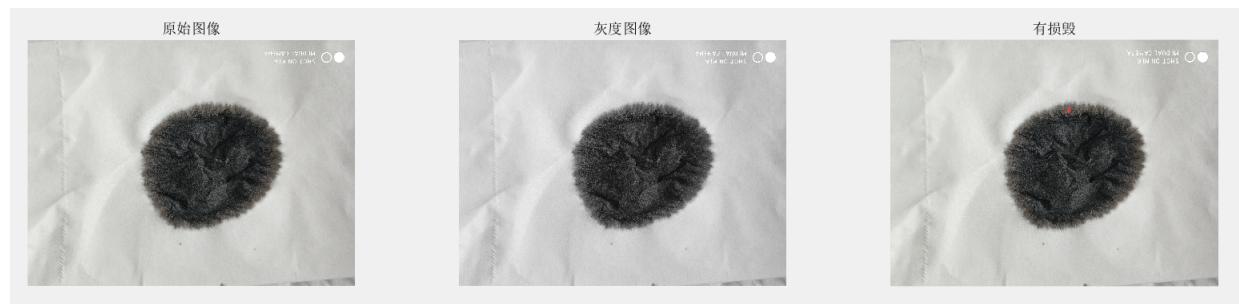


图 8: 结果 3



图 9: 结果 4

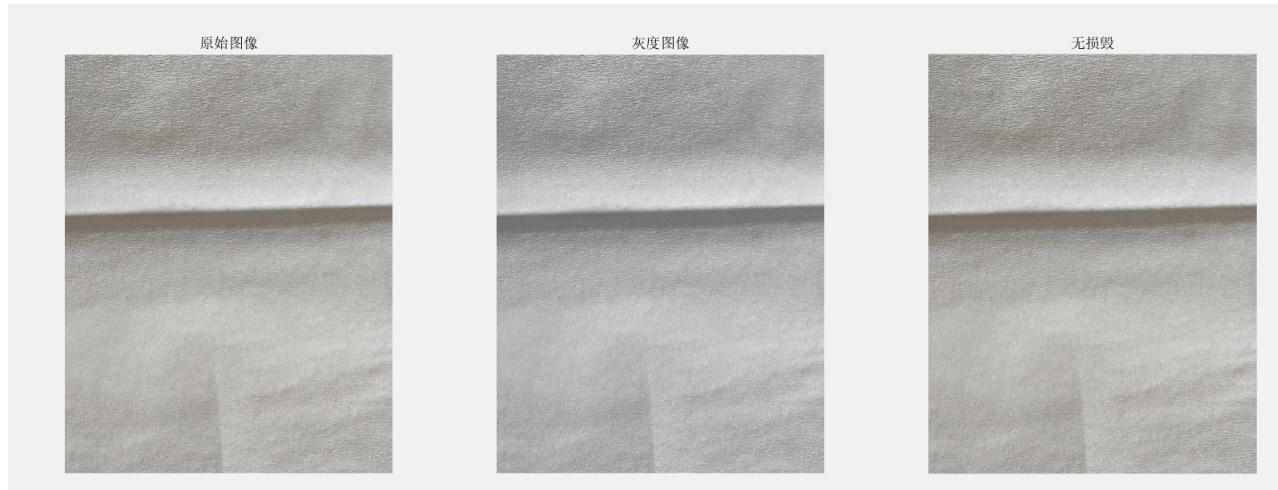


图 10: 结果 5

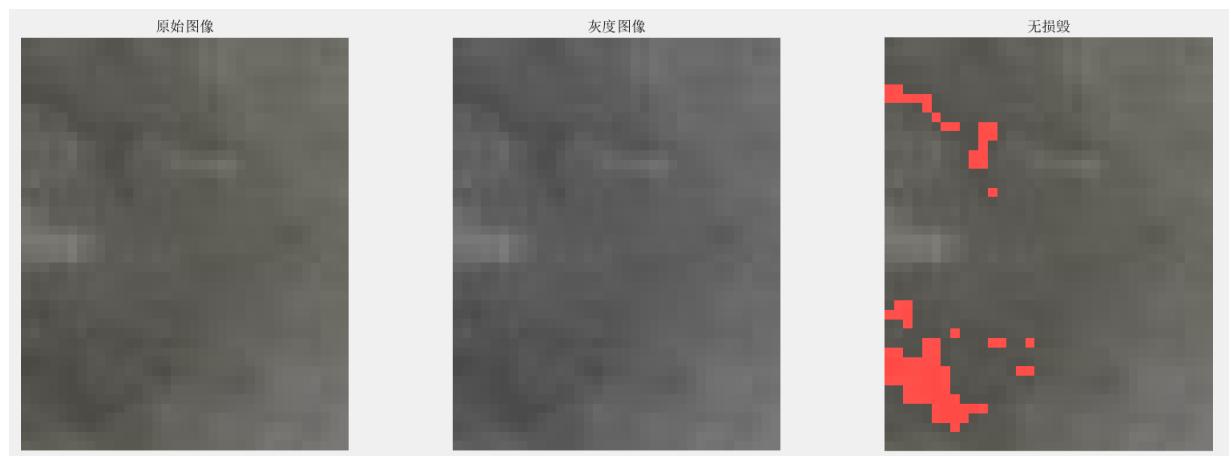


图 11: 结果 6 (截取自结果 5 的阴影部分)

5.2 结果分析

从5.1小节中的结果可以看出，在纸带充满整个图像的情况下，该方案可以有效地检测出纸带是否损毁。尤其是从图11可以看出，在像素阈值为80的情况下，虽然会有部分阴影处被认为是损毁，但不会有连续大面积的误判，证明通过调节阈值大小可以较为有效地排除阴影部分对于检测的影响。

6 总结体会

本次报告所提出的方案在白纸黑背景的情况下能达到较好的效果，但在其它情况下效果如何，受实验条件制约，未能进一步验证。另外，在提出方案的最初，其实是希望能在检测纸带是否有损毁的同时，还能用红色标记出损毁区域。但在本次报告所提的方案中，标出所有损毁区域意味着要用递归遍历所有相关像素点，对于损毁区域较大的样例，会导致内存溢出，故舍弃了该功能。