

重 庆 交 通 大 学

学生实验报告

课 程 名 称： 计算机视觉与模式识别 .

学 院： 信息学院 2021 年级物联网 21 级专业 2 班

学 生 姓 名： 李骏飞 学 号 632109160602 .

指 导 教 师： 蓝 章 礼 .

开 课 时 间： 2023 至 2024 学 年 第 二 学 期

成 绩	
教师签名	

pH 试纸估测综合实验——目录

第一章 绪论	- 3 -
第二章 纯视觉角度判读 pH 试纸的数值	- 4 -
2.1 主要思路	- 4 -
2.2 实现过程	- 5 -
2.2.1 绘制标准比色卡	- 5 -
2.2.2 加权欧式距离法计算最接近的 pH 值-返回整数 ..	- 8 -
2.2.3 插值法计算最接近的 pH 值-浮点数.....	- 12 -
2.2.4 在标准比色卡上标记检测到的 pH 值	- 17 -
第三章 获取目标区域估测 pH 试纸数值	- 20 -
3.1 实验流程	- 20 -
3.2 图像预处理——中值滤波	- 21 -
3.3 RGB 到 HSV 的转化	- 24 -
3.4 目标区域的提取	- 26 -
3.4.1 过滤“中性”部分	- 28 -
3.4.2 过滤“周围环境”部分	- 29 -
3.5 估测 pH 值	- 31 -
3.5.1 计算目标区域的平均 H 分量	- 31 -
3.5.2 计算目标区域的频率分布直方图	- 34 -
3.5.3 计算 pH 值并作图	- 37 -
第四章 总结	- 42 -

第一章 绪论

在当今科学研究和工业生产中，pH 值的准确测量对于评估和控制化学反应、环境监测和医疗诊断等众多领域至关重要。

传统的 pH 测量方法，如使用 pH 试纸，虽然操作简便，但依赖于人眼的主观判断，精度有限，且易受环境因素影响。随着计算机视觉技术的迅速发展，利用图像处理技术对 pH 试纸进行自动读数成为可能，这不仅提高了测量的准确性，也减少了人为误差。

本文旨在探索一种基于**计算机视觉的 pH 试纸数值判读**方法。我们面临的**主要问题**是如何将 pH 试纸的颜色变化转换为可量化的数值，并准确地计算出对应的 pH 值。

为此，我提出了一种初步的解决方案：首先，通过数字图像处理技术采集试纸的颜色信息并将其转换为图像文件；其次，对图像进行预处理，以提高颜色数据的准确性；然后，利用颜色空间转换技术，从 RGB 色彩模式转换到 HSV 色彩模型，提取与 pH 值相关的 H 分量；最后，通过与标准比色卡的比较，计算出最接近的 pH 值。

第二章 纯视觉角度判读 pH 试纸的数值

本章利用计算机视觉的方法依据 PH 标准色卡“精确地”读出 PH 值。不考虑其他因素。

2.1 主要思路

如果从纯视觉角度进行实验的话，思路和方法相对来说就比较简单，首先需要分别获取 pH 标准色卡不同 pH 值所对应的颜色 (RGB 值)，如下图所示，再获取 测量试纸的颜色 RGB 值，利用计算机视觉的办法将试纸颜色与标准色卡颜色逐一比较，找到最接近的颜色，即认为是测量的 PH 值。

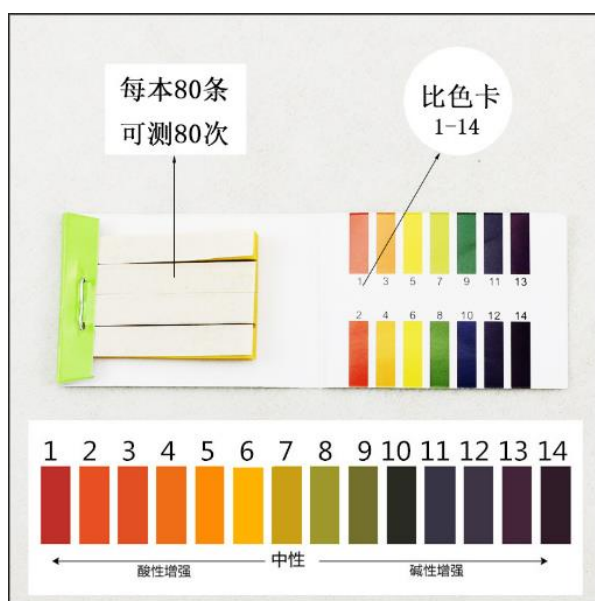


图 2-1 标准 pH 酸碱度对照表

利用 PowerToys 工具下的功能可以获取某一个颜色的 RGB 三通道的分量，安装 PowerToys 软件包之后，使用 shift+windows+c 的快捷键即可获取某一颜色的 RGB 分量，示例图如下：

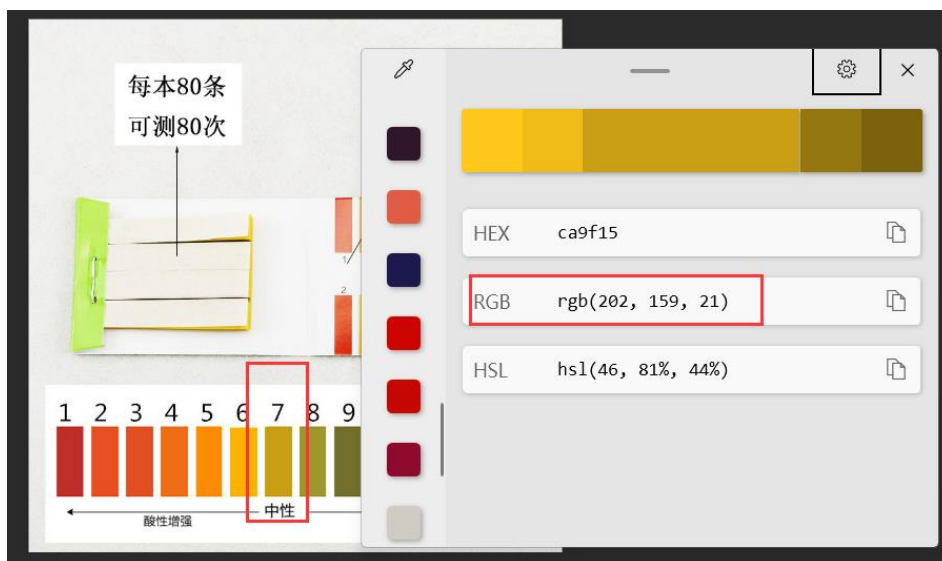


图 2-2 PowerToys 工具获取某一颜色的 RGB

同时为了更加精确,还可以分别获取**最相近**和**次相近颜色**,然后做**数值内插**,即可得到**精确到小数的 pH 值**。

不同颜色的比较通过计算色彩空间中的距离实现。可以分别将 RGB 看成三个轴,从而构成了一个三维的色彩空间,空间中的每一个点即代表一种颜色。通过计算测量的颜色与其它各标准颜色的**欧氏距离**,从而确定**最短距离**。

2.2 实现过程

2.2.1 绘制标准比色卡

网上有不少标准比色卡的参照表,但是网上的图片像素,清晰度很多都是比较模糊,因此我借助 PowerToys 工具,实现**手工获取 pH 试纸不同酸碱度的 RGB 值,制作自己的标准比色卡**。

(1) 获取 14 个酸碱度的 RGB 值

获取 RGB 值

➤ 算法描述:

使用了**列表**来存储颜色值,每个颜色值表示不同 pH 值对应的颜色,例如[41, 46, 191],这个颜色值表示 pH 值为 1 时的颜色,依次执行类似的操作,将其他颜色值添加到 pHColor 列表中,最终这些颜色值以从低到高的 pH 值顺序排列。

需要注意的是,python 中的颜色通道值不是 RGB,而是 **BGR**,因此我

们在创建颜色时需要调整顺序。

```
1. pHColor = [] # B G R pH
2. pHColor.append([41, 46, 191]) # 1
3. pHColor.append([32, 79, 232]) # 2
4. pHColor.append([35, 79, 226]) # 3
5. pHColor.append([23, 109, 239]) # 4
6. pHColor.append([4, 139, 253]) # 5
7. pHColor.append([0, 178, 255]) # 6
8. pHColor.append([21, 159, 202]) # 7
9. pHColor.append([44, 151, 158]) # 8
10. pHColor.append([45, 112, 115]) # 9
11. pHColor.append([34, 42, 41]) # 10
12. pHColor.append([69, 52, 55]) # 11
13. pHColor.append([69, 52, 61]) # 12
14. pHColor.append([57, 36, 68]) # 13
15. pHColor.append([40, 28, 48]) # 14
```

(1) 生成显示不同 pH 值颜色的彩色条图像

算法流程图如下图所示：

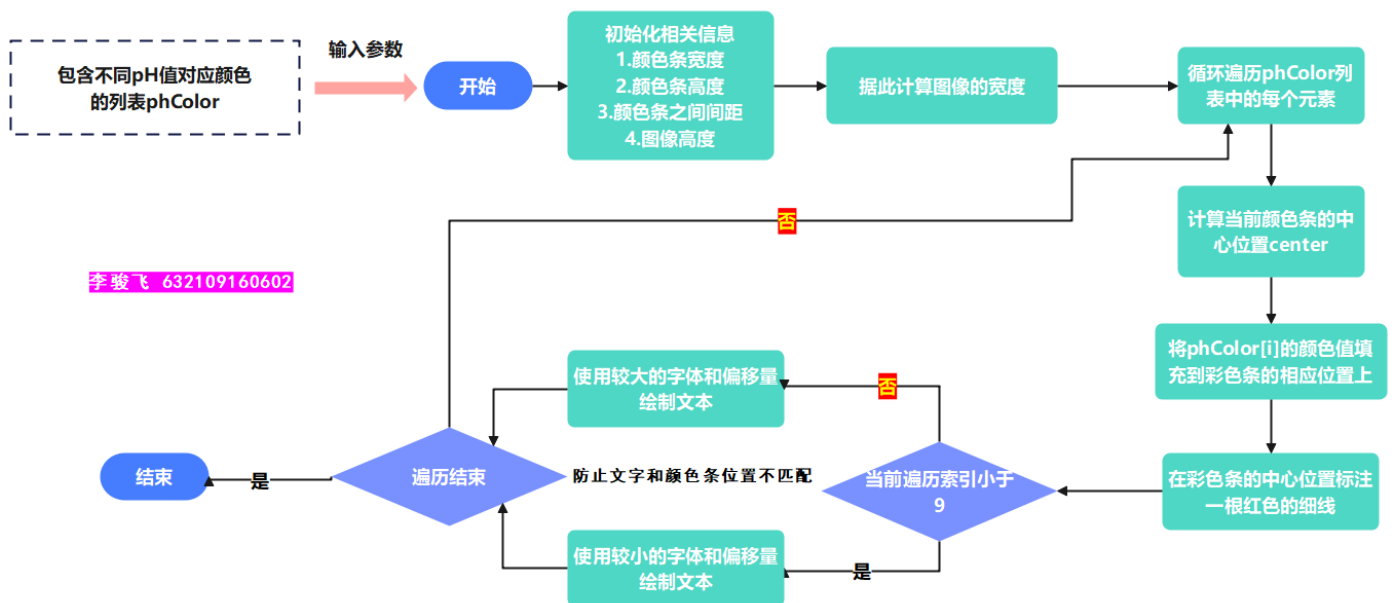


图 2-3 生成标准比色卡的算法流程图

生成彩色条图像

➤ 算法描述：

初始化基本的变量（颜色条宽度、颜色条高度、颜色条之间间距、图像高度等），遍历 pHColor 列表中的每个元素，计算当前颜色条的**中心位置 center**，根据当前索引 i 计算出颜色条的**水平位置**，在创建的空白图像上绘制当前颜色条的矩形区域，即将当前颜色值 pHColor[i] 填充到颜色条的**相应位置**上。

```

1.  # 生成显示不同 pH 值颜色的彩色条图像
2.  # pHColor 是一个包含不同 pH 值对应颜色的列表
3.  def genPHColorPlate(pHColor):
4.      # 颜色条的宽度
5.      color_bar_width = 50
6.      # 颜色条的高度
7.      color_bar_height = 150
8.      # 颜色条之间的间距
9.      color_bar_margin = 20
10.     # 图像的高度
11.     height = 200
12.     # 图像的宽度
13.     width = len(pHColor) * color_bar_width + (len(pHColor) + 1) * color_bar_marg
in
14.     # 创建一个空白图像，全白
15.     blank_img = np.zeros((height, width, 3), np.uint8) + 255
16.     # 循环遍历 pHColor 列表中的每个元素
17.     for i in range(len(pHColor)):
18.         # 计算当前颜色条的中心位置 center
19.         center = color_bar_margin + color_bar_width // 2 + i * (color_bar_width
+ color_bar_margin)
20.         # 将 pHColor[i] 的颜色值填充到彩色条的相应位置上，从而在图像中显示了对应 pH 值的
颜色条
21.         blank_img[color_bar_margin:color_bar_height,
22.                    center - color_bar_width // 2:center + color_bar_width // 2] =
pHColor[i]
23.         blank_img[:color_bar_height, center] = [0, 0, 255]
24.         # 在颜色条下方的位置添加文本，显示当前 pH 值。如果 i 大于等于 9，则使用较小的字体
和偏移量绘制文本，否则使用较大的字体和偏移量绘制文本
25.         if i >= 9:
26.             cv2.putText(blank_img, str(i + 1),
27.                          (center - 22, color_bar_height + 30),
28.                          cv2.FONT_HERSHEY_SIMPLEX,
29.                          1, (0, 0, 0), 1, cv2.LINE_AA)
30.         else:
31.             cv2.putText(blank_img, str(i + 1),
32.                          (center - 12, color_bar_height + 30),
33.                          cv2.FONT_HERSHEY_SIMPLEX,
34.                          1, (0, 0, 0), 1, cv2.LINE_AA)
35.     return blank_img

```

最终效果图如下：

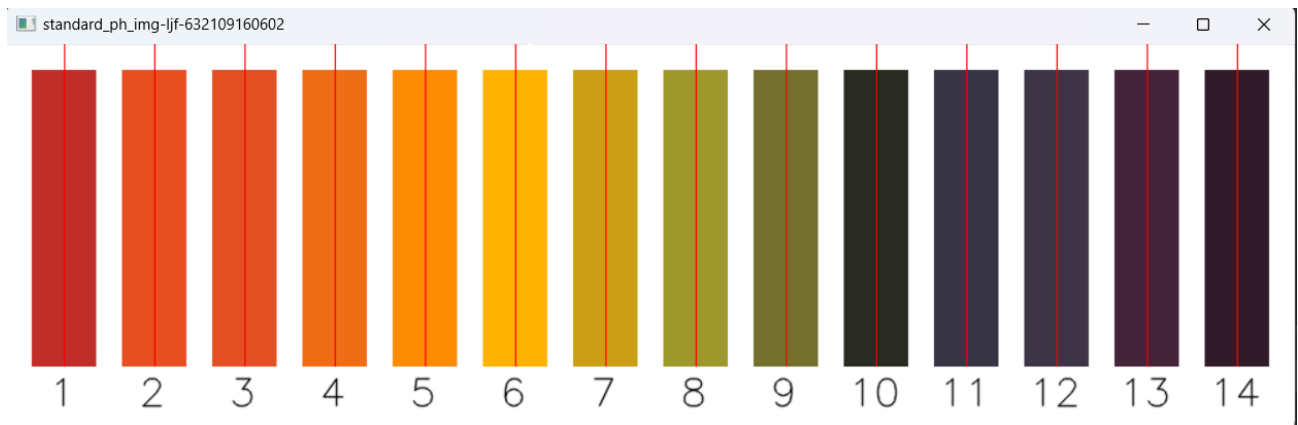


图 2-3 自定义的标准比色卡

2.2.2 加权欧式距离法计算最接近的 pH 值-返回整数

(1) 原理介绍

I:RGB 颜色空间

RGB 颜色空间俗称**三基色模式**。在大自然中有无穷多种不同的颜色，而人眼只能分辨有限种不同的颜色，RGB 模式可表示一千六百万种不同的颜色，在人眼看来它非常接近大自然的颜色，故又称为自然色彩模式。当三原色重叠时，由于不同的混色比例能产生各种中间色。

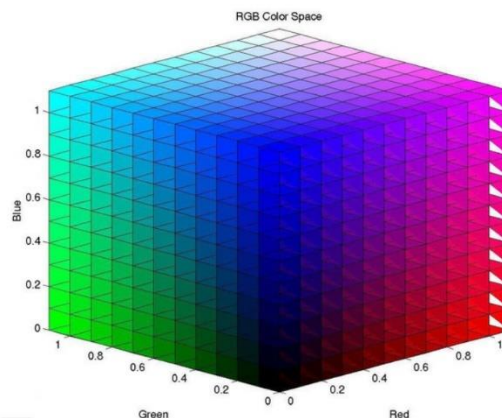


图 2-4 RGB 颜色空间

II:CIE XYZ 颜色空间

颜色的感觉是由于三种原色光刺激的综合结果。在红、绿、蓝三原色系统中，红、绿、蓝的刺激量分别以 R、G、B 表示，形成 RGB 彩色模型。

由于从实际光谱中选定的红、绿、蓝三原色光不可能调配出存在于自

然界的所有色彩，所以为了从理论上来匹配一切色彩并以非负值表示颜色，假设了并不存在于自然界的三种原色，即**理论三原色 XYZ**，相应的形成了 XYZ 颜色空间。

XYZ 三刺激值是由 RGB 彩色空间线性变换转换得到，变换后的空间就是 CIE XYZ 彩色空间，相当于使用匹配颜色的 XYZ 基底来代替 RGB 基底来表示颜色，如图所示，其中 x 和 y 两维定义颜色，第 3 维定义亮度。

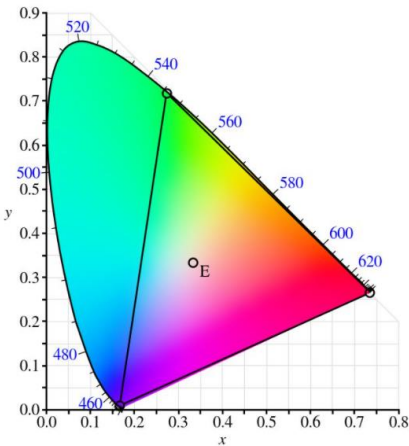


图 2-5 XYZ 颜色空间

III:HSV 颜色空间

HSV 是根据颜色的直观特创建的一种颜色空间，也称六角锥体模型，HSV 即色相 H（Hue）、饱和度 S（Saturation）、明度 V（Value）。

色相是色彩的基本属性，就是平常说的颜色的名称，如红色、黄色等，H 由绕 V 轴的旋转角给定，例如红色对应角度 0° ，绿色对应角度 120° ，蓝色对应角度 240° ，每一种颜色和它的补色相差 180° ；**饱和度（S）**是指色彩的纯度，越高色彩越纯，低则逐渐变灰；**明度（V）**，颜色明亮的程度，明度值与发光体的光亮度有关。圆锥的顶点处， $V=0$ ，代表黑色，圆锥的顶面中心处 $V=1$ ， $S=0$ ，代表白色，其空间模型如图所示。

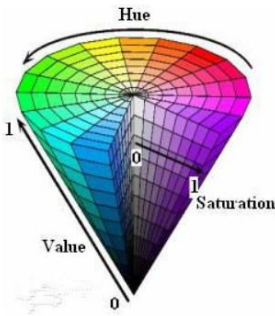


图 2-6 HSV 颜色空间

IV:Lab 颜色空间

Lab 颜色模型基于人对颜色的感觉，其数值描述正常视力的人能够看到的所有颜色。因为 Lab 描述的是颜色的显示方式，而不是设备（如显示器、桌面打印机或数码相机）生成颜色所需的特定色料的数量，所以 Lab 被视为与设备无关的颜色模型。Lab 色彩模型是由亮度（L）和有关色彩的 a、b 三个要素组成，L 表示亮度，a 表示从洋红色至绿色的范围，b 表示从黄色至蓝色的范围。L 的值域由 0 到 100，L=50 时，就相当于 50%的黑色；a 和 b 的值域都是由+127 至-128，其中当 a=+127 时颜色为红色，当 a 渐渐过渡到-128 时变成绿色；同样原理，当 b=+127 是黄色，b=-128 是蓝色，所有的颜色就以这三个值交互变化所组成，其空间如图所示。

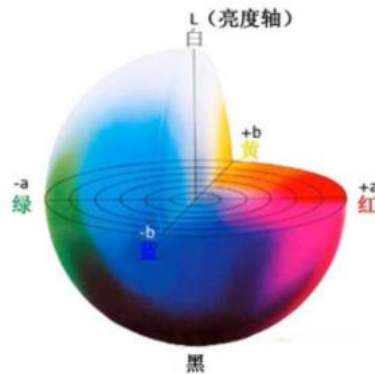


图 2-7 Lab 颜色空间

(2) 颜色距离计算

以上描述了一些常见的颜色空间，那么就有以下一些颜色距离公式

颜色距离指的是两个颜色之间的差距，通常距离越大，两个颜色相差越大，反之，两个颜色越相近。在计算颜色距离时，有类似计算两点间欧式距离的公式一样，以 RGB 空间为例，可以得到两个颜色之间的距离为：

$$||C_1 - C_2|| = \sqrt{(C_{1,R} - C_{2,R})^2 + (C_{1,G} - C_{2,G})^2 + (C_{1,B} - C_{2,B})^2}$$

图 2-8 RGB 的欧式距离

其中， C_1 和 C_2 表示颜色 1 和颜色 2， $C_{1,R}$ 表示颜色 1 的 R 通道，这是最为大众熟知的计算颜色距离的公式，是在 RGB 空间下计算得到的。

✚ 使用欧氏距离度量颜色值与 pH 颜色之间的距离

➤ 算法描述：

该函数有两个输入参数，给定的颜色值 color 和 标准 pH 颜色列表，

然后计算返回**最接近的 pH 值**，具体方法就是遍历 pH 颜色列表，计算每种标准色和给定颜色之间的加权欧式距离，并添加到距离列表中，最后返回**距离最小的标准色的索引**，由于列表的索引从 0 开始，因此最后结果应当加一。

```
1. # 根据给定的颜色值和 pH 颜色列表，计算出最接近的 pH 值
2. # 使用欧氏距离度量颜色值与 pH 颜色之间的距离，并返回最接近的 pH 值的整数
3. def getPhValueInt(color, phColor):
4.     dists = []
5.     for i in range(len(phColor)):
6.         dist = math.sqrt(
7.             pow(color[0] - phColor[i][0], 2) + pow(color[1] - phColor[i][1], 2) +
8.             pow(color[2] - phColor[i][2], 2))
9.         # print(dist)
9.         # dist = ColourDistance(color, phColor[i])
10.        dists.append(dist)
11.    return dists.index(min(dists)) + 1
12.
```

结果示例：

由于该方法是理想状态下的测量结果，因此这里我测试案例使用的是自己给定的颜色 rgb，然后输出最接近的 pH 值，我给定的颜色 rgb 为(255, 115, 34)，颜色图如下：

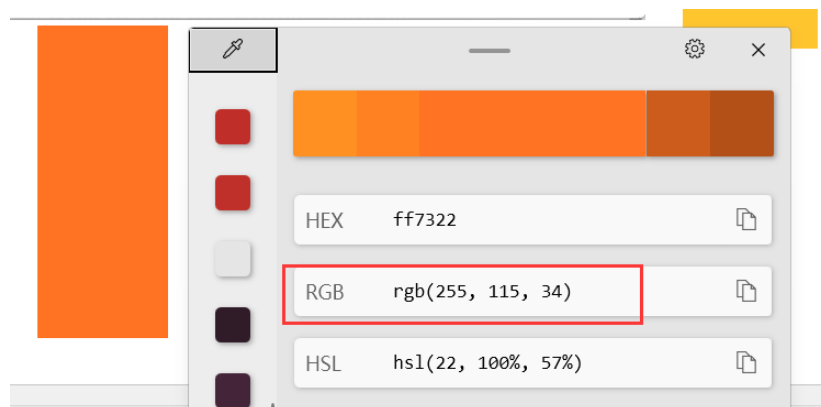


图 2-8 测试颜色

从肉眼上观察与 pH=4 的标准比色卡似乎最接近：

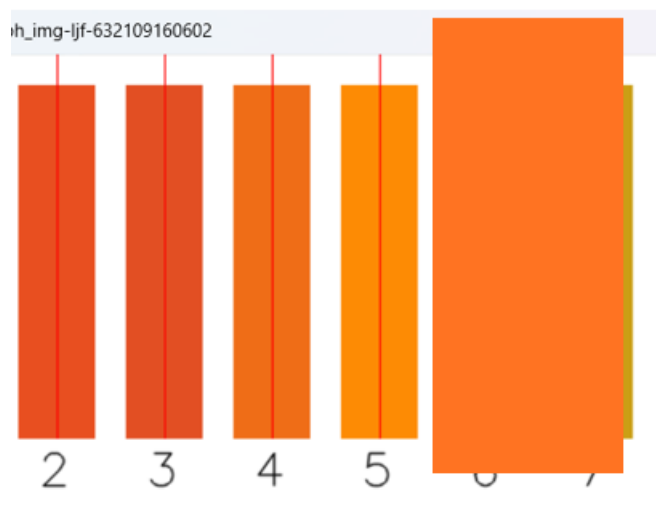


图 2-9 肉眼观察法

输出结果：

```
main x color_distance x
D:\pycharm\Python\python\python.exe D:\AllCode\python\ph\code\main.py
-ljf-632109160602
采用欧式距离计算，与标准比色卡最接近的ph值为：4
```

似乎输出结果跟我们肉眼评估的结果相似，但是仅仅返回整数值的 pH 值误差比较大，在下一小节，我们将使用插值法提高精度，测量更精准的 pH 值

2.2.3 插值法计算最接近的 pH 值-浮点数

(1) 原理介绍

采用插值法计算 pH 的目的是基于已知数据点之间的趋势或连续性，通过插值来估算未知点的 pH 值。与之相比，采用欧式距离计算是一种基于颜色之间的直接距离来评估相似性或差异性的方法。

两种方法区别如下：

插值法的连续性假设：插值法假设数据点之间存在某种连续性或趋势，并利用这种连续性来估计未知点的值。考虑了数据点之间的相对位置和数值变化的关系。这种方法适用于那些基于数据点之间的连续性或趋势进行估计的情况，比如在图像处理中根据相邻像素的颜色来估计中间像素的颜色。

欧式距离的直接度量：欧式距离是一种直接度量两个数据点之间的距

离的方法。它忽略了数据点之间的中间值，仅仅关注它们在特征空间中的位置差异。这种方法适用于那些基于数据点之间的直接差异性进行度量的情况，比如在聚类分析中根据数据点之间的距离来判断它们的相似性或差异性。

在计算 pH 值的情况下，插值法更适合，因为 pH 值通常与颜色之间的连续性或趋势相关。颜色与 pH 值之间可能存在某种关联，通过插值法可以利用已知颜色点的 pH 值的信息来推断未知颜色点的 pH 值。欧式距离在这种情况下可能无法捕捉到颜色与 pH 值之间的关系，因为它仅仅关注颜色之间的直接差异，而不考虑它们的关联性。

(2) 算法讲解

在使用插值法计算最接近的 pH 值的过程中，我们首先通过遍历循环，找到与标准比色卡中的元素欧式距离最小的索引 min_index1 和欧式距离次小的索引 min_index2。

当 $\text{min_index1} \leq \text{min_index2}$ 时，使用线性插值方法。线性插值方法通过使用 $\text{dist1} / (\text{dist1} + \text{dist2})$ 的比例来加权平均最接近的两个 pH 值的索引，从而计算出在这两个 pH 值之间的插值结果。把这个插值结果赋值给 final_pH。

当 $\text{min_index1} > \text{min_index2}$ 时，使用逆向线性插值方法，原理相同。

使用线性插值的优势在于简单且直观，它基于数据点之间的线性关系进行插值，适用于数据点之间存在较为平滑的连续性或趋势的情况。

在 pH 检测中采用这种插值法的好处是能够根据最接近的两个 pH 值之间的距离和相关性来估计给定颜色对应的 pH 值。通过插值法，可以使用已知数据点之间的趋势来推断未知点的 pH 值，从而填补数据的间隙。

据此先做出算法流程图：

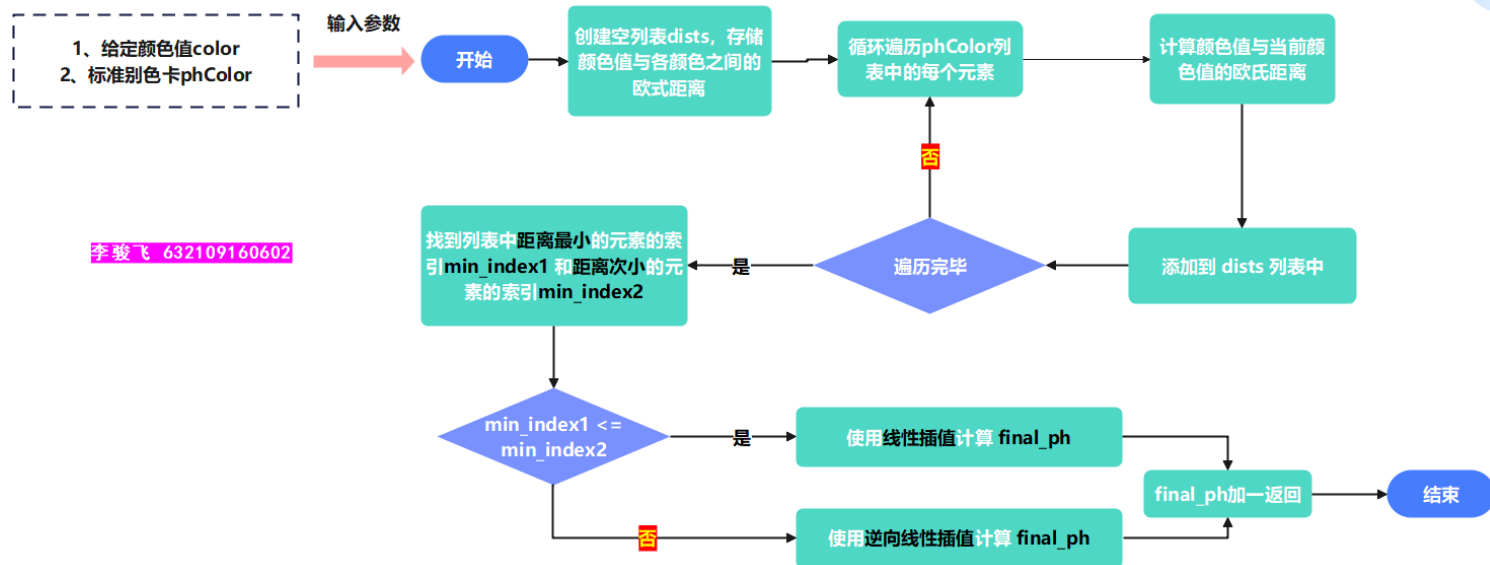


图 2-10 插值法计算 pH 的算法流程图

(3) 代码撰写

通过颜色插值方法获取在两个最接近的 pH 值之间的插值

➤ 算法描述：

具体描述详见 (2)

```

1. def getPHValueFloat(color, pHColor):
2.     # 存储 color 与每个 pHColor 之间的距离
3.     dists = []
4.     # 循环遍历 pHColor 列表
5.     for i in range(len(pHColor)):
6.         # 计算 color 与每个 pHColor 之间的欧氏距离
7.         dist = math.sqrt(
8.             pow(color[0] - pHColor[i][0], 2) + pow(color[1] - pHColor[i][1], 2) +
9.             pow(color[2] - pHColor[i][2], 2))
10.        # 将距离添加到 dists 列表中
11.        dists.append(dist)
12.
13.    # 找到 dists 列表中距离最小的元素的索引, 即与 color 最接近的 pHColor 的索引
14.    min_index1 = dists.index(min(dists))
15.    # 索引赋值
16.    dist1 = dists[min_index1]
17.
18.    # 设为一个较大的值, 以便下一步查找次小距离
19.    dists[min_index1] = 999999
20.
21.    # 找到列表中距离次小的元素的索引
22.    min_index2 = dists.index(min(dists))
23.    dist2 = dists[min_index2]
24.
25.    # 通过插值计算出在这两个最接近的 pH 值之间的插值
26.    if min_index1 <= min_index2:
27.        final_pH = min_index1 + abs(min_index1 - min_index2) * (dist1 / (dist1 + d
28.            ist2))
29.    else:
30.        final_pH = min_index1 - abs(min_index1 - min_index2) * (dist1 / (dist1 + d
31.            ist2))
32.    return final_pH + 1, min_index1 + 1, min_index2 + 1

```

实现效果:

与使用加权欧式距离计算的 pH 值相比, 插值法计算出来的是浮点数, 精度更高。同样采用 BGR 为(34, 115, 255)的待检测颜色, 可见插值法精度更高, 更准确。

```

main x color_distance x
D:\pycharm\Python\python\python.exe D:\AllCode\python\ph\code\m
-ljf-632109160602
采用欧式距离计算, 与标准比色卡最接近的ph值为: 4
采用插值法计算, 与标准比色卡最接近的ph值为: 4.345659225149619
4
5

```

图 2-11 插值法

多次测试结果:

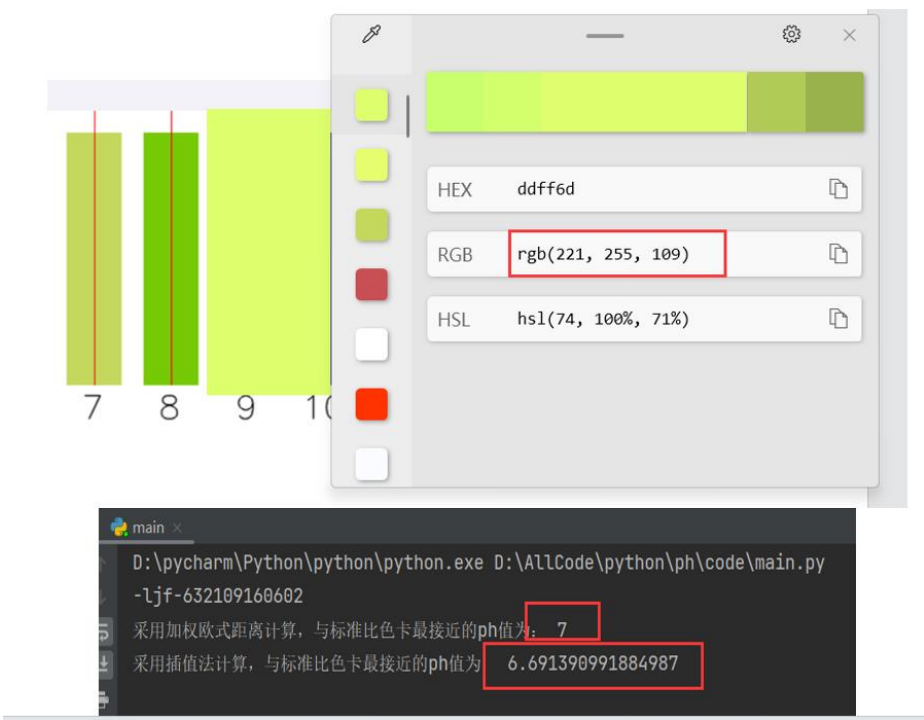


图 2-12 测试结果 1



图 2-13 测试结果 2

测试结果似乎同肉眼观测法观察结果相似，精度似乎也足够，说明从纯视觉角度来看，不管是**欧式距离**计算 pH，还是**插值法**计算 pH，准确度似乎都能得到保证。

2.2.4 在标准比色卡上标记检测到的 pH 值

为了更直观的看到待检测的颜色在 pH 试纸上的分布，我们将检测到的 pH 标记在我们绘制的标准比色卡之上

由于此部分代码并不是算法关键所在，因此不详细介绍，代码如下：

在比色卡上绘制检测到的 pH 值

➤ 算法描述：

先定义与颜色条绘制相关的参数（颜色条宽度 `color_bar_width`、颜色条与边界的间距 `color_bar_margin`、颜色条高度 `color_bar_height`），将待标记的 pH 值减去 1，赋值给变量 pH（pH 值从 0 开始索引）。

然后将最接近的两个已知 pH 值减去 1，赋值给变量 pH1 和 pH2。如果 pH1 大于 pH2，则交换它们的值，确保 pH1 表示较小的 pH 值，pH2 表示较大的 pH 值。

根据已知 pH 值在颜色条上的位置，计算待标记 pH 值在颜色条上的位置。由 pH 值的位置和颜色条的宽度和间距来计算，使用了**线性插值**的思想计算开始位置和结束位置的像素值，。

将待标记的 pH 值在颜色板图像上的位置进行标记，即将 `pH_img` 中对应位置的像素值设置为 `pH_color`，这样就在颜色条上标记了待标记的 pH 值。

使用 `cv2.putText` 函数在颜色板图像上绘制 pH 值的文本信息。文本信息包括“PH=”和**待标记的 pH 值**，位置在待标记位置的右上方。

返回绘制完成的颜色板图像即可。

```

1. # 函数会在彩色条图像上标注检测到的 pH 值，并将对应的颜色标记出来
2. def drawPH(pH_color, pH_val, pHColor):
3.     pH_img = genPHColorPlate(pHColor)
4.     color_bar_width = 50
5.     color_bar_margin = 20
6.     color_bar_height = 150
7.     pH = pH_val[0] - 1
8.     pH1 = pH_val[1] - 1
9.     pH2 = pH_val[2] - 1
10.    if pH1 > pH2:
11.        tmp = pH1
12.        pH1 = pH2
13.        pH2 = tmp
14.    start_pix = color_bar_margin + color_bar_width / 2 + pH1 * (color_bar_width
+ color_bar_margin)
15.    end_pix = color_bar_margin + color_bar_width / 2 + pH2 * (color_bar_width +
color_bar_margin)
16.    loc = int((pH - pH1) * (end_pix - start_pix) + start_pix)
17.    pH_img[:, loc - 1:loc + 1] = pH_color
18.    cv2.putText(pH_img, "PH=" + round(pH_val[0], 2).__str__(),
19.                (loc + 1, color_bar_height + 45),
20.                cv2.FONT_HERSHEY_SIMPLEX,
21.                0.4, (0,0,0), 1, cv2.LINE_AA)
22.    return pH_img

```

示例结果如下图：

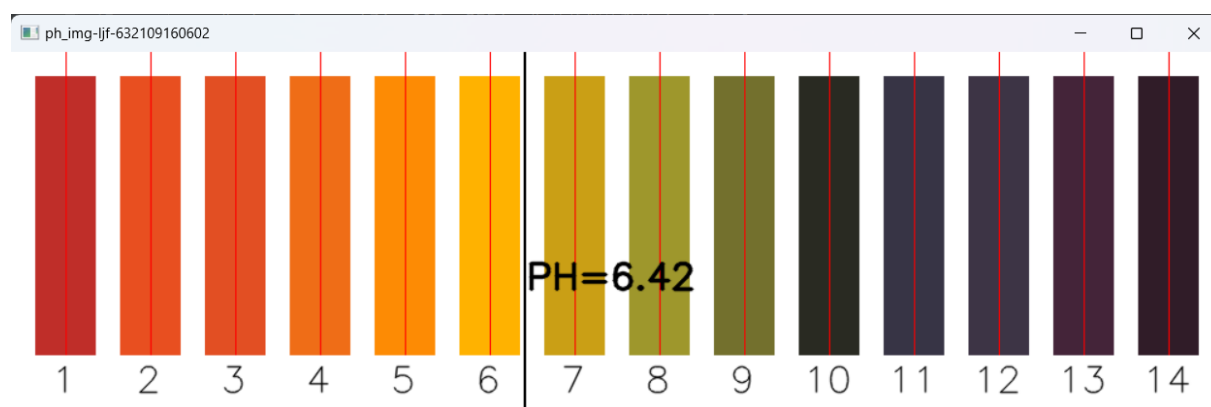


图 2-14 绘制比色卡上

更多测试图如下：

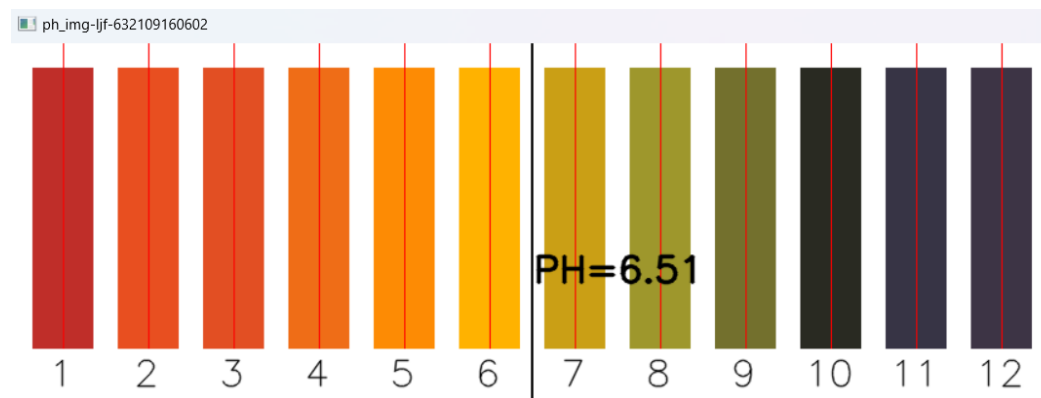


图 2-15 测试 2

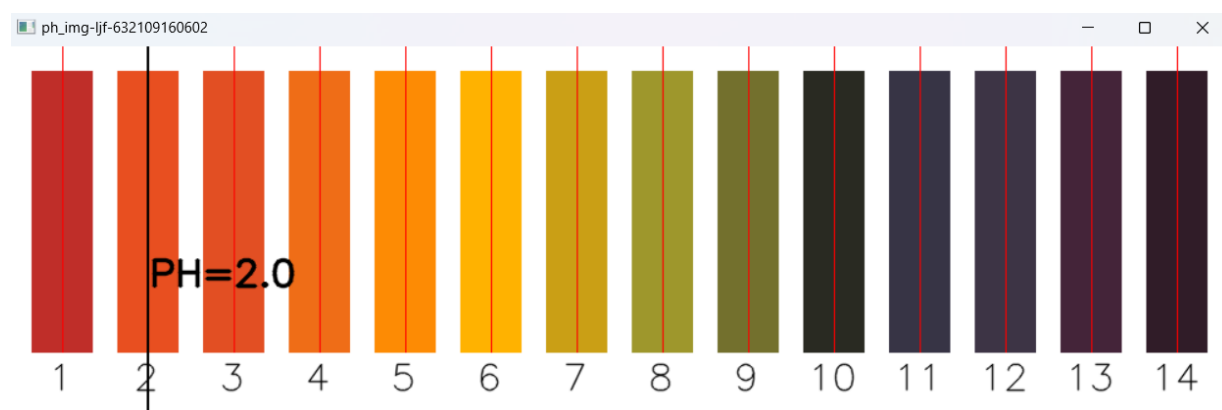


图 2-16 测试 3

第三章 获取目标区域估测 pH 试纸数值

在第二章中，我从纯视觉角度对 PH 试纸的数值判读进行了实现，其本质上类似于测量里的“估读”，提供了 pH 测量读数的另一种思路和方法。相比于人工比色只能比到整数而言，利用代码可以精确到小数，同时也可以避免一些人为的主观因素对结果造成的影响，如色弱、色盲等。

但由于我们前提条件是理想情况，即采集过程中无角度，光照等其他因素的影响，因此其准确性还有待进一步检验，在实际测量过程中，环境光对色彩的影响也不可忽视，而且 pH 值本身也并非线性变化，在不同 PH 值之间采用线性内插是否科学也有待进一步验证。

因此在第三章中，我根据若干张真实的 pH 试纸图片，试图寻找方法能估读出其 pH 值，pH 试纸图片如下：



图 3-1 未知溶液的 pH 试纸检测结果

3.1 实验流程

本章节的研究思路是通过数字图像处理技术模拟人的视觉，识别待测溶液试纸的颜色，从而判定 pH 值。

具体的过程是用数字图像处理技术去采集试纸的颜色信息，形成包含颜色信息的图像文件，然后选取出合适的区域进行中值滤波预处理，以避免原图像中存在的噪声干扰，提高颜色数据的准确性，随后将 RGB 的颜色空间转换成 HSV 的颜色空间，借助于 HSV 色彩模型中 H 分量值与 pH 试纸颜色之间的对应关系，对其进行分析，得到待测未知溶液的 H 值。其整体流程结构如图所示。



图 3-2 整体流程

3.2 图像预处理——中值滤波

(1) 原理解释

图像滤波是图像预处理中必不可少的过程, 将对其后图像处理和分析的可靠性和有效性造成很直接的影响。因此, 在尽力保留图像细节特征的情况下需要抑制噪声, 采用的方法是**中值滤波**。

中值滤波是一种非线性的图像平滑方法, 基本原理是将数字序列或图像里一点的值用该点的一个域中各点值的中值替代, 让四周的像素值接近真实值, 从而消除掉孤立的噪声点。方法是用某种结构的二维滑动模板, 把板内像素按照像素值的大小进行排序, 生成单调下降 (或上升) 的二维数据序列。

(2) 算法步骤

中值滤波的步骤为:

(1) 将滤波窗口 (即包含有若干个点的滑动模板) 在图像中来回滑动, 并将窗口中心与图像中某一个像素点的位置进行重合。

(2) 分别读取模板中各对应像素点的红、绿、黄三颜色分量的数据。

(3) 将这 3 种颜色单独按照其各自的像素值从小到大排列。

(4) 取这 3 列数据的中间数据, 并将其赋给对应模板中心位置像素点的红、绿、黄 3 个分量。如果窗口中包含有奇数个元素, 把元素按颜色分量像素

值大小进行排序后的中间元素像素值作为最终的中值；如果窗口中包含有偶数个元素，将元素按颜色分量像素值大小排序后，中间 2 个像素值的平均值作为中值。

根据上述步骤，我绘制的算法流程图如下：

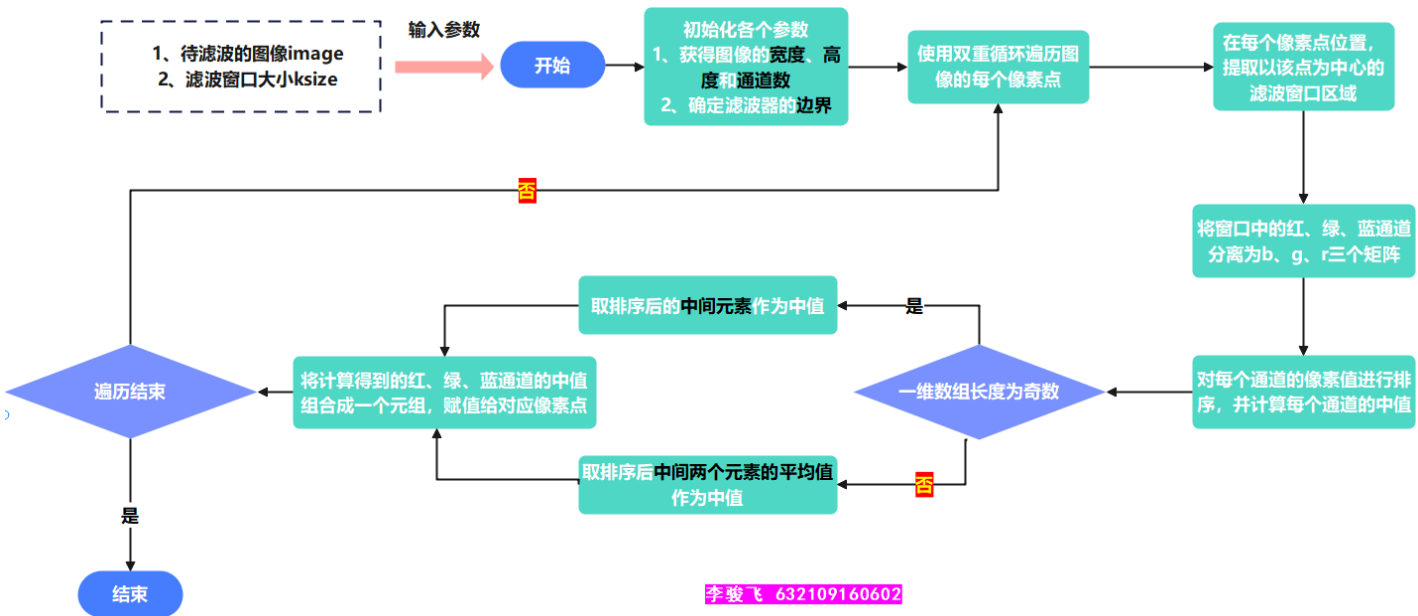


图 3-3 中值滤波的算法流程图

(3) 算法编写

自定义中值滤波

➤ 算法描述：

通过遍历图像中的每个像素点，以滤波窗口为单位，按照中值滤波的算法原理计算滤波后的像素值，并将其赋给滤波后图像对应位置的像素点。最终实现了自定义的中值滤波算法。具体算法原理描述见上（1）和（2）

```

1. # 自定义中值滤波
2. def median_filter(image, ksize):
3.     height, width, channels = image.shape
4.     filtered_image = np.zeros_like(image)
5.
6.     padding = ksize // 2
7.
8.     for y in range(padding, height - padding):
9.         for x in range(padding, width - padding):
10.            window = image[y - padding:y + padding + 1, x - padding:x + padding +
11.            1]
12.
13.            # 分离窗口中的红、绿、蓝通道
14.            b, g, r = cv2.split(window)
15.
16.            # 将每个通道的像素值按从小到大排序
17.            b_sorted = np.sort(b.flatten())
18.            g_sorted = np.sort(g.flatten())
19.            r_sorted = np.sort(r.flatten())
20.
21.            # 计算中值
22.            b_median = b_sorted[len(b_sorted) // 2] if len(b_sorted) % 2 == 1 else
23.            (b_sorted[len(b_sorted) // 2 - 1] + b_sorted[len(b_sorted) // 2]) // 2
24.            g_median = g_sorted[len(g_sorted) // 2] if len(g_sorted) % 2 == 1 else
25.            (g_sorted[len(g_sorted) // 2 - 1] + g_sorted[len(g_sorted) // 2]) // 2
26.            r_median = r_sorted[len(r_sorted) // 2] if len(r_sorted) % 2 == 1 else
27.            (r_sorted[len(r_sorted) // 2 - 1] + r_sorted[len(r_sorted) // 2]) // 2
28.
29.            # 将中值赋给对应位置的像素点
30.            filtered_image[y, x] = (b_median, g_median, r_median)
31.
32.    return filtered_image

```

示例效果图如下：



图 3-4 中值滤波示例图

3.3 RGB 到 HSV 的转化

数字图像处理中，照相机采集到的图片是 **RGB 色彩模式**。它是一种最基本常用的颜色标准，通过对红、绿、蓝 3 个颜色通道的变化，以及它们相互之间的叠加，得到其他各种各样的颜色，但是这种色彩模式对亮度比较敏感，很容易受到阴影、光照和遮挡等情况的影响。一旦亮度发生改变，RGB3 个分量都会随之发生改变，而且 RGB 色彩模型中的任意一种颜色是 RGB3 个分量的共同作用，它们之间的相关性很高，无法用单一的参数进行数字化调整。

因此，为了能够在环境复杂的情况下快速提取出 pH 值的颜色分量，同时又符合人工视觉的特点，本文使用 **HSV 色彩模型**。HSV 是一个倒锥形模型，它由色调 (H)，饱和度 (S)，明度 (V) 3 个参数构成。

从 RGB 到 HSV 的转换是一个简单的非线性变换，获取 H 分量的转换过程如下：

获取 HSV 模型中的色调 H

➤ 算法描述：

根据转换公式而来。

```
1. # 获取 HSI 模型中的色调 H
2. def get_hsi_h(color):
3.     B = color[0]
4.     G = color[1]
5.     R = color[2]
6.     CMax = max(R, G, B)
7.     CMin = min(R, G, B)
8.
9.     delta = CMax - CMin
10.    rt_val = 0
11.
12.    if delta == 0:
13.        return 0
14.    elif CMax == R:
15.        GB = (int(G) - int(B)) / delta
16.        rt_val = GB * 60
17.        if G < B:
18.            rt_val += 360
19.    elif CMax == G:
20.        BR = (int(B) - int(R)) / delta
21.        rt_val = BR * 60 + 120
22.    elif CMax == B:
23.        RG = (int(R) - int(G)) / delta
24.        rt_val = RG * 60 + 240
25.
26.    return rt_val
```


网络上有很多 pH 值的标准比色卡照片，但是因为灯光等种种原因，照片并不是很清晰或者不准确，我选取了一张较为清晰的 pH 值标准比色卡作为我判读 pH 值的依据，标准比色卡如下：

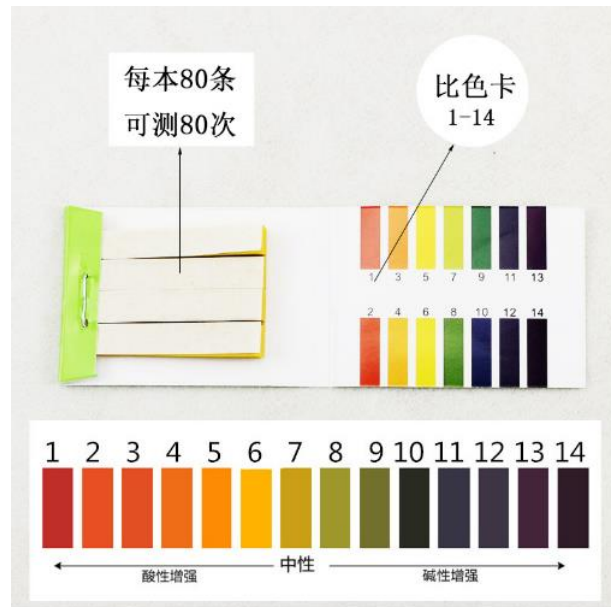


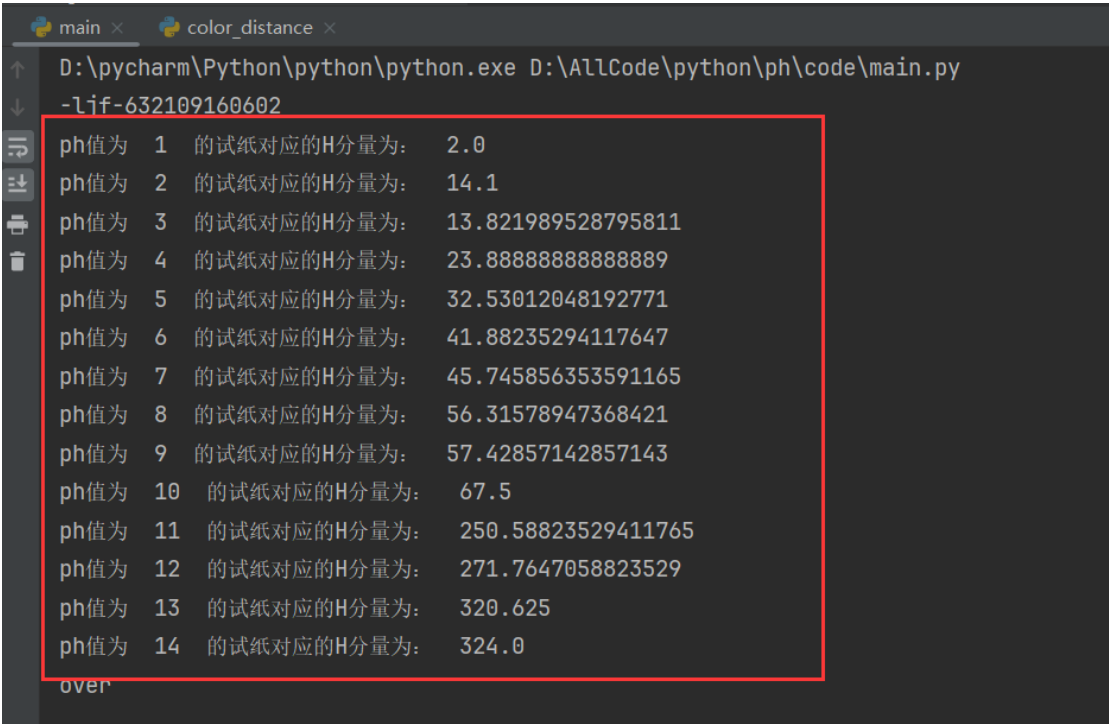
图 3-5 标准比色卡（图片来自网络）

根据标准比色卡，计算出 pH 值与 H 值分量的对应关系：

✚ 计算 pH 值与 H 分量的对应关系

```
1. pHColor.append([41, 46, 191])    # 1
2. pHColor.append([32, 79, 232])    # 2
3. pHColor.append([35, 79, 226])    # 3
4. pHColor.append([23, 109, 239])    # 4
5. pHColor.append([4, 139, 253])     # 5
6. pHColor.append([0, 178, 255])     # 6
7. pHColor.append([21, 159, 202])    # 7
8. pHColor.append([44, 151, 158])    # 8
9. pHColor.append([45, 112, 115])    # 9
10. pHColor.append([34, 42, 41])     # 10
11. pHColor.append([69, 52, 55])     # 11
12. pHColor.append([69, 52, 61])     # 12
13. pHColor.append([57, 36, 68])     # 13
14. pHColor.append([40, 28, 48])     # 14
15.
16. # 不同 pH 对应的 H 分量
17. for i in range(14):
18.     # 获取 H 分量的值
19.     print("pH 值为 ", i + 1, " 的试纸对应的 H 分量
    为: ", bb.get_hsi_h(pHColor[i]))
```

输出结果为：



```
main x color_distance x
D:\pycharm\Python\python\python.exe D:\AllCode\python\ph\code\main.py
-Ljf-632109160602
pH值为 1 的试纸对应的H分量为: 2.0
pH值为 2 的试纸对应的H分量为: 14.1
pH值为 3 的试纸对应的H分量为: 13.821989528795811
pH值为 4 的试纸对应的H分量为: 23.8888888888889
pH值为 5 的试纸对应的H分量为: 32.53012048192771
pH值为 6 的试纸对应的H分量为: 41.88235294117647
pH值为 7 的试纸对应的H分量为: 45.745856353591165
pH值为 8 的试纸对应的H分量为: 56.31578947368421
pH值为 9 的试纸对应的H分量为: 57.42857142857143
pH值为 10 的试纸对应的H分量为: 67.5
pH值为 11 的试纸对应的H分量为: 250.58823529411765
pH值为 12 的试纸对应的H分量为: 271.7647058823529
pH值为 13 的试纸对应的H分量为: 320.625
pH值为 14 的试纸对应的H分量为: 324.0
over
```

图 3-6 输出结果

本实验中，每一个 pH 值对应着一种颜色，而每一种色彩都对应着唯一的 H 分量，这就为图像检测未知溶液的 pH 值提供了依据。由图 3-6 所得 pH 值 0~14 相对应的 H 分量值如下表所示。

表 1 pH 值 0~14 相对应的 H 分量值

pH	1	2	3	4	5	6	7
H 分量	2.000	14.1	13.8219	23.8888	32.5301	41.8823	45.7458
pH	8	9	10	11	12	13	14
H 分量	56.3157	57.4285	67.5000	250.5882	271.7647	320.6250	324.0000

3.4 目标区域的提取

本实验中，得到待测溶液的 pH 试纸后，需要提取出目标区域。但是可能由于某些原因，造成并不是全部的 pH 试纸发生颜色色变，只是其中的一小部分区域，这就需要将显色反应的地方选择出来。

因此我们需要在获取的图片上，选择出一个合适的点，得到它的横坐标 x_1 、纵坐标 y_1 ，之后再选取另外一个点，同样可以知道其横、纵坐标值 x_2 和 y_2 。以

(x_1, y_1) 和 (x_2, y_2) 为对角顶点，裁剪得到一个矩形区域图形，此为**目标区域**。提取矩形区域的长度为 $|x_1 - x_2|$ 、宽度为 $|y_1 - y_2|$ ，即为待测图片。

起初我上网搜索了一些 pH 试纸条的检测结果示例，找到了一些图片，发现这些图片大多数情况下 pH 试纸条的位置并不是一个“很正”的位置，即照片中的区域总是沿着 x, y, z 三个轴都有**倾斜**(如下图),要想把照片翻转到**平行位置**，就需要进行**透视变换**，而透视变换需要同一像素点变换前后的坐标。因此就需要提取矩形区域四个角的坐标作为变换前的坐标，得到四个角的坐标也就需要得到矩形的边缘，如此才能实现。



图 3-7 倾斜的 pH 试纸

最初的想法实现流程如下（失败）：

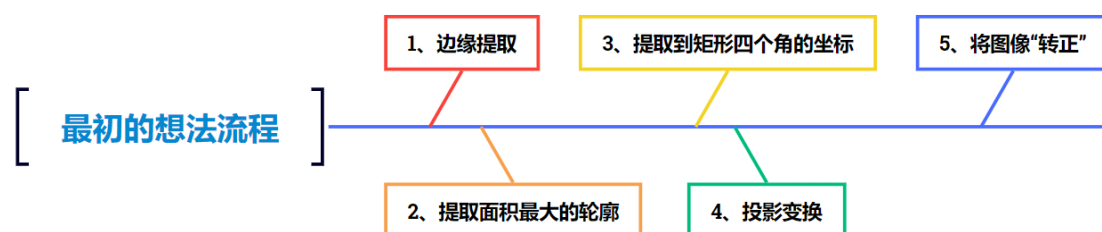


图 3-8 初步流程图

但经过几天反反复复的尝试之后，我很快就放弃的这个想法，因为我在第一步**边缘检测**就遇到了很大的**挫折**，由于图片的清晰度，试纸的变色区域，变色质量等问题，检测到的边缘有时候千奇百怪，需要不停的调整边缘提取的参数，才能够得到符合这张图片的一个边缘，而后的处理更是困难重重。遂经过几天的尝试之后**放弃该方案**。

而后经过一系列摸索，得到了我的一套**目标区域提取的方案**，将在以下子章节进行具体描述。

3.4.1 过滤“中性”部分

首先是第一步，将 pH 试纸条未变色的部分过滤掉，由于我们通常更需要检测 pH 的酸性和碱性部分，往往中性值并不需要特别的判定，且中性变色的部分与试纸条本身的颜色非常接近，因此会在一定程度上影响我们的图像识别，所以我放弃了中性部分的检测，专注酸碱性 pH 值的测定。

过滤黄色部分

➤ 算法描述：

可以知道的是，pH 试纸条本身的颜色是黄色，pH=7 和 pH=8 即中性环境下变色区域非常接近黄色，因此我规定了两个阈值，low_threshold 和 high_threshold，以 HSV 分量中的 H 作为检测标准，如果 H 在两个阈值之间，我就认为是黄色，就将其赋值为黑色，过滤掉。

```
1. # 将黄色区域删除
2.     low_threshold = 30
3.     high_threshold = 58
4.     image_del_yellow = image.copy()
5.     # 遍历图像中的每个像素[
6.     for x in range(height):
7.         for y in range(width):
8.             # 获取像素的 RGB 值
9.             b, g, r = image[x, y]
10.            color = (b, g, r)
11.            # 获取 h
12.            h = bb.get_hsi_h(color)
13.            if( h > low_threshold and h < high_threshold):
14.                image_del_yellow[x, y] = (0, 0, 0)
15.    cv2.imshow("image_del_yellow", image_del_yellow)
```

实现效果如图所示：

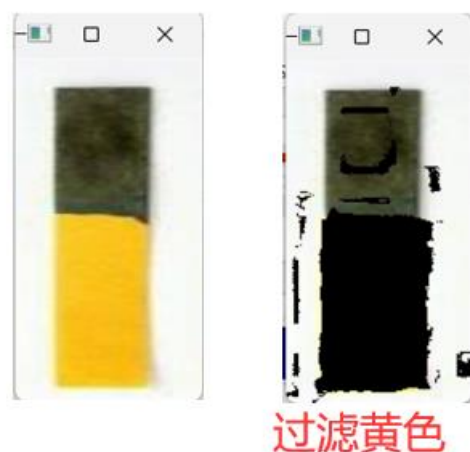


图 3-9 过滤示例图

其他测试图如下所示：



图 3-10 其他测试样例

可以看到黄色基本已经被过滤掉了，虽然在变色区域也有一部分被置为黑色，但是并不影响我们后续的判别。

3.4.2 过滤“周围环境”部分

在过滤掉黄色部分之后，我们希望只关注于变色部分，那么自然周围的环境部分就不需要我们关心。

因为检测 pH 所处环境的不同，可能是在桌面上，可能是在地板上，可能是在白纸上等等。因此周围环境的筛选就需要我们注意，这里我并未考虑每一种情况，而是根据 pH 值的 RGB 特点来筛选。

从 pH=1 到 pH=14，我发现其 RGB 值差不多都有一个共同特点，即通常不会三个值都特别大，往往是某一个通道值较大（大于 100，三位数），另一

个通道值较小（小于 100，两位数），因此我根据这个特点编写了代码来过滤掉“周围环境”部分。

排除周围环境

➤ 算法描述：

通过遍历图片的每一个像素，获取每个像素的 RGB 值，如果 RGB 同时大于规定的 150 阈值的话，那么就认为这是周围环境，就将其置为黑色。

```
1. # 将周围区域删除
2. image_del_round = image_del_yellow.copy()
3. # 遍历图像中的每个像素
4. for x in range(height):
5.     for y in range(width):
6.         # 获取像素的 RGB 值
7.         b, g, r = image[x, y]
8.         if b > 150 and g > 150 and r > 150:
9.             image_del_round[x, y] = (0, 0, 0)
10. cv2.imshow("image_del_round", image_del_round)
```

实现效果如图所示：



图 3-11 过滤环境部分

其他测试图像如下所示：

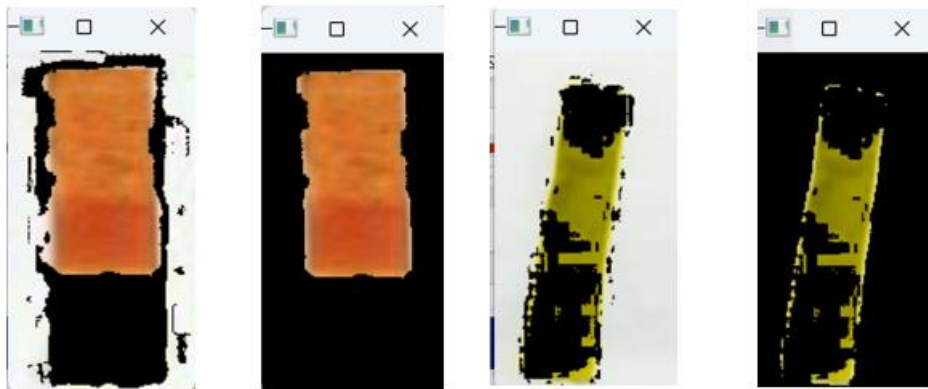


图 3-12 其他测试用例

可以看到周围的空白部分已经被过滤掉了，只保留了目标区域，但是我们发现在目标区域周围存在一些细微的边缘未能删除，因此在需要再次优化一下。

3.5 估测 pH 值

3.5.1 计算目标区域的平均 H 分量

(1) 流程讲解

在得到目标区域之后，我们便可以通过计算目标区域的平均 H 分量，从而判断其属于哪一种色彩分区，从而判读其 pH 值。

由于在上一小节中目标区域的边缘部分存在**瑕疵**，因此需要优化。这里我的做法如下：

由于在目标区域中，可能仍存在一定的颜色过浅或者颜色过深，因此我这里采用“去头去尾”的方法，即计算目标区域每个像素的 H 分量，经过从小到大排序后，去掉前 10%的较小的 H 分量值和后 10 较大的 H 分量值，保存中间部分，从而进行后续计算。

据此我画出的流程图如下：

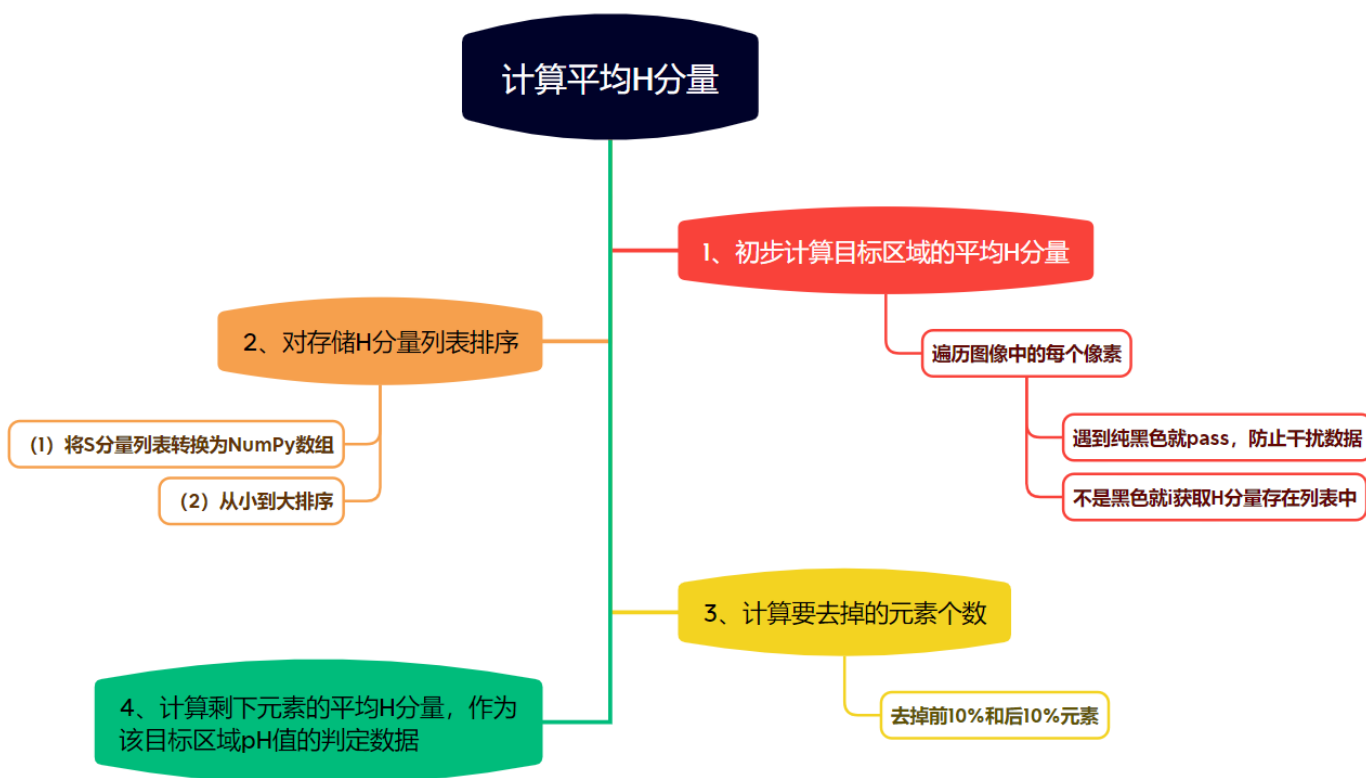


图 3-13 计算目标区域平均 H 分量的算法流程图

(2) 代码编写

🌈 计算目标区域平均 H 分量

➤ 算法描述：

通过遍历图片的每一个像素，获取每个像素的 RGB 值，如果 RGB 同时大于规定的 150 阈值的话，那么就认为这是周围环境，就将其置为黑色。


```

1. # 计算一个列表的平均值
2. def calculate_average(lst):
3.     if len(lst) == 0:
4.         return 0 # 处理空列表的情况, 返回 0 或其他默认值
5.
6.     total = sum(lst) # 计算列表中所有元素的总和
7.     average = total / len(lst) # 计算平均值
8.     return average
9.
10. # 优化
11. image_youhua = image_del_round.copy()
12. # 存储 H 分量的列表
13. h_values = []
14. # 遍历图像中的每个像素
15. for x in range(height):
16.     for y in range(width):
17.         # 获取像素的 RGB 值
18.         b, g, r = image_youhua[x, y]
19.
20.         # 全黑 pass
21.         if b == 0 and g == 0 and r == 0:
22.             pass
23.
24.         # 获取 h
25.         h = bb.get_hsi_h(color)
26.
27.         # 将 h 分量添加到列表中
28.         h_values.append(h)
29.
30. # 将 S 分量列表转换为 NumPy 数组
31. h_values = np.array(h_values)
32. # 对列表进行排序
33. h_values = sorted(h_values)
34. print("原始长度: ", int(len(h_values)))
35.
36. # 计算要去掉的元素个数
37. remove_count = int(len(h_values) * 0.1)
38. h_values_1 = []
39. # 去掉前 10%和后 10%的元素
40. for i in range(remove_count, len(h_values) - remove_count):
41.     h_values_1.append(h_values[i])
42. print("去掉前 10%和后 10%的元素之后的长度", int(len(h_values_1)))
43.
44. # 计算列表的平均值
45. h_average = calculate_average(h_values_1)
46. print("优化之后的目标区域的平均 H 分量为: ", h_average)

```

示例效果图如下:



```

main x
D:\pycharm\Python\python\python.exe D:\AllCode\python\ph\code\main.py
-ljf-632109160602
220 120
原始长度: 4476
去掉前10%和后10%的元素之后的长度 3582
优化之后的目标区域的平均H分量为: 63.83996670673185
  
```

图 3-14 计算 H 平均值

3.5.2 计算目标区域的频率分布直方图

(1) 平均值的误差

在 3.5.1 第二小节计算出平均 H 分量之后,我本打算直接将其于我们制作的 pH 值对应的 H 分量表格进行比较,然后估读出其 pH 值,但是我自己在测试过程中,发现取平均值并不准确,尤其是当该试纸条有一部分被成功染色,而另一部分未能正确染色,但是纸张的毛细作用和扩散现象,导致这一部分被染色了,示例图如下所示:

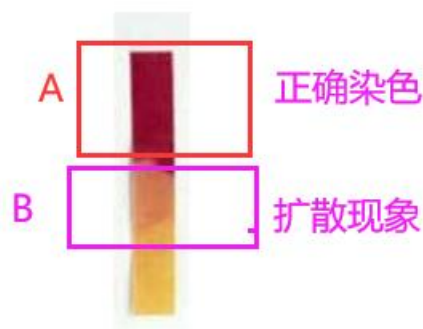


图 3-15 示例图

如上图所示，A 区域明显是正常染色形成的色彩图案，而 B 区域颜色很淡，是扩散而来的色彩区域。

如果按照我的方法去提取目标区域，去计算 H 分量的平均值，其结果如下：



图 3-16 示例图

很显然这个平均值并不能代表该试纸的检测数据，因此为了更好的评估 H 分量的数据，在通常境况下我就不采用平均值代表，而是计算 H 分量的频率分布直方图，选取频率分布直方图中出现次数最多的 H 分量数据，若有多个 H 分量数据一样，就去它们的平均值作为输出。

(2) 计算频率分布直方图

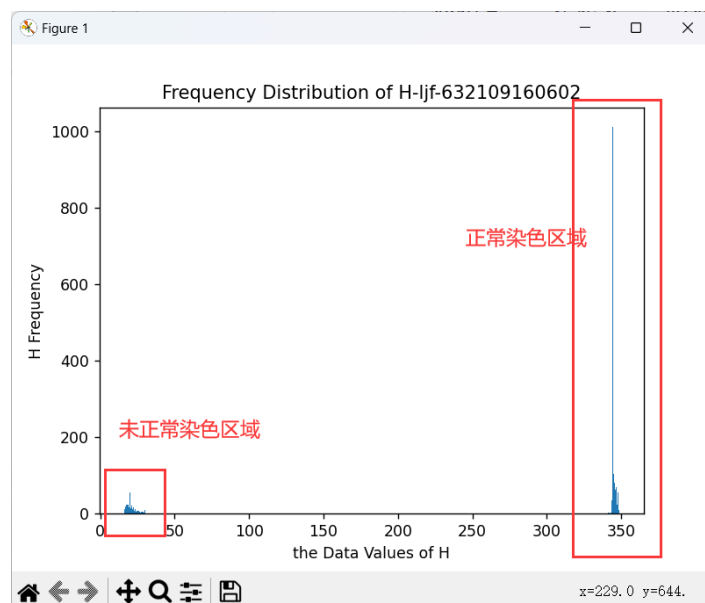
做出频率分布直方图

➤ 算法描述：

这段代码通过遍历这个列表来计算每个数据的出现次数，并将结果存储在 frequency_distribution 字典中。接着，将这个字典转换为绘图所需的格式，即两个列表：items（数据值）和 frequencies（对应的频率）。最后，使用 matplotlib 的 bar 函数绘制直方图，并设置了图表的标题和轴标签。

```
1. # 画出频率分布直方图
2. # 计算每个数据出现的次数
3. frequency_distribution = {}
4. for item in h_values_1:
5.     frequency_distribution[item] = frequency_distribution.get(item, 0) + 1
6.
7. # 将频率分布转换为可以绘图的格式
8. items = list(frequency_distribution.keys())
9. frequencies = list(frequency_distribution.values())
10.
11. # 找出出现次数最多的数据
12. max_frequency = max(frequency_distribution.values())
13. most_frequent_items = [item for item, frequency in frequency_distribution.items()
14.     if frequency == max_frequency]
15.
16. # 如果有多个数据出现次数相同，计算它们的平均值
17. if len(most_frequent_items) > 1:
18.     result = sum(most_frequent_items) / len(most_frequent_items)
19. else:
20.     result = most_frequent_items[0]
21. print("优化之后的目标区域出现频率最高的 H 值是:", result)
22.
23. # 绘制直方图
24. plt.bar(items, frequencies)
25. # 设置标题和轴标签
26. plt.title('Frequency Distribution of H'+logo)
27. plt.xlabel('the Data Values of H')
28. plt.ylabel('H Frequency')
29.
30. # 显示直方图
31. plt.show()
```

示例效果图如下：



```
D:\pycharm\Python\python\python.exe D:\AllCode\python\ph\code\main.py
-ljf-632109160602
220 120
原始长度: 8180
去掉前10%和后10%的元素之后的长度 6544
优化之后的目标区域的平均H分量为: 244.07570106814455
优化之后的目标区域出现频率最高的H值是: 344.42748091603056
```

图 3-17 作出频率分布直方图

显然根据频率分布直方图，我们看到**两极分化很突出**，如此我们便可以通过筛选**出现频率较高的 H 分量**作为输出即可。

我们可以直接将得到的 H 分量与我们制定的 H 分量标准表进行对比，找到**数据差距最小的就是我们估测的 pH 值**。

3.5.3 计算 pH 值并作图

(1) 计算 pH 值

经过上述描述，我们需要判断**数据差距最小的 H 值**，直接比较即可

计算 pH

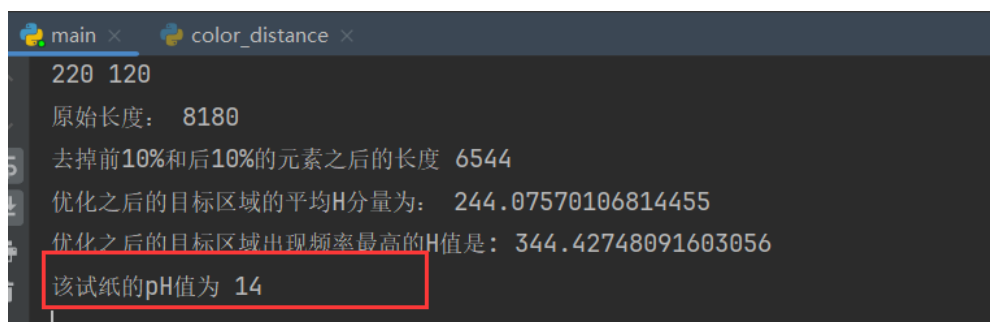
➤ 算法描述:

函数接收两个参数：`result`（频率分布直方图出现次数最多的 H 分量）和 `H_values`（标准 pH 比色卡对应的 H 分量的列表）。`min_distance` 初始化为正无穷，`nearest_index` 被初始化为 -1。

然后遍历列表，获得当前元素的索引 `i` 和值 `value`。对于 `H_values` 中的每个元素，计算它与 `result` 之间的距离即绝对值。如果当前元素与 `result` 之间的距离小于当前已知的 `min_distance`，则更新 `min_distance` 为这个新的更小的距离，并将当前元素的索引 `i` 赋值给 `nearest_index`。循环结束后，返回 `nearest_index+1`。最后的 pH 值需要加一。

```
1. # 现在有一个值 result, 找到它与 H_values 中的元素距离最小的一个, 返回该元素的索引
2. def find_nearest_index(result, H_values):
3.     min_distance = math.inf # 初始化最小距离为正无穷大
4.     nearest_index = -1 # 初始化最近元素的索引为-1
5.
6.     # 遍历 H_values 列表
7.     for i, value in enumerate(H_values):
8.         distance = abs(value - result) # 计算当前元素与 result 的距离
9.
10.        if distance < min_distance:
11.            min_distance = distance
12.            nearest_index = i
13.
14.    return nearest_index+1
15.
16. # 9. 计算 pH 值
17. pH_measure = find_nearest_index(result, H_values)
18. print("该试纸的 pH 值为", pH_measure)
19. pH_measure = str(pH_measure)
```

示例图如下：



```
main x color_distance x
220 120
原始长度: 8180
去掉前10%和后10%的元素之后的长度 6544
优化之后的目标区域的平均H分量为: 244.07570106814455
优化之后的目标区域出现频率最高的H值是: 344.42748091603056
该试纸的pH值为 14
```

图 3-18 pH 估测值

(2) 作图

在原始图像上标注 pH 值

➤ 算法描述：

这里将文字绘制在图片的左上角即可。

```

1. # 在图像上作出文字
2. def draw_image(image, text):
3.     image_pH = image.copy()
4.     # 设置文本和字体
5.     text = 'pH:'+text # 替换为你要添加的文本
6.     font = cv2.FONT_HERSHEY_SIMPLEX
7.
8.     # 指定文本的左上角位置
9.     position = (10, 30) # 这里(10, 30)是左上角的坐标，你可以根据需要调整
10.
11.    # 设置文本的尺寸和颜色
12.    font_scale = 1
13.    color = (0, 0, 255) # 红色
14.    thickness = 2
15.
16.    # 添加文本到图像
17.    cv2.putText(image_pH, text, position, font, font_scale, color, thickness)
18.
19.    return image_pH
20.
21. # 作图
22. image_pH = draw_image(image, pH_measure) # 替换为你的图像路径
23. cv2.imshow("image_pH", image_pH)

```

实现效果图：



图 3-19 作图

更多测试用例如下图：

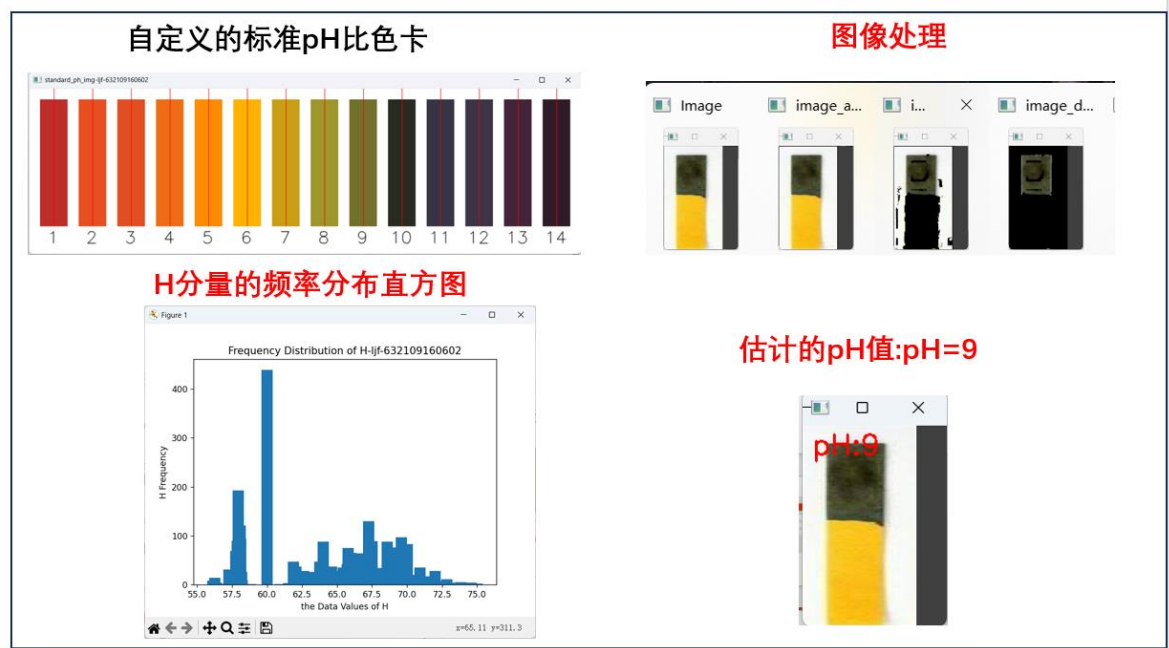


图 3-20 测试图 1

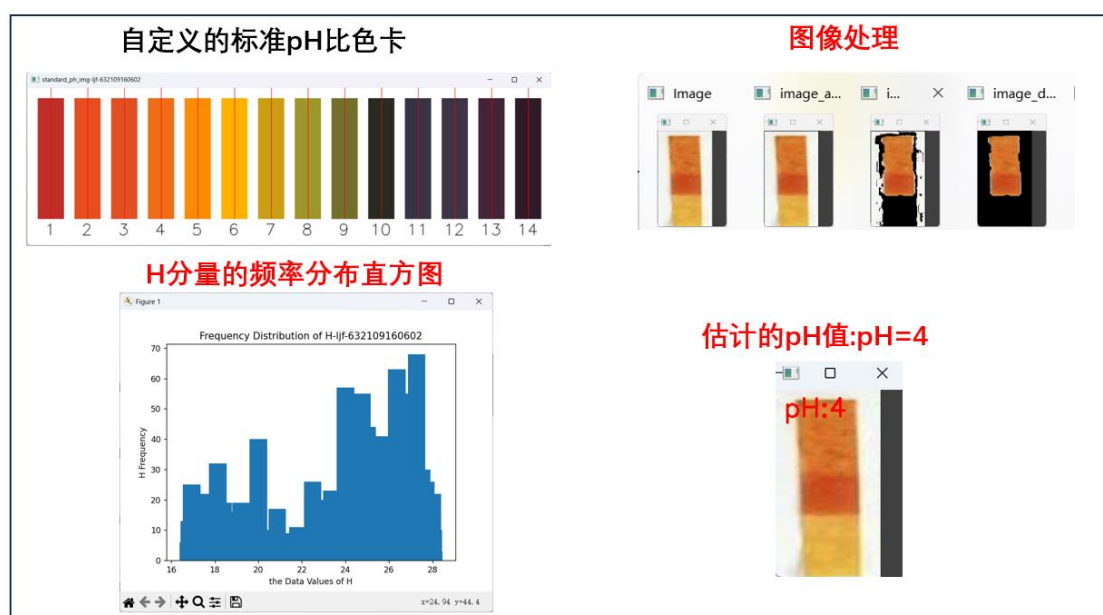


图 3-21 测试图 2

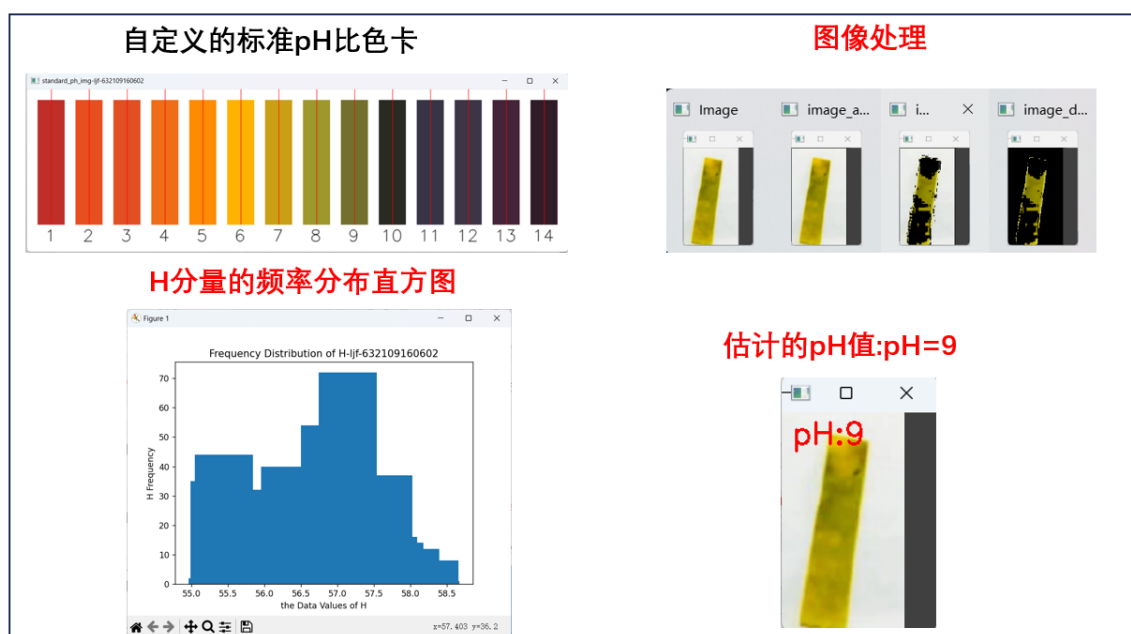


图 3-22 测试图 3

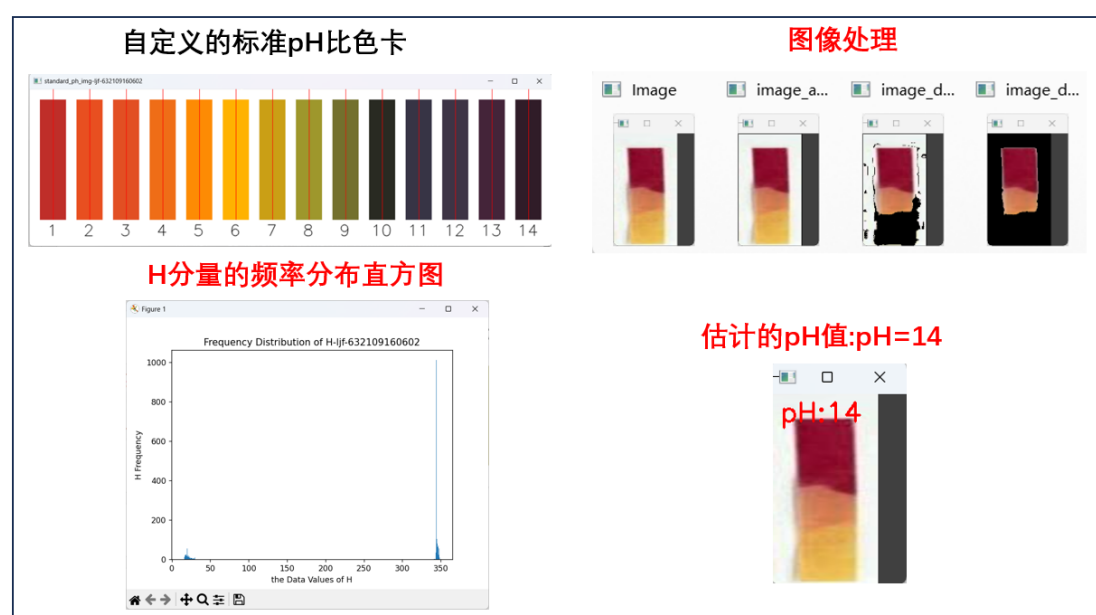


图 3-23 测试图 4

至此，成功测量出了 pH 值！完成了老师的任务！

第四章 总结

在本次实验中，我成功地实现了一种基于计算机视觉的 pH 试纸数值判读方法。

通过第二章和第三章的实验，我验证了该方法的可行性和准确性。

在第二章中，我们从**纯视觉角度**出发，利用计算机视觉技术精确地读出 pH 值，并通过**欧式距离法**和**插值法**计算最接近的 pH 值，实现了从整数值到浮点数值的精度提升。

在第三章中，我通过数字图像处理技术对**真实的 pH 试纸图片**进行处理，包括**滤波**、**颜色空间转换**和**目标区域提取**，最终实现了对未知溶液 pH 值的估测。

在解题过程中，我首先在第二章中提出了问题，并提出了基于计算机视觉的初步解决方案。我们通过绘制标准比色卡、计算欧式距离和插值法等步骤，实现了对 pH 值的精确测量。在第三章中，进一步探索了在**非理想条件**下，如何通过**图像预处理**和**目标区域提取**来估测 pH 值。我们采用了**中值滤波**和**RGB 到 HSV**的转换技术，有效地提高了颜色数据的准确性，并最终通过计算目标区域的**平均 H 分量**和**频率分布直方图**来确定 pH 值。

通过本次实验，我不仅提高了 pH 测量的精度，也为图像处理技术在其他领域的应用提供了新的思路。然而，我也意识到，尽管本方法在实验条件下表现良好，但在实际应用中可能还会受到环境光、图像清晰度等因素的影响。因此，未来的工作需要进一步优化算法，以提高其鲁棒性和实用性。此外，我还将探索如何将本方法与其他传感器技术相结合，以实现更全面的化学分析。