**OpenCV颜色分割**



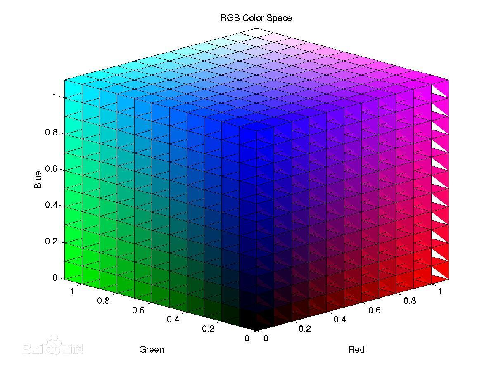
**颜色空间**

颜色空间也称彩色模型（又称彩色空间或彩色系统）它的用途是在某些标准下用通常可接受的方式对彩色加以说明。计算机中常用的颜色空间为RGB颜色空间和HSV颜色空间

|  |
| --- |
| https://baike.baidu.com/item/%E9%A2%9C%E8%89%B2%E7%A9%BA%E9%97%B4 |

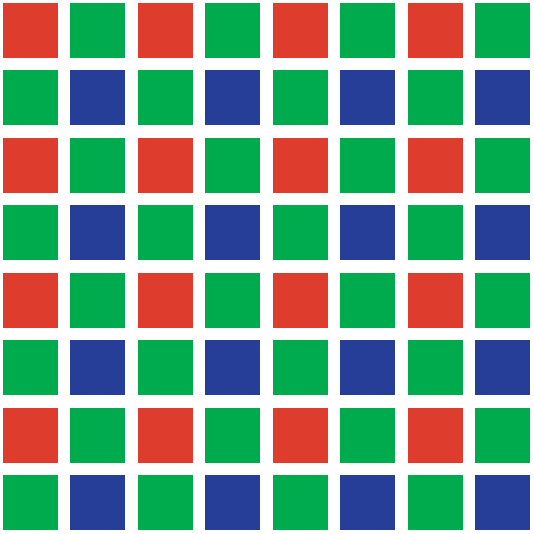
这是百度百科中的一个描述，比较抽象。用通俗的语言来说，颜色空间就是给每一个颜色设定了一个坐标，比如白色在RGB颜色空间中可以描述为(100%, 100%, 100%)，黑色可以描述为(0, 0, 0)，正红色为(100%, 0, 0)。

**RGB颜色空间**



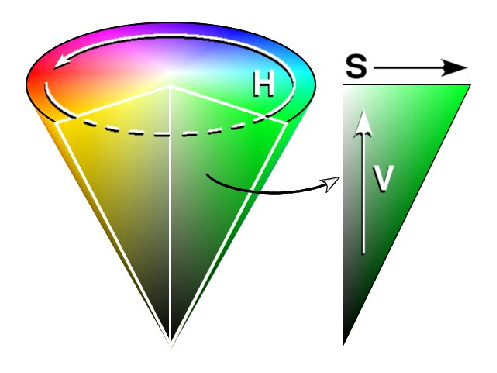
RGB颜色空间是面向硬件的颜色空间。

实际上图像传感器上的每个单元只能分辨亮度的强弱，并不能分辨色相（红色蓝色还是绿色）。为了让相机可以拍出彩色图片，就需要分别测量三原色的光照强度。1974年Bryce Bayer提出著名的Bayer阵列，其方法是将传感器单元四个为一组，其中两个采用绿色滤光片，另外两个分别采用红色和蓝色滤光片，如下图。



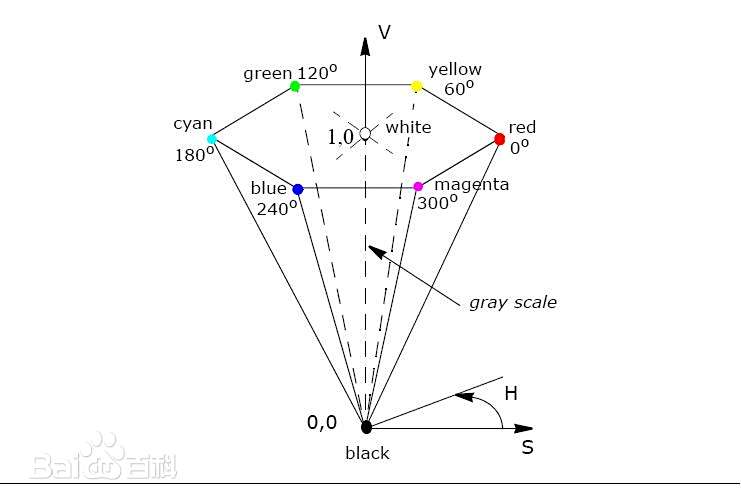
这样就可以测量出图像上不同颜色的强度分布情况，通过插值算法（通过周围的像素估计没有直接测量的像素测量值），将其他区域的颜色补上，就可以得到每个像素单元的RGB光强度值了。绿色单元数量是2倍是因为人类的视力对绿色更为敏感，这样绿色的采样率更高，精度更好。在工程中也有采用其他阵列的情况，例如华为曾经搭载过采用RYYB阵列的传感器，原理上其实区别不大。

**HSV颜色空间**



HSV颜色空间是面向用户的颜色空间。

HSV(Hue色调, Saturation饱和度, Value值-即亮度)更符合人类对于色彩的理解，色相、饱和度、亮度。色相决定了颜色本身，饱和度决定了颜色的鲜艳程度，亮度决定了颜色的明亮程度。靖桥喜欢的莫兰迪色系就是属于低饱和度的颜色。



**认识图像**

**图像分类**

图像一般分为位图和矢量图。照相机拍出来的图片一般都是位图，其基本单位为像素。通过n个像素构建出整张图片。矢量图用几何关系描述图像，一般用于设计等领域，常见的矢量图如计算机里的文字，它通过矢量图存储，再转换为位图显示。矢量图典型的特征是其不受图像大小的影响，缩放图像不会导致失真。位图中包含的信息是一个定值，因此放大图像不会让图像显示更多的细节。视觉组研究的问题以位图为主

**像素**

像素是图像的基本单元，每个像素中的信息用于描述这一像素的特性，例如RGB值，HSV值。存储的信息取决于图像的格式，例如深度相机使用的RGBD图像中，就包含红色强度、绿色强度、蓝色强度、距离。

**通道数**

每个像素需要存储的信息类型决定了图像的通道数（图像是由不同的图层一层一层叠起来的），例如HSV图像为三通道，黑白图像为单通道，RGBD图像为四通道。原理上可以多个通道一起处理，但是一般我们会将多通道图像拆成多个单通道图像处理。

**OpenCV的使用**

本文采用python对颜色分割案例进行演示。请注意OpenCV因为历史原因，像素数据的存储顺序是BGR而不是RGB。

在python中使用OpenCV需要在程序的开始使用import指令引入OpenCV库。

|  |
| --- |
| Python import cv2 |

读取图像使用imread函数，这个函数的第一个参数为图像的存储位置，在使用相对路径的情况下，其根目录是程序运行目录，其详细解释见链接

|  |
| --- |
| Python img\_bgr = cv2.imread("test.png") |

|  |
| --- |
| https://docs.opencv.org/4.10.0/d4/da8/group\_\_imgcodecs.html#gab32ee19e22660912565f8140d0f675a8 |

OpenCV提供了颜色空间转换函数cvtColor，转换成HSV，⽤于对象跟踪

|  |
| --- |
| Python img\_hsv = cv2.cvtColor(img\_bgr, cv2.COLOR\_BGR2HSV) |

|  |
| --- |
| https://docs.opencv.org/4.10.0/d8/d01/group\_\_imgproc\_\_color\_\_conversions.html#ga397ae87e1288a81d2363b61574eb8cab |

此时图像是三通道的。这里使用split函数完成通道分离这一步操作

|  |
| --- |
| Python img\_h, img\_s, img\_v = cv2.split(img\_hsv)  # 打印HSV通道的统计信息，帮助确定合适的阈值  print(f"H通道范围: {img\_h.min()} - {img\_h.max()}")  print(f"S通道范围: {img\_s.min()} - {img\_s.max()}")  print(f"V通道范围: {img\_v.min()} - {img\_v.max()}")  # 显示各通道图像以便分析  cv2.imshow("H Channel", img\_h)  cv2.imshow("S Channel", img\_s)  cv2.imshow("V Channel", img\_v) |

|  |
| --- |
| https://docs.opencv.org/4.10.0/d2/de8/group\_\_core\_\_array.html#ga8027f9deee1e42716be8039e5863fbd9 |

现在img\_h，img\_s，img\_v就是三个单通道图像。我们可以针对每个通道构建掩膜（白色区域表示完全通过，黑色区域表示完全遮挡），从而找到我们关注的部分，构建掩膜的方式是选择一定的数值范围。使用inRange函数.

掩膜(Mask)原理

什么是掩膜？

掩膜就像一个模板或筛子，它是一张二值图像（只有黑色和白色），用来决定原图像中哪些部分应该被保留，哪些部分应该被丢弃。

​白色区域(255)​​：完全保留原图像内容 → "通过"

​黑色区域(0)​​：完全丢弃原图像内容 → "阻挡"

|  |
| --- |
| Python # mask\_h = cv2.inRange(img\_h, 20, 30) #原来的这句代码不对 |
| # 针对粉红色调整HSV范围  # 偏紫的粉红(H:150-180)  # 粉红色范围（偏紫）  mask\_h = cv2.inRange(img\_h, 150, 180)  mask\_s = cv2.inRange(img\_s, 50, 255)  mask\_v = cv2.inRange(img\_v, 50, 255) |

|  |
| --- |
| https://docs.opencv.org/4.10.0/d2/de8/group\_\_core\_\_array.html#ga48af0ab51e36436c5d04340e036ce981 |

对于每个通道，我们都可以得到我们关注的部分。这里要注意，通过单个通道就得到我们想分离出的物体，这种情况并不多见，更多的情况是通过多个通道各得到一部分，通过逻辑运算（与或非）得到我们的目标（本质上是集合运算）。所以每个通道只需要等到我们想要的效果就可以了，例如色相通道，背景中与目标同样色相的部分可以通过其他通道滤除，不必纠结于一点点的差异，过于严苛的条件会让程序的适应能力变差，太过相近的颜色不在本方法的讨论范围内。

通过bitwise\_and/bitwise\_or/bitwise\_not可以对图像进行按像素的逻辑运算，这里我们通过bitwise\_and对特征进行融合

|  |
| --- |
| Python mask\_h\_and\_s = cv2.bitwise\_and(mask\_h, mask\_s) mask = cv2.bitwise\_and(mask\_h\_and\_s, mask\_v) |

|  |
| --- |
| https://docs.opencv.org/4.10.0/d2/de8/group\_\_core\_\_array.html#ga60b4d04b251ba5eb1392c34425497e14 |

最后通过掩膜将图像输出出来，imshow用于显示图像，imwrite用于保存图像，waitKey用于等待按键输入防止窗口直接关闭，设置为0表示无限等待直到有按键输入，其返回值为按键值，这里未使用。

|  |
| --- |
| Python img\_out = cv2.bitwise\_and(img\_bgr, img\_bgr, mask = mask) #这相当于：当mask为白色时：输出原图像像素；当mask为黑色时：输出黑色  cv2.imshow("img", img\_bgr)  cv2.imshow("imghsv", img\_hsv)  cv2.imshow("H Mask (Purple-Pink)", mask\_h)  cv2.imshow("Combined Mask", mask) cv2.imshow("img", img\_out) cv2.imwrite("img\_out.png", img\_out) cv2.waitKey(0) |

这个过程就像：

1.先用H通道筛一遍：只保留H值在150-180的区域

2.再用S通道筛：只保留饱和度50-255的区域

3.最后用V通道筛：只保留明度50-255的区域

4.最终只留下同时满足三个条件的区域

|  |
| --- |
| https://docs.opencv.org/4.10.0/d7/dfc/group\_\_highgui.html#ga453d42fe4cb60e5723281a89973ee563  https://docs.opencv.org/4.10.0/d4/da8/group\_\_imgcodecs.html#ga8ac397bd09e48851665edbe12aa28f25  https://docs.opencv.org/4.10.0/d7/dfc/group\_\_highgui.html#ga5628525ad33f52eab17feebcfba38bd7 |

|  |
| --- |
| Tips：调试时可以通过某些截图软件方便的查看其各通道值，例如飞书截图，Ctrl+Shift+a |

效果图：



完整代码：

import cv2

# import numpy as np

img\_bgr = cv2.imread("test.png")

img\_hsv = cv2.cvtColor(img\_bgr, cv2.COLOR\_BGR2HSV)

img\_h, img\_s, img\_v = cv2.split(img\_hsv)

# 显示各通道图像以便分析

cv2.imshow("H Channel", img\_h)

cv2.imshow("S Channel", img\_s)

cv2.imshow("V Channel", img\_v)

# 针对粉红色调整HSV范围

# 粉红色通常有偏紫的粉红(H:150-180)

mask\_h = cv2.inRange(img\_h, 150, 180)

mask\_s = cv2.inRange(img\_s, 50, 255)

mask\_v = cv2.inRange(img\_v, 50, 255)

mask\_h\_and\_s = cv2.bitwise\_and(mask\_h, mask\_s)

mask = cv2.bitwise\_and(mask\_h\_and\_s, mask\_v)

img\_out = cv2.bitwise\_and(img\_bgr, img\_bgr, mask = mask)

cv2.imshow("img", img\_bgr)

cv2.imshow("imghsv", img\_hsv)

# cv2.imshow("H Mask 1 (Red-Pink)", mask\_h1)

cv2.imshow("H Mask (Purple-Pink)", mask\_h)

cv2.imshow("Combined Mask", mask)

cv2.imshow("img\_out", img\_out)

cv2.imwrite("img\_out.png", img\_out)

cv2.waitKey(0)

cv2.destroyAllWindows()