上海交通大学
SHANGHAI JIAO TONG UNIVERSITY

SJTU

树莓派应用入门：视觉编程

董德礼

arli@sjtu.edu.cn

2021年3月9日

上海交通大學
SHANGHAI JIAO TONG UNIVERSITY

目 录

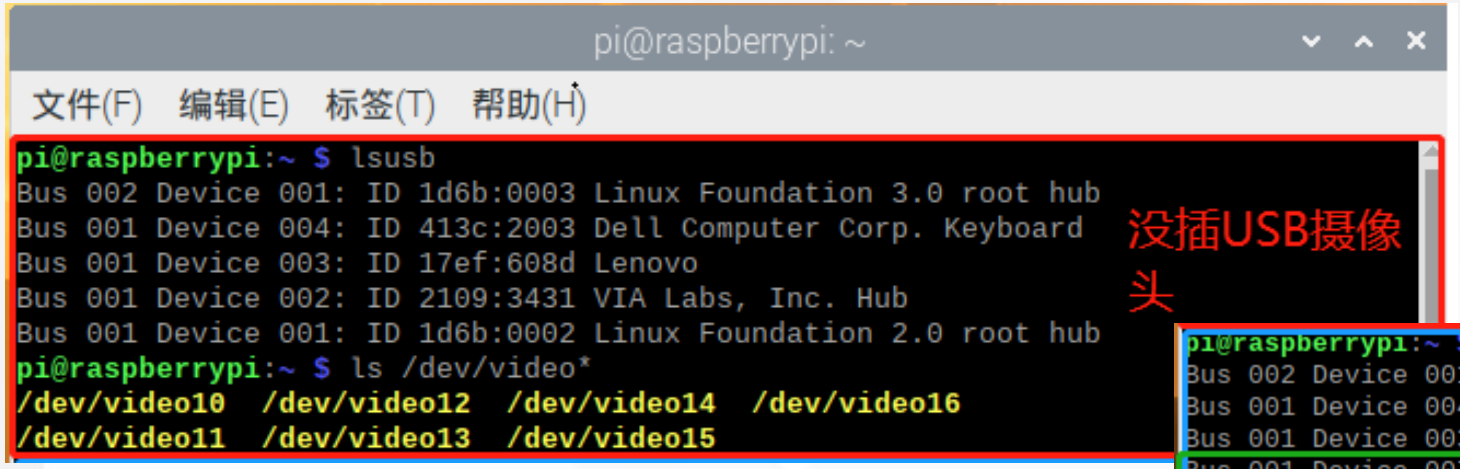
01

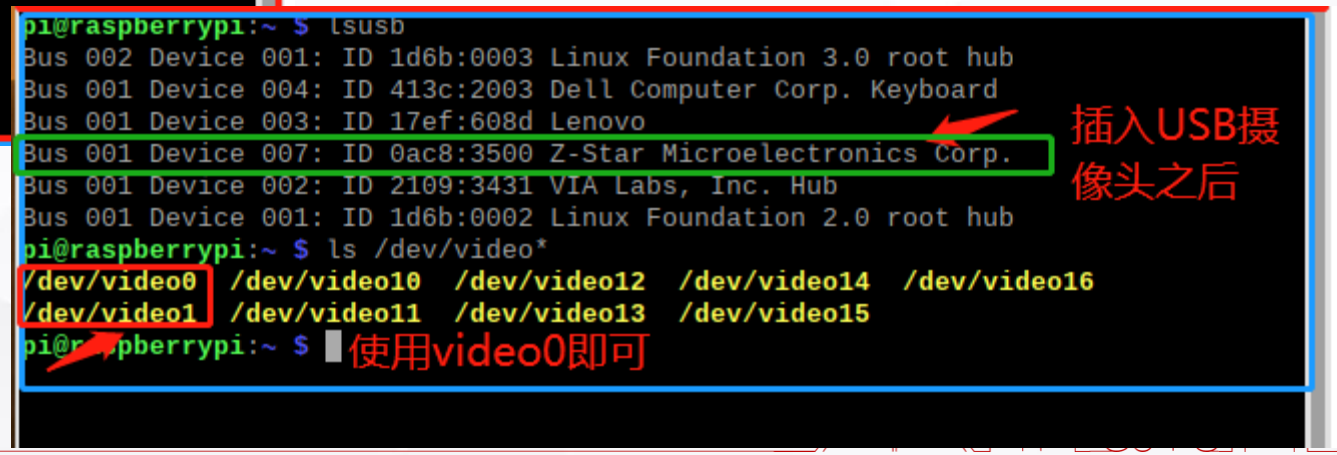
USB摄像头的接入配置

免驱USB摄像头

摄像头硬件连接

- 首先，最好使用linux（Raspberry）能免驱识别的USB接口摄像头。
- 将摄像头的USB线插入树莓派的USB接口（2.0或3.0）都可以。
- 在终端使用命令*lsusb*查看USB设备，并使用ls /dev/video*查看视频设备文件。
- 最好能在插入设备前后，分别执行以上两条命令，对比查看，即可找到设备和文件：





测试摄像头

- 使用fswebcam命令测试抓图：

```
$ sudo apt-get install fswebcam
$ fswebcam /dev/video0 ~/image.jpg
```

  - 这是fswebcam的最简单用法，第一个参数是摄像头对应的设备文件路径，第二个参数是保存截图的路径。

- 如果不满足于抓图，还可以查看摄像头的视频效果，使用工具软件luvcview即可：

```
$ sudo apt-get install luvcview
$ luvcview -s 1080x720
```

- 还可以安装mplayer管理查看摄像头，mplayers是LINUX系统下一款功能强大也很好用的视频播放软件：

```
sudo apt-get install mplayer
sudo mplayer tv://
```

- 如果由多个摄像头，可以指定要查看的设备：`sudo mplayer tv:// -tv device=/dev/video2`

02

Opencv-python库

Opencv-python库简介

opencv-python库的安装配置

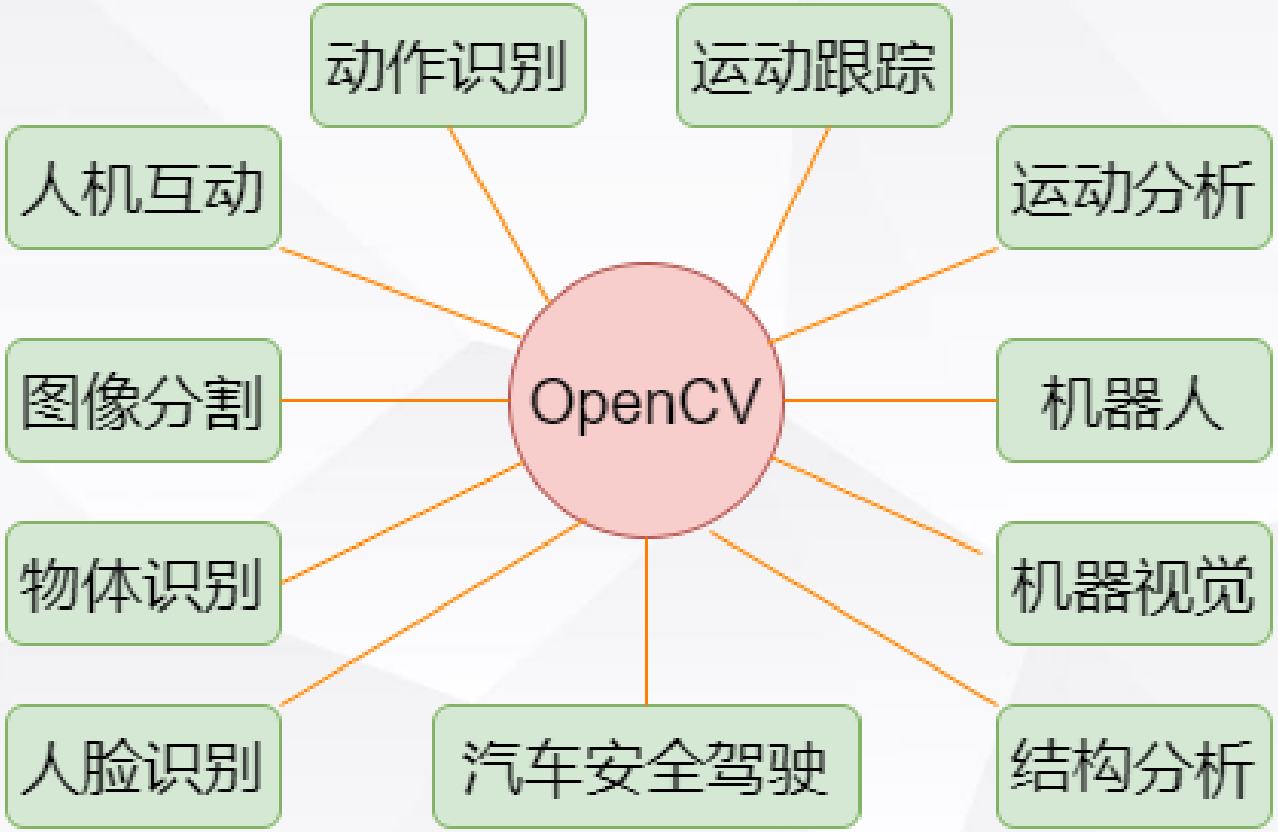
Opencv-python简介

- **Open Source Computer Vision**
- OpenCV， Gary Bradsky1999年开始于Intel，2000年发布第一个版本。
- 2005年之后，在Willow Garage的支持下，由Gary Bradsky和Vadim Pisarevsky共同领导该项目。
- OpenCV是一个基于BSD许可（开源）发行的跨平台计算机视觉和机器学习软件库。
- 可以运行在Linux、Windows、Android和Mac OS操作系统上。
- 轻量级而且高效，由C/C++实现，提供了**Python**、Java、C#、Ruby、MATLAB、等语言的接口，实现了图像处理和计算机视觉方面的很多通用算法。

https://docs.opencv.org/master/d0/de3/tutorial_py_intro.html

OpenCV应用领域

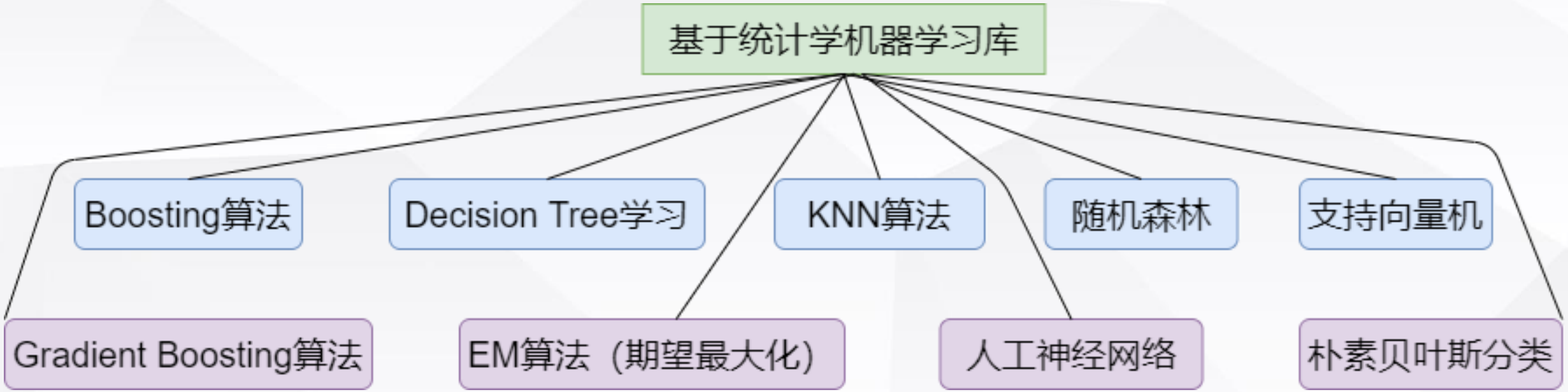
- 在计算机视觉项目的开发中，OpenCV作为较大众的开源库，拥有丰富的常用图像处理函数库，采用C/C++语言编写，可以运行在Linux/Windows/Mac等操作系统上，能够快速的实现一些图像处理和识别的任务。



OpenCV模块结构

1. **核心功能模块**：定义了基本数据结构（包括密集的多维Mat数组）和被其它模块使用的基本功能
2. **图像处理模块**：包括线性和非线性图像滤波、几何图形转换（重置大小，放射和透视变形，通用基本表格重置映射等）、色彩空间转换、直方图等。
3. **影像分析模块**：包括动作判断、背景弱化和目标跟踪算法等。
4. **3D校准模块**：基于多视图的几何算法，平面和立体摄像机校准，对象姿势判断，立体匹配算法，和3D元素的重建等。
5. **平面特征模块**：特征判断，特征描述和对特征描述的对比。
6. **对象侦查模块**：目标和预定义类别实例化的侦查（例如，脸、眼睛、被子、人、汽车等）。
7. **视频输入输出模块**：一个容易使用的视频采集和视频解码器。
8. **Highgui模块**：一个容易使用的用户功能界面。
9. **GPU模块**：来自不同OpenCV模块的GPU加速算法。

OpenCV模块结构——基于统计学机器学习库



Opencv-python库的安装

- 在树莓派上安装opencv-python库：
- sudo pip3 install opencv-python
- 安装完成后，可在命令终端中执行***python3***，***import cv2*** 命令
- 如果没有错误提示就表示安装成功，可使用***cv2.__version__***查看版本号
- 如果有提示库文件缺失，就使用

```
sudo apt-get update
sudo apt-get install
```

命令，缺什么库，就安装什么库

- 例如，错误提示：

```
ImportError: libcblas.so.3: cannot open shared object file: No such file or directory
ImportError: libatlas.so.3: cannot open shared object file: No such file or directory
ImportError: libjasper.so.1: cannot open shared object file: No such file or directory
ImportError: libQtTest.so.4: cannot open shared object file: No such file or directory
```

- 安装指令：

```
sudo apt-get install libcblas-dev
sudo apt-get install libatlas-base-dev
sudo apt-get install libjasper-dev
sudo apt-get install libqt4-test
```

03

图像视频基本操作

读入、显示、保存图像

视频的捕获、播放、保存

SJTU

读入图像

读入图像：cv2.imread()

- 第一个参数是图像路径，第二个参数是要告诉函数如何读取这幅图像：
  - Cv2.IMREAD_COLOR(1)：读入一幅彩色图像，图像的透明度被忽略,这是默认值
  - Cv2.IMREAD_GRAYSCALE(0)：以灰度模式读入图像
  - Cv2.IMREAD_UNCHANGED(-1)：读入一幅图像，并且包括图像的alpha通道

```
import numpy as np
import cv2

# Load an color image in grayscale
img = cv2.imread('messi5.jpg',0)
```

注意：就算你指定的图像路径是错的，OpenCV也不会提醒你，但当你使用print输出时会得到None。

显示图像

- **显示图像：cv2.imshow()**
  - 窗口会自动调整为图像大小
  - 第一个参数是窗口的名字，其次才是我们的图像
  - 你可以创建多个窗口，只要你喜欢，但是必须给他们不同的名字

```
cv2.imshow('image',img)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

- 一种特殊的情况是，你也可以先创建一个窗口，之后再加载图像。
- 这种情况下，你可以决定窗口是否可以调整大小。
- 使用到的函数是 **cv2.namedWindow()**。
- 初始设定函数标签**是 cv2.WINDOW_AUTOSIZE**。

```
import numpy as np
import cv2

cv2.namedWindow('image', cv2.WINDOW_NORMAL)
cv2.imshow('image',img)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

- 但是如果你把标签改成**cv2.WINDOW_NORMAL**，你就可以调整窗口大小了。
- 当图像维度太大，或者要添加轨迹条时，调整窗口大小将会很有用

保存图像

保存图像：cv2.imwrite()

- 第一个参数时文件名，第二个参数是你要保存的图像。

```
cv2.imwrite('messigray.png',img)
```

总结一下：

- 加载一个灰度图，显示图片
- 按下‘s’键保存后退出
- 或者按下ESC键退出不保存

```
import numpy as np
import cv2

img = cv2.imread('messi5.jpg',0)
cv2.imshow('image',img)
k = cv2.waitKey(0)
if k == 27:         # wait for ESC key to exit
    cv2.destroyAllWindows()
elif k == ord('s'): # wait for 's' key to save and exit
    cv2.imwrite('messigray.png',img)
    cv2.destroyAllWindows()
```

用摄像头捕获视频

VideoCapture对象

- 首先实例化一个VideoCapture对象
  - 该对象的参数就是摄像头设备的索引号
  - 就是用*ls /dev/video**得到的摄像头文件号
- 然后就可以使用read()成员函数一帧一帧获取图像
- 最后别忘记释放VideoCapture对象
- **cap.isOpened()**
- **Cap.set(propID,value):**
  - Cap.set(3,320)
  - Cap.set(4,240)
- **Cap.get(propID): propID=0~18**

```python
import numpy as np
import cv2

cap = cv2.VideoCapture(0)

while(True):
    # Capture frame-by-frame
    ret, frame = cap.read()

    # Our operations on the frame come here
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

    # Display the resulting frame
    cv2.imshow('frame',gray)
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break

# When everything done, release the capture
cap.release()
cv2.destroyAllWindows()
```

从文件中播放视频

- 与从摄像头中捕获一样，你只需要把设备索引号改成视频文件的名字。
- 在播放每一帧时，使用 cv2.waiKey() 设置适当的持续时间。
- 如果设置的太低视频就会播放的非常快，如果设置的太高就会播放的很慢（你可以使用这种方法控制视频的播放速度）。通常情况下 25 毫秒就可以了。

```
1   import numpy as np
2   import cv2
3   cap = cv2.VideoCapture("example.mp4")
4   while True:
5       ret,frame = cap.read()
6       cv2.imshow('frame',frame)
7       if cv2.waitKey(25) == ord('q'):
8           break
9   cap.release()
10  cv2.destroyAllWindows()
```

保存视频

- 在捕获视频，并对每一帧都进行加工之后我们想要保存这个视频。
- 对于图片来时很简单只需要**使用 cv2.imwrite()**，但对于视频来说就要多做点工作：
- 这次我们要创建一个 VideoWriter 的对象。我们应该确定一个输出文件的名字。接下来指定 FourCC 编码（下面会介绍）。播放频率和帧的大小也都需要确定。最后一个是 isColor 标签。如果是 True，每一帧就是彩色图，否则就是灰度图。
- FourCC 就是一个 4 字节码，用来确定视频的编码格式。可用的编码列表可以从**fourcc.org**查到，这参数是依赖平台的。
  - In Fedora: DIVX, XVID, MJPG, X264, WMV1, WMV2. (XVID is more preferable. MJPG results in high size video. X264 gives very small size video)
- FourCC 码以下面的格式传给程序，以 XVID编码为例：
  - cv2.VideoWriter_fourcc('X', 'V', 'I', 'D')

保存视频的代码示例

```python
import numpy as np
import cv2
cap = cv2.VideoCapture(0)
fourcc = cv2.VideoWriter_fourcc('X', 'V', 'I', 'D')
#fourcc = cv2.VideoWriter_fourcc('H', '2', '6', '4')
# fourcc = cv2.VideoWriter_fourcc('H', 'E', 'V', 'C')
out = cv2.VideoWriter('output.avi',fourcc,20.0,(640,480))
while cap.isOpened():
    ret,frame = cap.read()
    if ret:
        frame=cv2.flip(frame,1)
        cv2.imshow('frame',frame)
        out.write(frame)
        if cv2.waitKey(1) == ord('q'):
            break
    else:
        break
cap.release()
out.release()
cv2.destroyAllWindows()
```

- 预编译安装的opencv默认支持**XVID**编码，不支持H264和H265（HEVC）编码，需要额外安装。

04

opencv-python绘图

Line

Circle

基本绘图函数

- **Cv2.line()，cv2.circle()，cv2.rectangle()，cv2.ellipse()，cv2.putText()**
- 所有这些函数都需要以下几个参数：
  - **img**：想要绘制图形的那幅图像
  - **color**：所绘图形的颜色，以RGB为例，需要传入一个元组，例如(255,0,0)代表蓝色；对于灰度图形，只需要传入灰度值。
  - **thickness**：所绘图形线条的粗细，如果设置为-1，那么所绘图形就会被填充（实心），默认值是1。
  - **linetype**：线条的类型，例如连接、抗锯齿等类型。默认值是8-连接。Cv2.LINE_AA为抗锯齿，这样看起来会非常平滑。

基本绘图代码举例

画线、矩形、圆、椭圆

```
import numpy as np
import cv2

# Create a black image
img=np.zeros((512,512,3), np.uint8)

# Draw a diagonal blue line with thickness of 5 px
cv2.line(img,(0,0),(511,511),(255,0,0),5)

cv2.rectangle(img,(384,0),(510,128),(0,255,0),3)

cv2.circle(img,(447,63), 63, (0,0,255), -1)

cv2.ellipse(img,(256,256),(100,50),0,0,180,255,-1)
```

在图片上添加文字:

要在图片上绘制文字，你需要设置下列参数:

你要绘制的文字、你要绘制的位置

字体类型、字体的大小

文字的一般属性如颜色，粗细，线条的类型等。

为了更好看一点推荐使用linetype=cv2.LINE_AA。

```
font=cv2.FONT_HERSHEY_SIMPLEX
cv2.putText(img,'OpenCV',(10,500), font, 4,(255,255,255),2)
```

```
winname = 'example'
cv2.namedWindow(winname)
cv2.imshow(winname, img)
cv2.waitKey(0)
cv2.destroyWindow(winname)
```

05

颜色识别

RGB颜色空间

HSV颜色空间

HSL颜色空间

颜色识别及追踪实验

SJTU

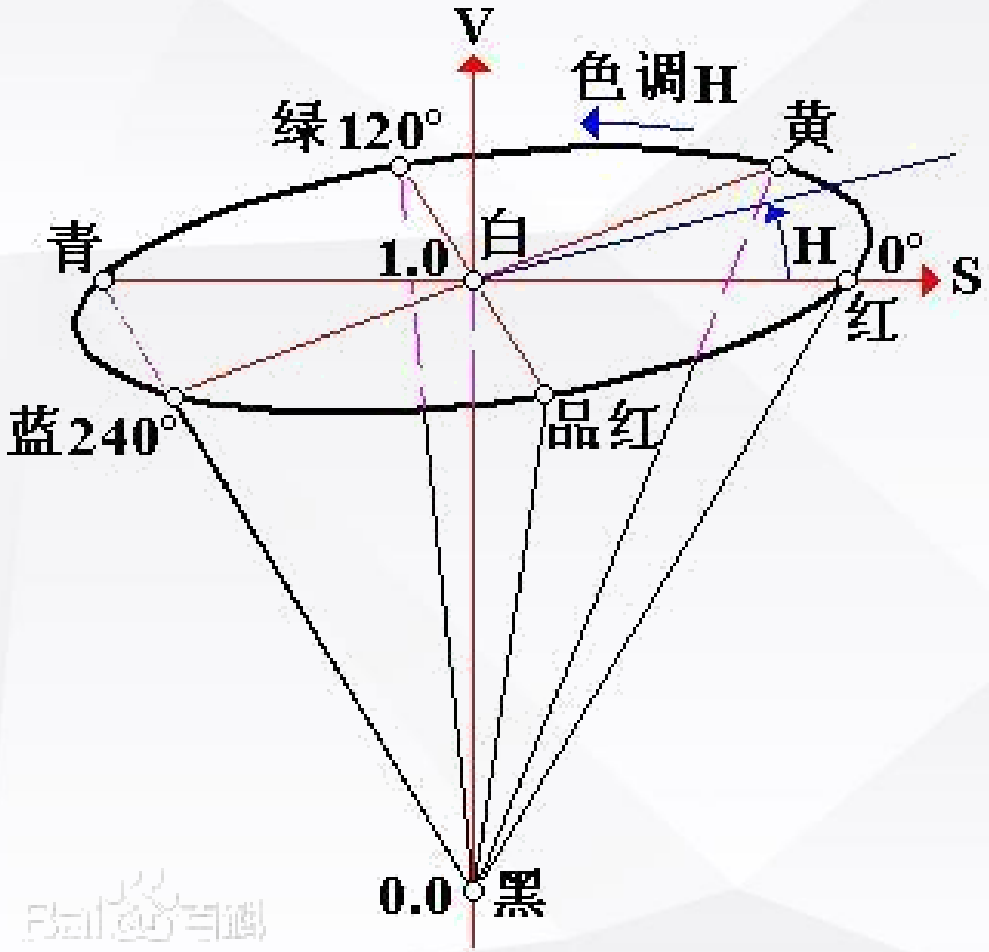
RGB颜色空间

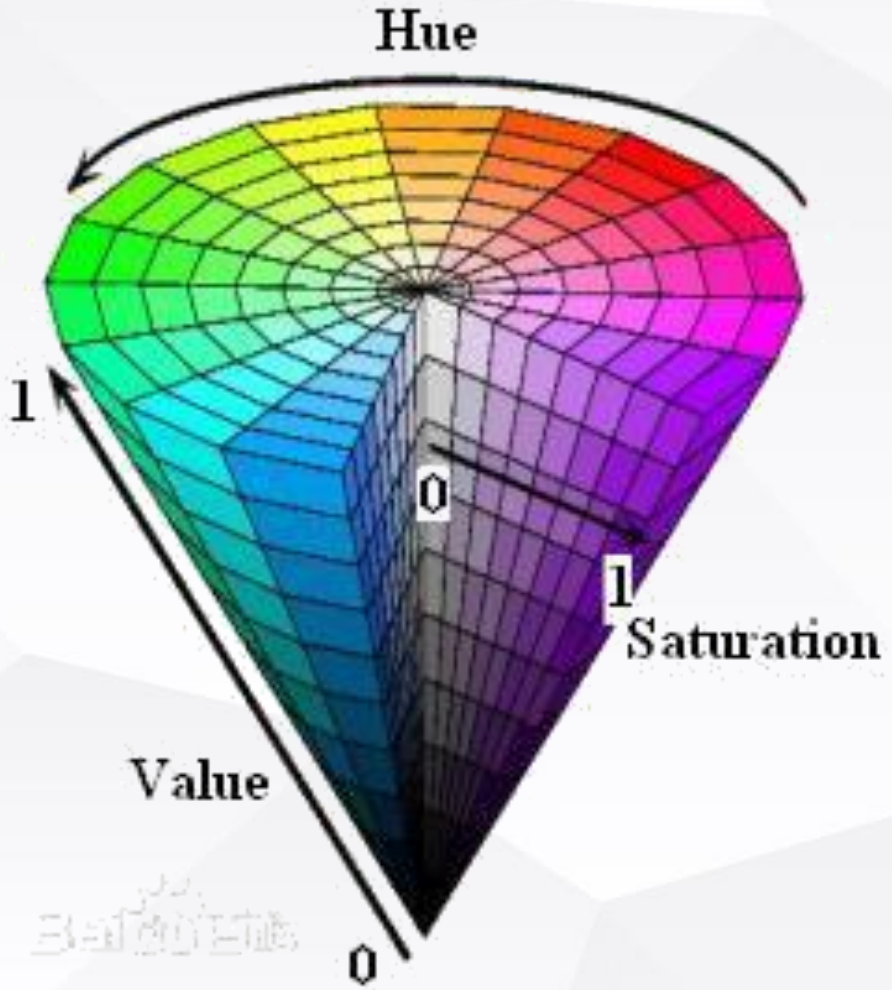
- **RGB颜色空间**（在OpenCV里习惯用**BGR**）
- 用三个通道表示一幅图像，这三种颜色不不同线性组合可以形成几乎所有的其它颜色。
- RGB颜色空间是图像处理中最基本、最常用、面向硬件的颜色空间，比较容易理解。
- 但它有局限性，**适合于显示图像，却不适合于图像处理**：
  - 在RGB颜色空间中，任何颜色都与这三个分量有关，而且这三个分量之间也是高度相关，所以连续变换颜色时并不直观，想对图像的颜色进行调整需要更改三个分量才行。
  - 在RGB颜色空间中，获取到的图像容易受光照、遮挡和阴影等情况的影响，即对亮度比较敏感。而RGB颜色空间的三个分量都与亮度密切相关，即只要亮度改变,三个分量都会随之改变。
  - 最后一点，人眼对这三种分量的敏感程度时不一样的，在单色中，人眼对红色最不敏感，对蓝色最敏感，这导致RGB颜色空间是一种均匀性很差的颜色空间。对于某一种颜色，人们很难推测出较为j精确的RGB数值来表示。

HSV颜色空间

- HSV(Hue, Saturation, Value)颜色空间
- HSV是根据颜色的直观特性有A.R.Smith于1978年创建的一种颜色空间。
- 在图像处理中使用较多的是HSV颜色空间，它比RGB更接近人们对色彩的感知经验，非常直观的表达颜色的色调、鲜艳程度和明暗程度，方便进行颜色对比。
- 在 HSV 颜色空间下，比 BGR 更容易跟踪某种颜色的物体，常用于分割指定颜色的物体。
- HSV 表达彩色图像的方式由三个部分组成：
  - Hue（色调、色相）：**0~360°，**从红色开始按逆时针方向计算，红色为0°，绿色为120°,蓝色为240°。它们的补色是：黄色为60°，青色为180°,紫色为300°。
  - Saturation（饱和度、色彩纯净度）：**0~100%，表示颜色接近光谱色（白色）的程度**，值越小，光谱色（白色）成分越大，越接近白色；值越大，白色成分越小，颜色越饱和。
  - Value（明度）：**0~100%，表示颜色明亮的程度。**

HSV颜色模型



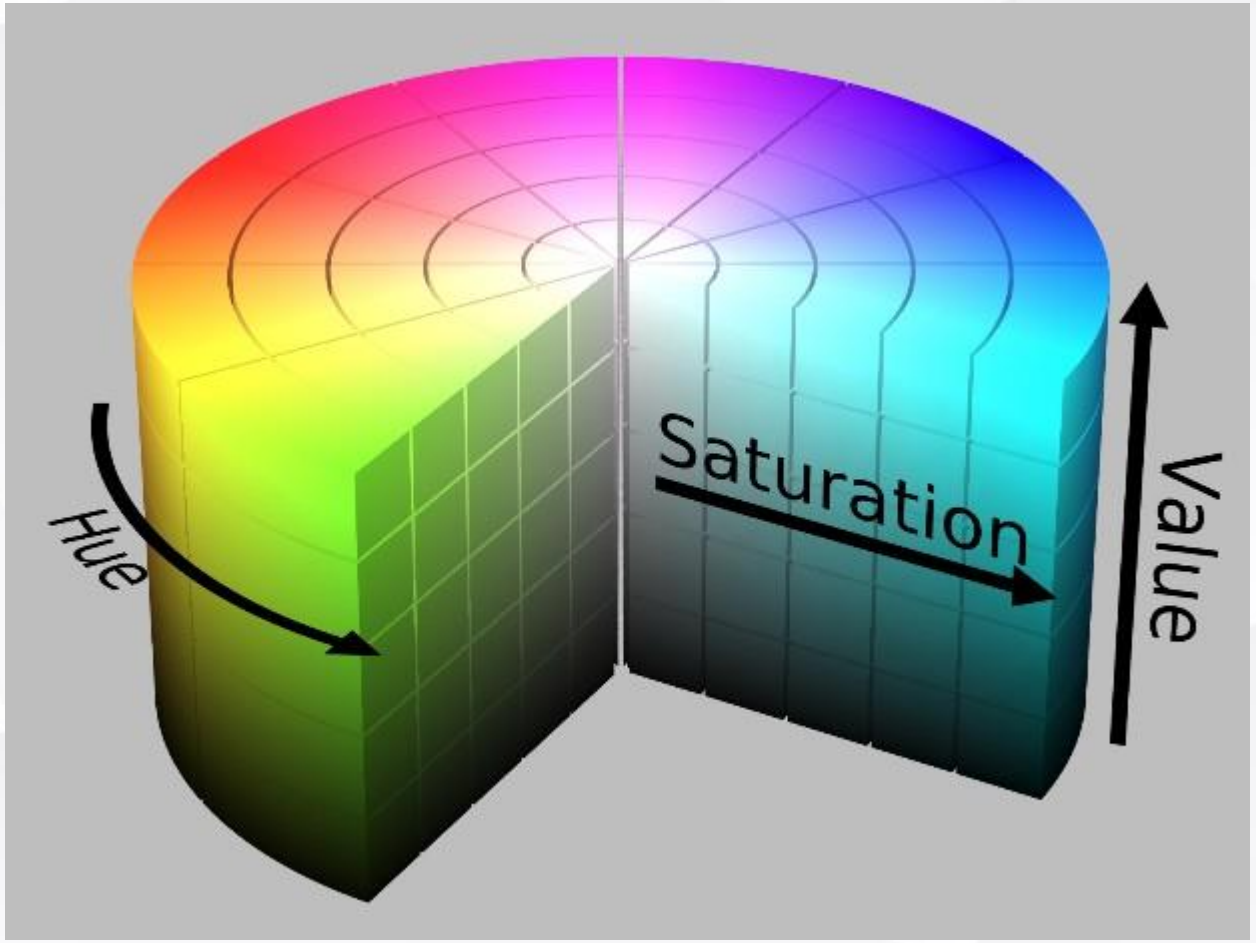


- H参数表示色彩信息，即所处的光谱颜色的位置。用一角度值来表示，红、绿、蓝分别相隔120°
- S是一个比例值，S=0时只有灰度；V表示色彩的明亮程度，它和光强度之间并没有直接的联系。

RGB和CMY颜色模型是面向硬件的　　HSV颜色模型是面向用户的

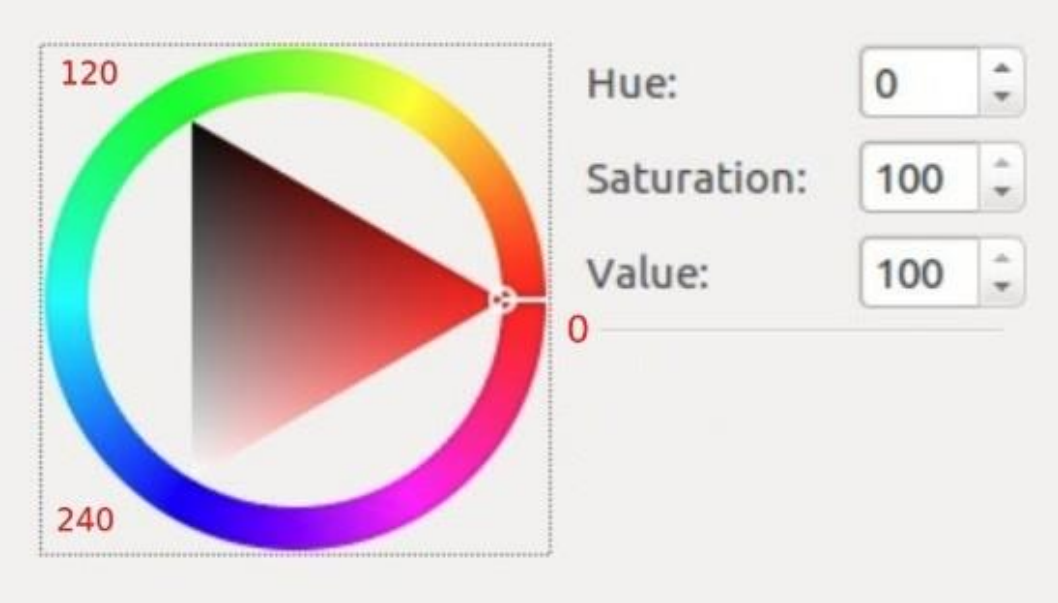
HSV颜色空间的圆柱体表示

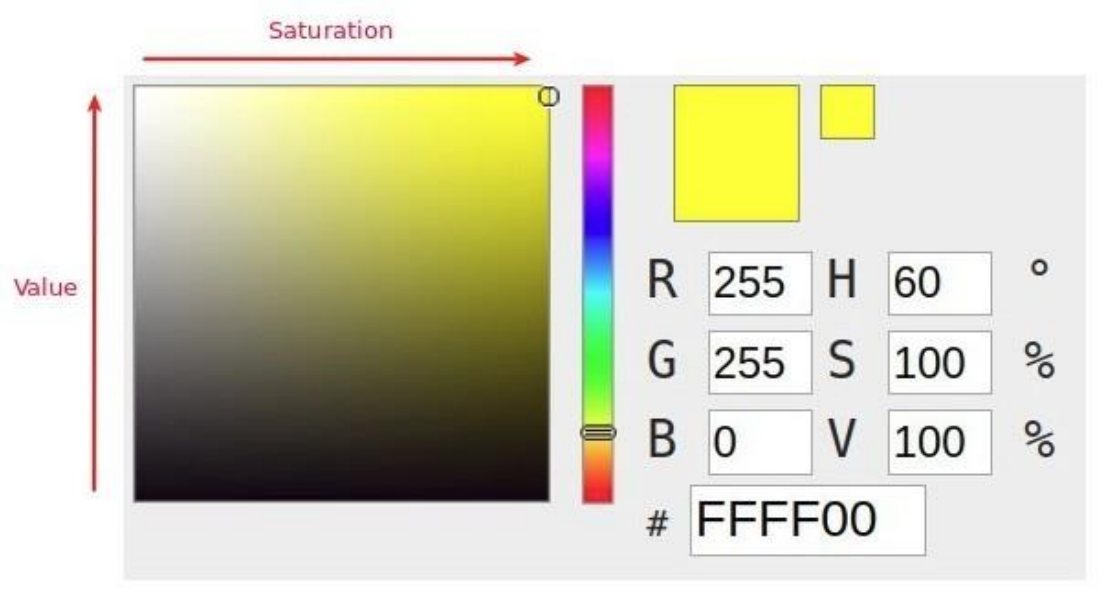
- 用下面这个圆柱体来表示 HSV 颜色空间，圆柱体的横截面可以看做是一个极坐标系，H 用极坐标的极角表示，S 用极坐标的极轴长度表示，V 用圆柱中轴的高度表示。



HSV——Hue值

- Hue 用角度度量，取值范围为0～360°，表示色彩信息，即所处的光谱颜色的位置。，表示如下：





- 颜色圆环上所有的颜色都是光谱上的颜色，从红色开始按逆时针方向旋转，Hue=0 表示红色，Hue=120 表示绿色，Hue=240 表示蓝色等等。
- 在RGB颜色空间中颜色由三个值共同决定，比如黄色为即 (255,255,0)；在HSV中，黄色只由一个值决定，Hue=60即可，即HSV 圆柱体的半边横截面（Hue=60），如上图右所示。

HSV——S和V值的意义

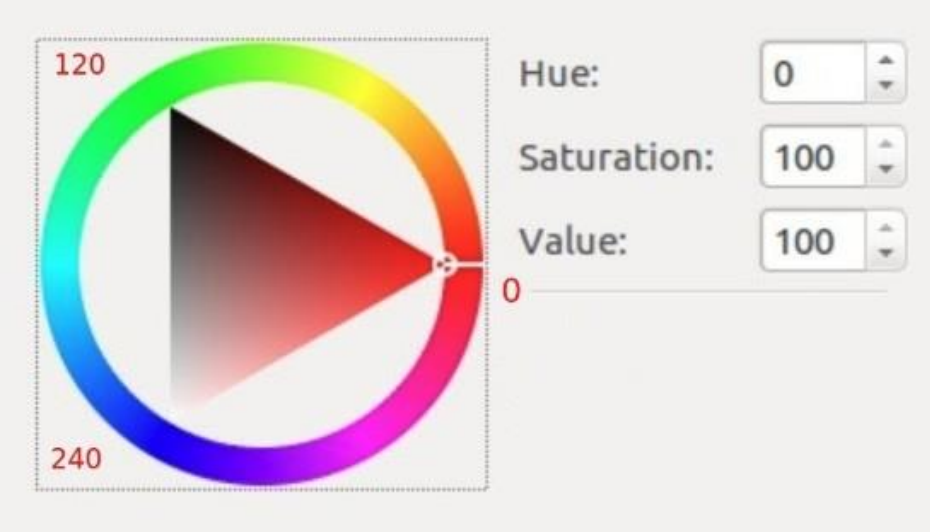
- 圆柱体截面的水平方向（即S值）表示**饱和度**，饱和度表示颜色接近光谱色的程度。
- 一种颜色，可以看成是某种**光谱色与白色**混合的结果。其中光谱色所占的比例愈大，颜色接近光谱色的程度就愈高，颜色的饱和度也就愈高。饱和度高，颜色则深而艳。也就是说：
  - 饱和度越高，说明颜色越深，越接近光谱色；饱和度越低，说明颜色越浅，越接近白色。
  - 饱和度为0表示纯白色。取值范围为**0～100%**，值越大，颜色越饱和。
- 圆柱体竖直方向（即V值）表示**明度**，决定颜色空间中颜色的明暗程度，明度越高，表示颜色越明亮，范围是 **0-100%**。明度为0表示纯黑色（此时颜色最暗）。

HSV颜色空间的意义

- HSV**对用户**来说是一种比较直观的颜色模型：
  - 我们可以很轻松地得到单一颜色，即指定颜色角H，并让V=S=1。
  - 然后通过向其中加入黑色和白色来得到我们需要的颜色。增加黑色可以减小V而S不变，同样增加白色可以减小S而V不变。
  - 例如，要得到深蓝色，V=0.4 S=1 H=240度。要得到浅蓝色，V=1 S=0.4 H=240度。
- HSV 的**拉伸、对比度增强**就是对 S 和 V 两个分量进行归一化(min-max normalize)即可，H 保持不变。
- RGB颜色空间更加面向于硬件，适合显示图像；而HSV更加面向于用户，大多数做图像识别这一块的都会运用HSV颜色空间，因为HSV颜色空间表达起来更加直观！

OpenCV中的HSV颜色空间的理解

- 在 OpenCV 中 HSV 三个分量的取值范围为：
  - H = [0,179]
  - S = [0,255]
  - V = [0,255]
- S和V的取值比较容易理解，将HSV颜色空间中S和V的取值范围呈上255即可。
- 但H的取值0~179，其实是把HSV颜色空间在S分量上进行了压缩，可以这样理解：
  - 在S颜色环上，从0°处切开，拉直，然后压缩得到。
  - **所以在OpenCV中，颜色空间的两头都有红色分量即0°~5°和175°~179°**



HSV和RGB颜色空间的直观对比举例

- 下面代码分别输出鼠标点击处像素的BGR值和HSV值，通过实验观测可以加深体会

```
import cv2
import numpy as np
cameraCapture = cv2.VideoCapture(0,cv2.CAP_DSHOW)
cv2.namedWindow('frame')
#cv2.namedWindow('frameHSV')
success,frame = cameraCapture.read()
hsv = cv2.cvtColor(frame,cv2.COLOR_BGR2HSV)

def GetPosHsv(event,x,y,flags,param):
    if event == cv2.EVENT_LBUTTONDOWN:
        print('HSV is',hsv[y,x])
def GetPosBgr(event,x,y,flags,param):
    if event == cv2.EVENT_LBUTTONDOWN:
        print('BGR is',frame[y,x])
        print('HSV is',hsv[y,x])
cv2.setMouseCallback('frame',GetPosBgr)
#cv2.setMouseCallback('frameHSV',GetPosHsv)

while success and cv2.waitKey(1) == -1:
    cv2.imshow('frame',frame)
    #cv2.imshow('frameHSV',hsv)
    success,frame = cameraCapture.read()
    hsv = cv2.cvtColor(frame,cv2.COLOR_BGR2HSV)
cv2.destroyWindow('frame')
#cv2.destroyWindow('frameHSV')
cameraCapture.release()
```

- 注意观测两点：

1. HSV颜色空间对颜色的判断不容易受光照影响，而RGB颜色空间对光照、阴影等特别敏感
2. HSV颜色空间是面向用户的，不适合用于显示图像，适用于图像处理。

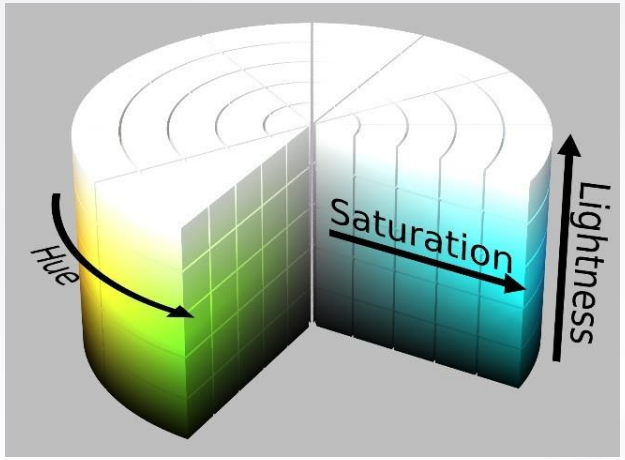
课堂实验：HSV空间颜色识别（4分）

- 实验要求：在视频图像上圈出颜色小球。

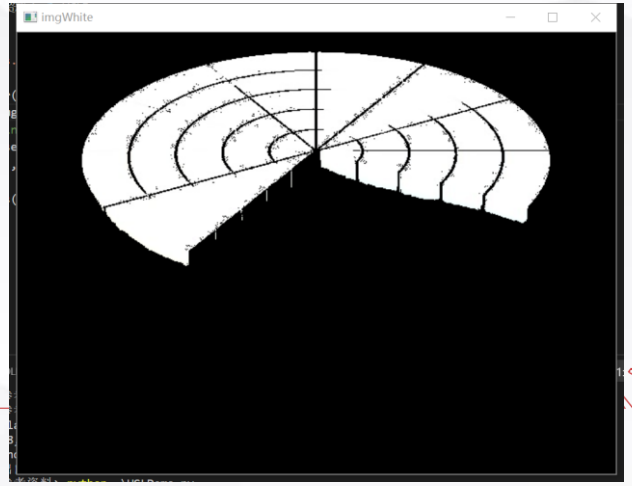
```python
import numpy as np
import cv2
import time
import copy

# 设定红色阈值，HSV空间
redLower1 = np.array([0, 43, 46])
redUpper1 = np.array([4, 255, 255])
redLower2 = np.array([175,43,46])
redUpper2 = np.array([180,255,255])
# 打开摄像头
camera = cv2.VideoCapture(0)
# 等待两秒
time.sleep(2)
# 遍历每一帧，检测红色瓶盖
while True:
    # 读取帧
    (ret, frame) = camera.read()
    # 判断是否成功打开摄像头
    if not ret:
        print('No Camera')
        break
    frame = cv2.flip(frame,1) # 完成镜像，让使用更加友好
    # 转到HSV空间
    hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)
    # 根据阈值构建掩膜
    #mask = cv2.inRange(hsv,redLower2,redUpper2)
    mask = cv2.inRange(hsv, redLower1, redUpper1) + cv2.inRange(hsv,redLower2,redUpper2)
    cv2.imshow('Frame2', mask)
    # 腐蚀操作
    mask = cv2.erode(mask, None, iterations=2)
    # 膨胀操作，其实先腐蚀再膨胀的效果是开运算，去除噪点
    mask = cv2.dilate(mask, None, iterations=2)
    #cv2.imshow('Frame3', mask)
    # 轮廓检测
    cnts = cv2.findContours(mask.copy(), cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)[-2]
    # 初始化瓶盖圆形轮廓质心
    center = None
    # 如果存在轮廓
    if len(cnts) > 0:
        # 找到面积最大的轮廓
        c = max(cnts, key=cv2.contourArea)
        # 确定面积最大的轮廓的外接圆
        ((x, y), radius) = cv2.minEnclosingCircle(c)
        # 只有当半径大于5时，才执行画图
        if radius > 5:
            cv2.circle(frame, (int(x), int(y)), int(radius), (0, 255, 255), 2)
    cv2.imshow('Frame', frame)
    # 键盘检测，检测到esc键退出
    k = cv2.waitKey(5) & 0xFF
    if k == 27:
        break
# 摄像头释放
camera.release()
# 销毁所有窗口
cv2.destroyAllWindows()
```

HLS颜色空间

- HLS 和 HSV 比较类似，也有三个分量，hue（色相）、lightness（亮度）、saturation（饱和度）。
- HLS 和 HSV 的区别就是，HLS 用的是 light(亮度)，HSV 用的是 value(明度)。
- HLS 中的 L 分量为亮度，亮度为100%，表示白色，亮度为0，表示黑色；HSV 中的 V 分量为明度，明度为100%，表示光谱色，明度为0，表示黑色。
- 提取白色物体时，使用 HLS 更方便，因为 HSV 中的Hue里没有白色，白色需要由S和V共同决定（S=0, V=100%）。而在 HLS 中，白色仅由亮度L一个分量决定。所以检测白色时使用 HSL 颜色空间更准确。
- 将改图存储为hls.jpg，然后用HLS颜色空间对其处理。




```python
import cv2
import numpy as np
img = cv2.imread("hls.jpg")
# Convert BGR to HLS
imgHLS = cv2.cvtColor(img, cv2.COLOR_BGR2HLS)
mask = cv2.inRange(imgHLS, np.array([0,250,0]), np.array([255,255,255]))
# Apply Mask to original image
imgWhite = cv2.bitwise_and(img, img, mask=mask) #位与操作检出白色
cv2.imshow('imgWhite',imgWhite)
cv2.waitKey(0)
cv2.destroyAllWindows()
```



课堂实验：基于二维云台跟踪红色物体（5.5分）

```python
import cv2
import time
import numpy as np
import Adafruit_PCA9685
import threading
servo = Adafruit_PCA9685.PCA9685()
servo.set_pwm_freq(50)
servo.set_pwm(4,0,307)
servo.set_pwm(5,0,307)
time.sleep(1)
pid_thisError_x = 0
pid_lastError_x = 0
pid_thisError_y = 0
pid_lastError_y = 0
current_x = 320
current_y = 240
pid_Y_P = 307
pid_X_P = 307
pid_flag = False
ball_red_lower1 = np.array([170,43,46])
ball_red_upper1 = np.array([179,255,255])
ball_red_lower2 = np.array([0,43,46])
ball_red_upper2 = np.array([5,255,255])
def ServoControlPID(X_P,Y_P):
    while pid_flag:
        servo.set_pwm(4,0,600-pid_X_P)
        servo.set_pwm(5,0,pid_Y_P)
cameraCapture = cv2.VideoCapture(0)
cameraCapture.set(3,640)
cameraCapture.set(4,480)
cv2.namedWindow('MyWindow')
print('Showing camera feed.click window or press any key to stop')
success,frame = cameraCapture.read()
servo_tid = threading.Thread(target=ServoControlPID,args=(pid_X_P,pid_Y_P))
servo_tid.setDaemon(True)
pid_flag = True
servo_tid.start()
while success and cv2.waitKey(1) == -1:   #只要有键盘按下就退出
    hsv = cv2.cvtColor(frame,cv2.COLOR_BGR2HSV)
    mask = cv2.inRange(hsv,ball_red_lower1,ball_red_upper1)
    mask = cv2.erode(mask,None,iterations = 2)
    mask = cv2.dilate(mask,None,iterations = 2)
    cnts = cv2.findContours(mask.copy(),cv2.RETR_EXTERNAL,cv2.CHAIN_APPROX_SIMPLE)[-2]
    if len(cnts) > 0:
        cap_cnt = max(cnts,key=cv2.contourArea)
        (current_x,current_y),radius = cv2.minEnclosingCircle(cap_cnt)
        print('current_x=%d,current_y=%d,radius=%d'%(current_x,current_y,radius))
        cv2.circle(frame,(int(current_x),int(current_y)),int(radius),(255,0,0),2)
        pid_thisError_x = current_x - 320
        pid_thisError_y = current_y - 240
```

颜色识别及追踪实验：基于二维云台跟踪红色物体

```
51            print('pid_thisError_x=%d,pid_thisError_y=%d'%(pid_thisError_x,pid_thisError_y))
52            pwm_x = pid_thisError_x * 3 + 1 * (pid_thisError_x - pid_lastError_x)
53            pwm_y = pid_thisError_y * 3 + 1 * (pid_thisError_y - pid_lastError_y)
54            print('pwm_x=%d,pwm_y=%d'%(pwm_x,pwm_y))
55            pid_lastError_x = pid_thisError_x
56            pid_lastError_y = pid_thisError_y
57            pid_XP = pwm_x/100
58            pid_YP = pwm_y/100
59            print('pid_XP=%d,pid_YP=%d'%(pid_XP,pid_YP))
60            pid_X_P = pid_X_P + int(pid_XP)
61            pid_Y_P = pid_Y_P + int(pid_YP)
62            print('pid_X_P=%d,pid_Y_P=%d'%(pid_X_P,pid_Y_P))
63            if pid_X_P > 600:
64                pid_X_P = 600
65            if pid_X_P < 0:
66                pid_X_P = 0
67            if pid_Y_P > 500:
68                pid_Y_P = 500
69            if pid_Y_P < 125:
70                pid_Y_P = 125
71        cv2.imshow('MyWindow',frame)
72        success,frame = cameraCapture.read()
73    cv2.destroyWindow('MyWindow')
74    cameraCapture.release()
75    pid_flag = False
```

06

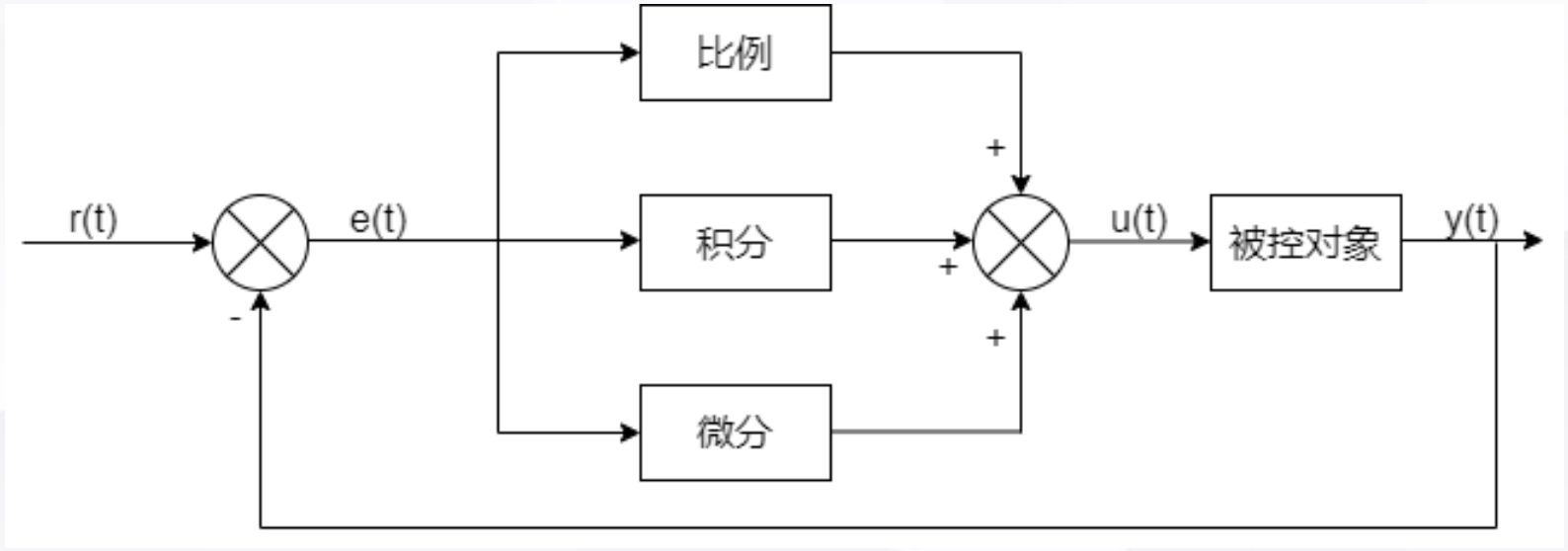
PID控制

PID控制原理

离散PID控制

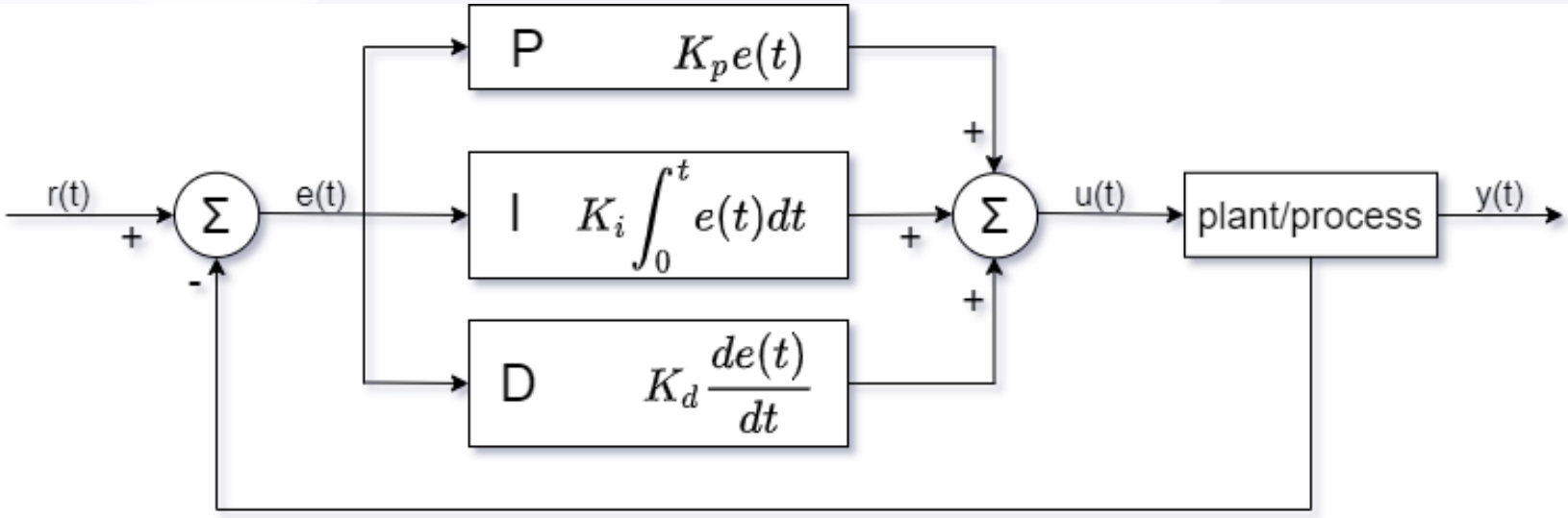
PID控制原理

- 比例、积分、微分控制，简称PID控制，其算法简单、鲁棒性好和可靠性高，被广泛应用于工业过程控制。经典控制理论在实际控制系统中的典型应用就是PID控制器。
- PID控制器是一种线性控制器，简单的说，根据给定值和实际输出值构成控制**偏差**，将偏差按比例、积分和微分通过**线性组合**构成控制量，对被控对象进行控制。这是一个典型的单位**负反馈**控制系统。



PID控制原理的数学描述

- PID控制器是一种线性控制器，它根据给定值r(t)与实际输出值c(t)构成偏差：e(t)=r(t)-c(t)。将偏差的比例(P)、积分(I)和微分(D)通过线性组合构成控制量，对受控对象进行控制。其控制规律为：



- $u(t) = K_p\left[\mathrm{e}(t) + \frac{1}{T_i}\int_0^t \mathrm{e}(t)\,\mathrm{d}(t) + T_d\frac{\mathrm{de}(t)}{\mathrm{d}t}\right] = \boldsymbol{K_p e(t) + K_i\int_0^t \mathrm{e}(t)\,\mathrm{dt} + K_d\frac{\mathrm{de}(t)}{\mathrm{dt}}}$

- 式中，K_p为比例系数，T_i为积分时间常数，T_d为微分时间常数；$\boldsymbol{K_i}$=K_p/T_i，为积分系数；$\boldsymbol{K_d}$=$\boldsymbol{K_p}$*T_d，为微分系数。

PID控制各环节的作用

- 比例环节：即时成比例地反应控制系统的偏差信号e(t)，偏差一旦产生，控制器立即产生控制作用以减小误差。当偏差e=0时，控制作用也为0。因此，比例控制是基于偏差进行调节的，即**有差调节**。
- 积分环节：能对误差进行记忆，主要用于消除静差，提高系统的无差度，积分作用的强弱取决于积分时间常数Ti，Ti越大，积分作用越弱，反之则越强。
- 微分环节：能反映偏差信号的变化趋势(变化速率)，并能在偏差信号值变得太大之前，在系统中引入一个有效的早期修正信号，从而加快系统的动作速度，减小调节时间。
- 从时间的角度讲，比例作用是针对系统当前误差进行控制，积分作用则针对系统误差的历史，而微分作用则反映了系统误差的变化趋势，这三者的组合是“过去、现在、未来”的完美结合。

离散PID

- 前面介绍的是PID控制的基本原来，是连续系统的PID控制算式。
- 计算机控制系统是一个采样控制系统，它只能根据采样时刻的偏差值来计算控制量，yin在计算机控制系统中,必须对公式进行离散化，具体就是用求和代替积分，用向后差分来代替微分，使模拟PID离散化为数字形式的差分方程。
- 假设采样周期为T，则在k时刻：
  - 偏差为e(k)
  - 积分为e(k)+e(k-1)+e(k-2)+...+e(0)
  - 微分为(e(k)-e(k-1))/T
- $$u(k) = k_p \mathrm{e}(\mathrm{k}) + k_i \sum_{n=0}^{k} e(k) + k_d\big(\mathrm{e}(k) - \mathrm{e}(k-1)\big)$$
- 这就是离散的PID控制算式，在采样周期T远小于信号变化周期的情况下，这种逼近是非常准确的。
- 可见，k时刻的控制量与之前的全部采样时刻的偏差值都有关系，也就是我们常说的**位置式PID**。
- 离散的位置式PID，运算量比较大，一个累加就有可能导致嵌入式控制器内存不足。此外，一旦掉电，有可能丢失之前的所有状态。
- 如果，控制系统不需要控制量的绝对值，那么只需要计算增加量即可。

离散PID——增量式PID

- 根据离散的位置式PID算式：

$$u(k) = k_p \mathrm{e}(\mathrm{k}) + k_i \sum_{n=0}^{k} e(k) + k_d\big(\mathrm{e}(k) - \mathrm{e}(k-1)\big)$$

- k-1时刻

$$u(k-1) = k_p \mathrm{e}(\mathrm{k}-1) + k_i \sum_{n=0}^{k-1} e(k-1) + k_d\big(\mathrm{e}(k-1) - \mathrm{e}(k-2)\big)$$

- 推导出偏差值增量：

$$\Delta u(k) = u(k) - u(k-1) = k_P, \mathrm{e}(k) - \mathrm{e}(k-1) + k_i \mathrm{e}(k) + k_d(\mathrm{e}(k) - 2\mathrm{e}(\mathrm{k}-1) + \mathrm{e}(\mathrm{k}-2))$$

- 上式就是增量式PID的表现形式，计算出来的增量只跟**最近三次**的偏差值有关。
- 注意这里计算出来的是增量值，也就是说如果我们要求u(k)的话应该是：

$$u(k) = u(k-1) + \Delta u(k)$$

PID控制实例分析

- PID调节器出现于上世纪30年代，PID的三个单元分别是比例，积分，微分。在工程实际中，一般P是必须的，所以衍生出多种组合的PID控制器，如PD，PI，PID等。
- 例如，在颜色追踪实验中，使用的就是离散的增量式PI控制方法：

```
(current_x,current_y),radius = cv2.minEnclosingCircle(cap_cnt)
 cv2.circle(frame,(int(current_x),int(current_y)),int(radius),(255,0,0),2)
 pid_thisError_x = current_x - 160
 pid_thisError_y = current_y - 120
 pwm_x = pid_thisError_x * 3 + 1 * (pid_thisError_x - pid_lastError_x)
 pwm_y = pid_thisError_y * 3 + 1 * (pid_thisError_y - pid_lastError_y)
 pid_XP = pwm_x/100
 pid_YP = pwm_y/100
 pid_X_P = pid_X_P + int(pid_XP)
 pid_Y_P = pid_Y_P + int(pid_YP)
 pid_lastError_x = pid_thisError_x
 pid_lastError_y = pid_thisError_y
```

07

人脸检测

分类器

人脸检测

人脸追踪

人脸检测

- 人脸检测属于计算机视觉的范畴，早期人们的主要研究方向是**人脸识别**，即根据人脸来识别人物的身份，后来在复杂背景下的人脸检测需求越来越大，人脸检测也逐渐作为一个单独的研究方向发展起来。
- 目前人脸检测的方法主要有两大类：基于知识和基于统计。
  - 基于知识的方法：主要利用先验知识将人脸看作器官特征的组合，根据眼睛、眉毛、嘴巴、鼻子等器官的特征以及相互之间的几何位置关系来检测人脸。主要包括模板匹配、人脸特征、形状与边缘、纹理特性、颜色特征等方法。
  - 基于统计的方法：将人脸看作一个整体的模式——二维像素矩阵，从统计的观点通过大量人脸图像样本构造人脸模式空间，根据相似度量来判断人脸是否存在。主要包括主成分分析与特征脸、神经网络方法、支持向量机、隐马尔可夫模型、Adaboost算法等。
- 接下来将要介绍Haar分类器方法，就包含了Adaboost算法。所谓分类器，在这里就是指对人脸和非人脸进行分类的算法，在机器学习领域，很多算法都是对事物进行分类、聚类的过程。OpenCV中的ml模块提供了很多分类、聚类的算法。

Haar分类器方法

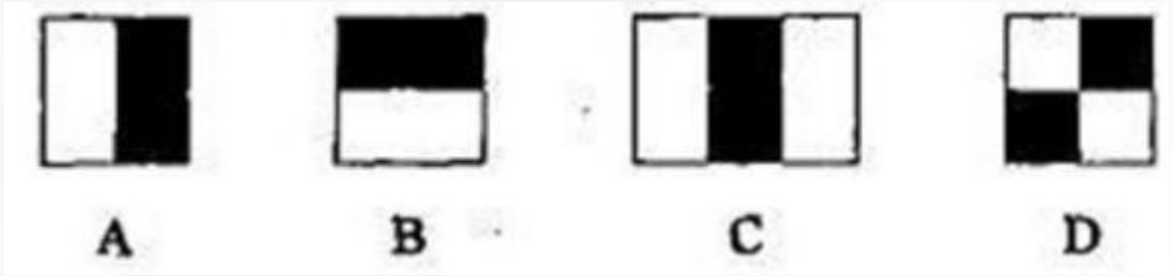
- Haar分类器方法主要用来实现对刚性物体进行检测的分类器的训练。
- 一个好的分类器能够很灵敏的对目标物体进行识别和跟踪，但它的训练需要数量庞大的正样本与负样本。
- 分类器的训练主要分为四步：

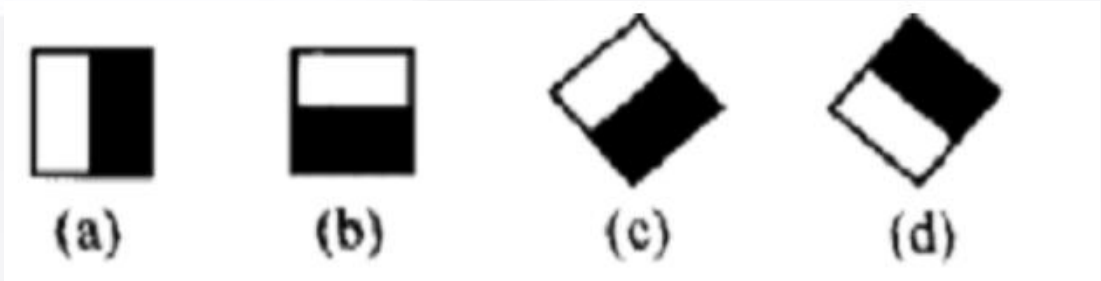
1. 样本的创建
2. 分类器的训练
3. **xml文件生成**
4. 对训练好的分类器的效果的检测

- Haar分类器方法 = Haar-like特征 + 积分图方法 + AdaBoost +级联。
- Haar分类器算法的要点如下：
  - 使用Haar-like特征做检测。
  - 使用积分图（Integral Image）对Haar-like特征求值进行加速。
  - 使用AdaBoost算法训练区分人脸和非人脸的强分类器。
  - 使用筛选式级联把强分类器级联到一起，提高准确率。。

Haar-like特征

- Haar(哈尔)特征分为：边缘特征、线性特征、中心特征和对角线特征，组合成**特征模板**。
- A、B是边缘特征；C是线性特征；D是对角线特征





边界特征

中心特征

线性特征

- 特征模板内有白色和黑色两种矩形，并定义该模板的**特征值**为白色矩形像素和减去黑色矩形像素和。

特征模板称为“特征原型”；特征原型在图像子窗口中扩展（**平移伸缩**）得到的特征称为“矩形特征”；矩形特征的值称为**“特征值”**。

Haar-like特征的计算

- 这些所谓的特征不就是一堆堆带条纹的矩形么，到底是干什么用的?
  - 解释如下：将上面的任意一个矩形放到人脸区域上，然后，将白色区域的像素和减去黑色区域的像素和，得到的值我们暂且称之为**人脸特征值**，如果你把这个矩形放到一个非人脸区域，那么计算出的特征值应该和人脸特征值是不一样的，而且越不一样越好，所以这些方块的目的就是把人脸特征量化，以区分人脸和非人脸。我们希望当把矩形放到人脸区域计算出来的特征值和放到非人脸区域计算出来的特征值差别越大越好，这样就可以用来区分人脸和非人脸。
- 矩形特征可位于图像任意位置，大小也可以任意改变，所以矩形特征值是矩形模版类别、矩形位置和矩形大小这三个因素的函数。故类别、大小和位置的变化，使得很小的检测窗口含有非常多的矩形特征，如：在24*24像素大小的检测窗口内矩形特征数量可以达到16万个。
- 这样就有两个问题需要解决了：
  1. 如何快速计算那么多的特征？——通过积分图；
  2. 哪些矩形特征才是对分类器分类最有效的？——通过AdaBoost算法来训练。

人脸检测实验代码

```python
import cv2
import numpy as np
face_cascade = cv2.CascadeClassifier('C:/Program Files/Python38/Lib/site-packages/cv2/data/haarcascade_frontalface_default.xml')
#eye_cascade = cv2.CascadeClassifier('C:/Program Files/Python38/Lib/site-packages/cv2/data/haarcascade_eye.xml')
cap = cv2.VideoCapture(0,cv2.CAP_DSHOW)
while True:
    ret,img = cap.read()
    img = cv2.flip(img,1)
    gray = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
    faces = face_cascade.detectMultiScale(gray,1.3,5)
    for (x,y,w,h) in faces:
        cv2.rectangle(img,(x,y),(x+w,y+h),(255,0,0),2)
        roi_gray = gray[y:y+h,x:x+w]
        roi_color = img[y:y+h,x:x+w]
        print(int(x+w/2),int(y+h/2))
        '''
        eyes = eye_cascade.detectMultiScale(roi_gray)
        for (ex,ey,ew,eh) in eyes:
            cv2.rectangle(roi_color,(ex,ey),(ex+ew,ey+eh),(0,255,0),2)
        '''
    cv2.imshow('img',img)
    k = cv2.waitKey(20) & 0xff
    if k == 27: #press 'ESC' to quit
        break
cap.release()
cv2.destroyAllWindows()
```

人脸追踪

- 请根据之前学习过的实验内容自主完成该实验。
- 参考颜色追踪、PID控制、人脸检测等实验内容和代码

本讲结束

饮 水 思 源　爱 国 荣 校

上海交通大學

SHANGHAI JIAO TONG UNIVERSITY

感谢聆听

饮水思源 爱国荣校

上海交通大学

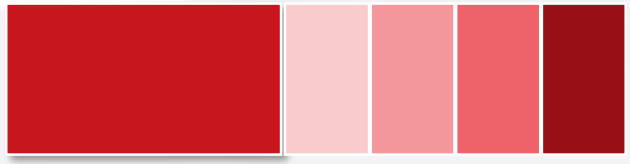
SHANGHAI JIAO TONG UNIVERSITY

感谢聆听

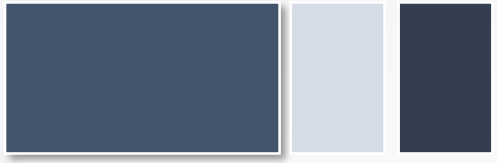
饮 水 思 源　爱 国 荣 校

色彩规范

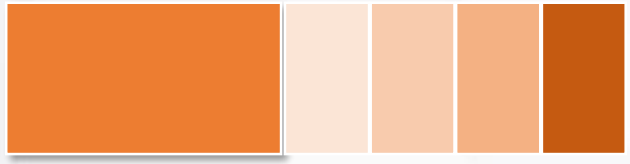
主色



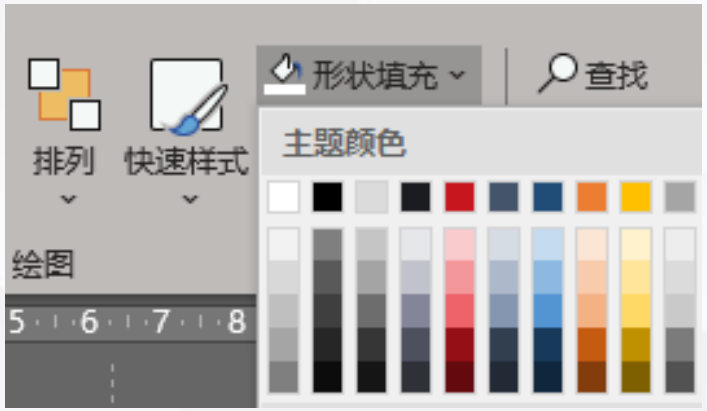
对比色



平衡色



浅色与深色



建议尽量选择以上调色板中的颜色

字体规范

中文标题	中文正文
微软雅黑	微软雅黑

英文标题	英文正文
Arial	Arial

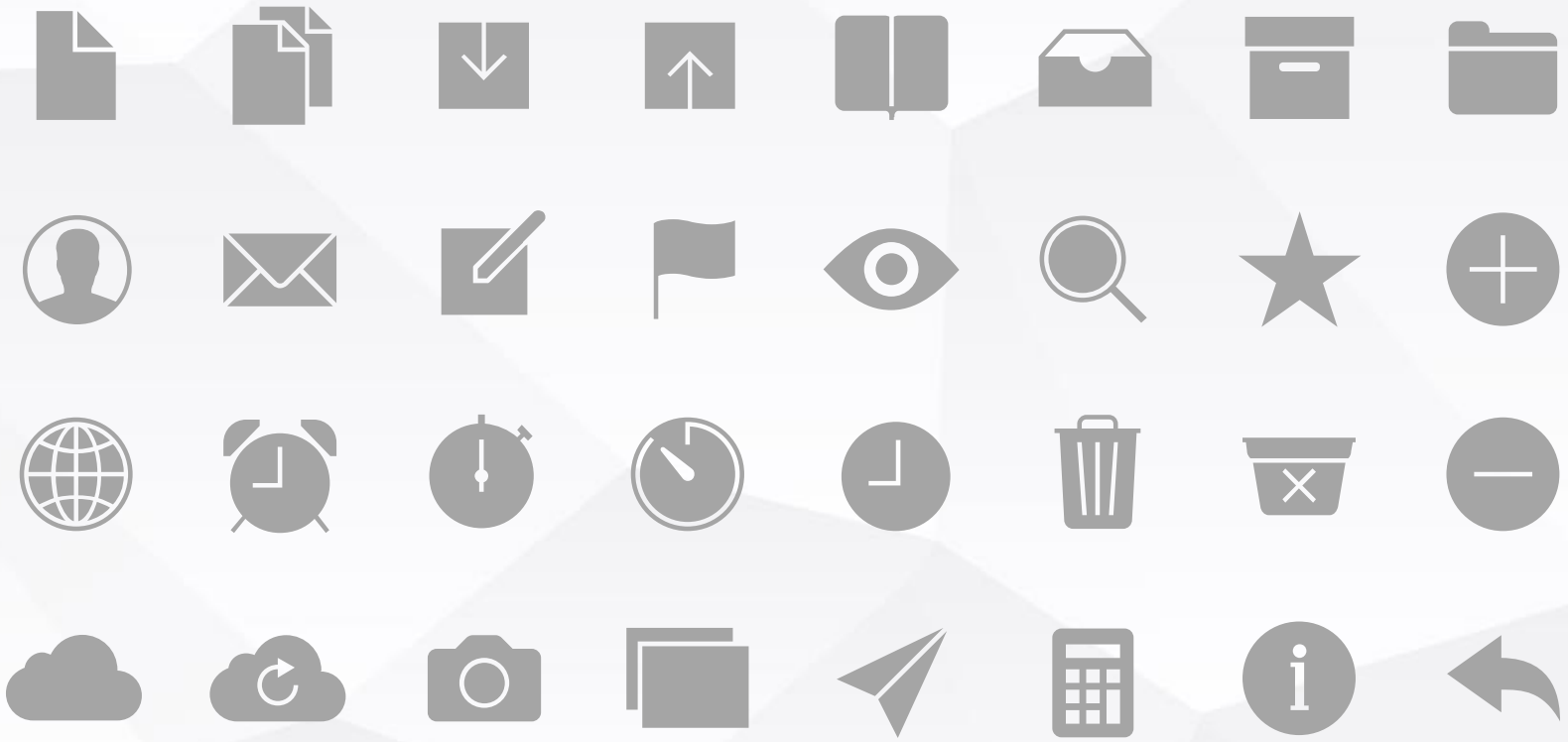
注意：

微软雅黑属于版权字体，商用请购买！

更多免费商用字体
https://jbox.sjtu.edu.cn/l/WuCIHQ

图标

图标



标注

使用说明

本PPT模板为作者原创，著作权归作者所有。
您仅可以个人非商业用途使用本PPT模板，未经权利人书面明确授权，不可将信息内容的全部或部分用于出售，或以出租、出借、转让、分销、发布等其他任何方式供他人使用，否则将承担法律责任。

声明

OfficePLUS尊重知识产权并注重保护用户享有的各项权利。
OfficePLUS拥有对本PPT模板进行展示、报道、宣传及用于市场活动的权利，若在比赛或商业应用过程中发生版权纠纷，其法律责任由作者本人承担。