Chapter17 颜色识别与云台追踪

从这开始会用到计算机视觉的知识，我们可以直接使用opencv来进行图像处理。

opencv相关知识见Chapter17。

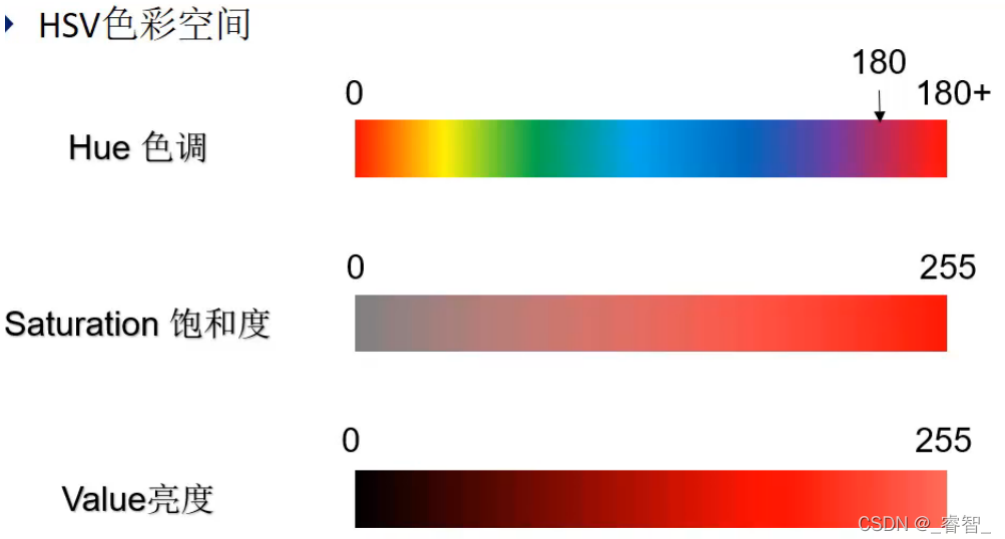
颜色识别

我们从基本的颜色识别来入手，从摄像头中捕获我们想要的颜色。

RGB与HSV

在opencv中，使用BGR来表示图像。但我们要将每一帧由BGR转换为HSV来进行处理，原因是：

RGB通道并不能很好地反映出物体具体的颜色信息，而HSV能够非常直观的表达色彩的明暗、色调、以及鲜艳程度，方便进行颜色之间的对比，而且RGB受光线影响很大，所以采取HSV）



Hue（H）：色调、色相（具体的颜色）

Saturation（S）：饱和度、色彩纯净度

Value（V）：明度

Hue范围是[0,179]，饱和范围是[0,255]，值范围是[0,255]

HSV颜色对照表：

	黑	灰	白	红		橙	黄	绿	青	蓝	紫
hmin	0	0	0	0	156	11	26	35	78	100	125
hmax	180	180	180	10	180	25	34；	77	99	124	155
smin	0	0	0	43		43	43	43	43	43	43
smax	255	43	30	255		255	255	255	255	255	255
vmin	0	46	221	46		46	46	46	46	46	46
vmax	46	220	255	255		255	255	255	255	255	255

API

- cv2.inRange(hsv, color_lower, color_upper)

  将区间内的像素置为白色，其余置为黑色。

- cv2.bitwise_and(image,image,mask=mask)

  掩膜mask一般为黑白二值图像，与image相与，起到提取特定区域的作用

代码实现

```
import cv2 as cv
import numpy as np

cap = cv.VideoCapture(0)
print(cap.isOpened())

str='''请输入你想识别的颜色
red
blue
yellow
green
orange
'''

# 红色区间
red_lower = np.array([0, 43, 46])
red_upper = np.array([10, 255, 255])

# #绿色区间
green_lower = np.array([35, 43, 46])
green_upper = np.array([77, 255, 255])

# #蓝色区间
blue_lower=np.array([100, 43, 46])
blue_upper = np.array([124, 255, 255])

# #黄色区间
yellow_lower = np.array([26, 43, 46])
yellow_upper = np.array([34, 255, 255])
```

```
# #橙色区间
orange_lower = np.array([11, 43, 46])
orange_upper = np.array([25, 255, 255])

color_dict={"red":[red_lower,red_upper],"green":[green_lower,green_upper],
            "blue":[blue_lower,blue_upper],"yellow":[yellow_lower,yellow_upper]}

def color_recognize():
    choice=input(str)
    while (1):
        ret, frame = cap.read()
        if not ret:
            print("wrong")
            break
        hsv = cv.cvtColor(frame, cv.COLOR_BGR2HSV)
        color_lower,color_upper=color_dict[choice]
        #获取掩膜mask
        mask = cv.inRange(hsv, color_lower, color_upper)
        img = cv.bitwise_and(frame, frame, mask=mask)
        cv.imshow('mask', mask)
        cv.imshow("res",img)
        if cv.waitKey(1000//60) & 0xFF == ord('q'):
            break

    cap.release()
    cv.destroyAllWindows()

color_recognize()
```

云台追踪

我们使用两个舵机，分别负责x轴和y轴的旋转。

为了保证追踪的稳定，我们引入PID控制算法，使得云台追踪更为稳定。

PID控制算法

PID控制电机

位置式PID：

```
class PositionalPID:
    def __init__(self, P, I, D):
        self.Kp = P
        self.Ki = I
        self.Kd = D

        self.SystemOutput = 0.0
        self.ResultValueBack = 0.0
        self.PidOutput = 0.0
        self.PIDErrADD = 0.0
        self.ErrBack = 0.0


    def SetStepSignal(self,StepSignal):
        Err = StepSignal - self.SystemOutput
        KpWork = self.Kp * Err
        KiWork = self.Ki * self.PIDErrADD
        KdWork = self.Kd * (Err - self.ErrBack)
        self.PidOutput = KpWork + KiWork + KdWork
        self.PIDErrADD += Err
        if self.PIDErrADD > 2000:
            self.PIDErrADD = 2000
        if self.PIDErrADD < -2500:
            self.PIDErrADD = -2500
        self.ErrBack = Err


    def SetInertiaTime(self, InertiaTime,SampleTime):
        self.SystemOutput = (InertiaTime * self.ResultValueBack + \
        SampleTime * self.PidOutput) / (SampleTime + InertiaTime)
        self.ResultValueBack = self.SystemOutput
```

增量式PID：

```
class IncrementalPID:
    def __init__(self, P, I, D):
        self.Kp = P
        self.Ki = I
        self.Kd = D

        self.PIDOutput = 0.0
        self.SystemOutput = 0.0
        self.LastSystemOutput = 0.0

        self.Error = 0.0
        self.LastError = 0.0
```

```
        self.LastLastError = 0.0


    def SetStepSignal(self,StepSignal):
        self.Error = StepSignal - self.SystemOutput
        IncrementValue = self.Kp * (self.Error - self.LastError) +\
        self.Ki * self.Error +\
        self.Kd * (self.Error - 2 * self.LastError + self.LastLastError)

        self.PIDOutput += IncrementValue
        self.LastLastError = self.LastError
        self.LastError = self.Error


    def SetInertiaTime(self,InertiaTime,SampleTime):
        self.SystemOutput = (InertiaTime * self.LastSystemOutput + \
            SampleTime * self.PIDOutput) / (SampleTime + InertiaTime)

        self.LastSystemOutput = self.SystemOutput
```

我们使用位置式PID。

代码实现

```
import RPi.GPIO as GPIO
import cv2 as cv
import time
import numpy as np
import PID

# 舵机引脚定义
servoPin1 = 11  # S2
servoPin2 = 9   # S3

color_x = color_y = color_radius = 0
target_valuex = 1400
target_valuey = 1500

xservo_pid = PID.PositionalPID(0.1, 0.2, 0.1)
yservo_pid = PID.PositionalPID(0.1, 0.2, 0.1)

# 红色区间
red_lower = np.array([0, 43, 46])
red_upper = np.array([10, 255, 255])

# #绿色区间
green_lower = np.array([35, 43, 46])
green_upper = np.array([77, 255, 255])

# #蓝色区间
blue_lower=np.array([100, 43, 46])
```

```
blue_upper = np.array([124, 255, 255])

# #黄色区间
yellow_lower = np.array([26, 43, 46])
yellow_upper = np.array([34, 255, 255])

# #橙色区间
orange_lower = np.array([11, 43, 46])
orange_upper = np.array([25, 255, 255])

#黑色区间
black_lower = np.array([0,0,0])
black_upper = np.array([180,255,46])


color_dict={"red":[red_lower,red_upper],"green":[green_lower,green_upper],
            "blue":[blue_lower,blue_upper],"yellow":[yellow_lower,yellow_upper],
            "black":[black_lower,black_upper]}

choice_str='''请输入你想识别的颜色
red
blue
yellow
green
black
'''


def init():
    GPIO.setup(servoPin1, GPIO.OUT)
    GPIO.setup(servoPin2, GPIO.OUT)


# 根据舵机脉冲控制范围为500-2500usec内:
def servo_pulse(servo1, servo2):
    '''
    servo1--lower servo
    servo2--higher servo
    '''
    init()
    if servo1 < 500:
        servo1 = 500
    elif servo1 > 2500:
        servo1 = 2500
    if servo2 < 500:
        servo2 = 500
    elif servo2 > 2500:
        servo2 = 2500
    pulsewidth = servo1
    GPIO.output(servoPin1, GPIO.HIGH)
    time.sleep(pulsewidth / 1000000.0)
    GPIO.output(servoPin1, GPIO.LOW)
    time.sleep(20.0 / 1000 - pulsewidth / 1000000.0)
```

```
    pulsewidth = servo2
    GPIO.output(servoPin2, GPIO.HIGH)
    time.sleep(pulsewidth / 1000000.0)
    GPIO.output(servoPin2, GPIO.LOW)
    time.sleep(20.0 / 1000 - pulsewidth / 1000000.0)


def color_track():
    global color_lower, color_upper
    global target_valuex, target_valuey

    times = 0
    choice=input(choice_str)
    color_lower,color_upper=color_dict[choice]

    cv.namedWindow("tracking")
    while True:
        ret, frame = cap.read()
        assert ret,print("摄像头开启失败 in while")

        center_x=frame.shape[1]//2
        center_y=frame.shape[0]//2
        frame = cv.GaussianBlur(frame, (5, 5), 0)
        hsv = cv.cvtColor(frame, cv.COLOR_BGR2HSV)
        mask = cv.inRange(hsv, color_lower, color_upper)
        mask = cv.erode(mask, None, iterations=2)
        mask = cv.dilate(mask, None, iterations=2)
        mask = cv.GaussianBlur(mask, (3, 3), 0)
        cnts = cv.findContours(mask.copy(), cv.RETR_EXTERNAL,
cv.CHAIN_APPROX_SIMPLE)[1]

        if len(cnts) > 0:
            cnt = max(cnts, key=cv.contourArea)
            (color_x, color_y), color_radius = cv.minEnclosingCircle(cnt)   #最小
包围圆
            if color_radius > 10:
                times += 1
                cv.circle(frame, (int(color_x), int(color_y)), int(color_radius),
(0, 255, 0), 2)

                xservo_pid.SystemOutput = color_x
                xservo_pid.SetStepSignal(center_x)
                xservo_pid.SetInertiaTime(0.01, 0.01)   #一阶惯性系统
                target_valuex = int(1400 + xservo_pid.SystemOutput)

                yservo_pid.SystemOutput = color_y
                yservo_pid.SetStepSignal(center_y)
                yservo_pid.SetInertiaTime(0.01, 0.01)
                target_valuey = int(1500 + yservo_pid.SystemOutput)

                # 将云台转动至PID调校位置
                time.sleep(0.008)
                if times == 2:
                    times = 0
```

```
                    servo_pulse(target_valuex,target_valuey)

        cv.imshow("tracking",frame)
        if cv.waitKey(33)==ord("q"):
            break

try:
    GPIO.setmode(GPIO.BCM)
    GPIO.setwarnings(False)
    cap = cv.VideoCapture(0)

    color_track()
    cap.release()
    cv.destroyAllWindows()

except:
    print("摄像头开启失败 in except")
```