Color recognition

This part is mainly for the preparation of the following functional gameplay. Main steps:

- Convert the RGB image to be detected into an HSV image
- Define an object of Mat type: mask
- Define the upper and lower limits of the color

The upper limit is a Scalar object, containing three values: hmin, smin, vmin, indicating the minimum value of the three elements of hsv;

The lower limit is also a Scalar object, containing three values: hmax, smax, vmax, indicating the maximum value of the three elements of hsv.

- Use the inRange function to detect whether each pixel of the src image is between lowerb and upperb

If so, the pixel is set to 255 and saved in the mask image, otherwise it is 0.

1. Basic principles

The commonly used models in digital image processing are RGB (red, green, blue) model and HSV (hue, saturation, brightness). RGB is widely used in color monitors and color video cameras. Our usual pictures are generally RGB models. The HSV model is more in line with the way people describe and interpret colors. The color description of HSV is natural and very intuitive to people. Another reason for choosing the HSV model is that the RGB channel cannot reflect the specific color information of the object very well. Compared with the RGB space, the HSV space can very intuitively express the brightness, hue, and brightness of the color, which is convenient for color comparison.

2. HSV model

HSV (Hue, Saturation, Value) is a color space created by A. R. Smith in 1978 based on the intuitive characteristics of color, also known as the Hexcone Model. The color parameters in this model are: hue (H), saturation (S), and brightness (V).

H: 0 — 180

S: 0 — 255

V: 0 — 255

- HSV parameter table:

	Black	**Gray**	**White**	**Red**	**Orange**	**Yellow**	**Green**	**Cyan**	**Blue**	**Purple**	
H_min	0	0	0	0	156	11	26	35	78	100	125
H_max	180	180	180	10	180	25	34	77	99	124	155
S_min	0	0	0	43	43	43	43	43	43	43	
S_max	255	43	30	255	255	255	255	255	255	255	
V_min	0	0	0	46	46	46	46	46	46	46	
V_max	46	220	255	255	255	255	255	255	255	255	

3. Main code

Code path:

```
~/home/jetson/dofbot_pro/dofbot_basic_visual/scripts/03.Color_Recognition.ipynb
```

```
#!/usr/bin/env python
# coding: utf-8
import cv2 as cv
import threading
from time import sleep
from dofbot_utils.dofbot_config import *
import ipywidgets as widgets
from IPython.display import display
from dofbot_utils.fps import FPS
from dofbot_utils.robot_controller import Robot_Controller
```

```
robot = Robot_Controller()
robot.move_look_map()
fps = FPS()
```

```
model = 'General'
color_hsv  = {"red"   : ((0, 43, 46), (10, 255, 255)),
              "green" : ((35, 43, 46), (77, 255, 255)),
              "blue"  : ((100, 43, 46), (124, 255, 255)),
              "yellow": ((26, 43, 46), (34, 255, 255))}
# HSV参数路径  HSV Parameter path
HSV_path="/home/jetson/dofbot_pro/dofbot_color_identify/scripts/HSV_config.txt"
# 读取HSV配置文件,更新HSV值  Read HSV configuration file and update HSV value
try: read_HSV(HSV_path,color_hsv)
except Exception: print("Read HSV_config Error!!!")
```

```
# 创建控件布局  Create widget layout
button_layout  = widgets.Layout(width='200px', height='70px',
align_self='center')
# 输出打印  Output printing
output = widgets.Output()
# 退出按钮  exit button
exit_button = widgets.Button(description='Exit', button_style='danger',
layout=button_layout)
# 图像控件  Image widget
imgbox = widgets.Image(format='jpg', height=480, width=640,
layout=widgets.Layout(align_self='center'))
# 垂直放置  Vertical placement
controls_box = widgets.VBox([imgbox, exit_button],
layout=widgets.Layout(align_self='center'))
# ['auto', 'flex-start', 'flex-end', 'center', 'baseline', 'stretch', 'inherit',
'initial', 'unset']
```

```
def exit_button_Callback(value):
    global model
    model = 'Exit'
#     with output: print(model)
exit_button.on_click(exit_button_Callback)
```

```
# 获取颜色块
def find_color(image, color_name, hsv_lu):
    (lowerb, upperb) = hsv_lu
    # Set identification area
    # 设置识别区域
    point_Xmin=80
    point_Xmax=560
    point_Ymin=20
    point_Ymax=460
    # draw a rectangle
    # 画矩形框
    # cv.rectangle(image, (point_Xmin, point_Ymin), (point_Xmax, point_Ymax),
(105,105,105), 2)
    # Copy the original image to avoid interference during processing
    # 复制原始图像,避免处理过程中干扰
    img = image.copy()
    mask = img[point_Ymin:point_Ymax, point_Xmin:point_Xmax]
    # mask = image.copy()
    # Convert image to HSV
    # 将图像转换为HSV
    HSV_img = cv.cvtColor(mask, cv.COLOR_BGR2HSV)
    # filter out elements between two arrays
    # 筛选出位于两个数组之间的元素
    img = cv.inRange(HSV_img, lowerb, upperb)
    # Set the non-mask detection part to be all black
    # 设置非掩码检测部分全为黑色
    mask[img == 0] = [0, 0, 0]
    # Get structuring elements of different shapes
    # 获取不同形状的结构元素
    kernel = cv.getStructuringElement(cv.MORPH_RECT, (5, 5))
    # morphological closure
    # 形态学闭操作
    dst_img = cv.morphologyEx(mask, cv.MORPH_CLOSE, kernel)
    # Convert image to grayscale
    # 将图像转为灰度图
    dst_img = cv.cvtColor(dst_img, cv.COLOR_RGB2GRAY)
    # Image Binarization Operation
    # 图像二值化操作
    ret, binary = cv.threshold(dst_img, 10, 255, cv.THRESH_BINARY)
    # Get the set of contour points (coordinates)
    # 获取轮廓点集(坐标)
    find_contours = cv.findContours(binary, cv.RETR_EXTERNAL,
cv.CHAIN_APPROX_SIMPLE)
    if len(find_contours) == 3: contours = find_contours[1]
    else: contours = find_contours[0]
    for i, cnt in enumerate(contours):
        x, y, w, h = cv.boundingRect(cnt)
        # Calculate the area of the contour
```

```
        # 计算轮廓的面积
        area = cv.contourArea(cnt)
        if area > 1000:
            # Central coordinates
            # 中心坐标
            x_w_ = float(x + w / 2)
            y_h_ = float(y + h / 2)
            cv.rectangle(image, (x + point_Xmin, y + point_Ymin), (x + w +
point_Xmin, y + h + point_Ymin), (0, 255, 0), 2)
            # drawing center
            # 绘制中心
            cv.circle(image, (int(x_w_ + point_Xmin), int(y_h_ + point_Ymin)), 5,
(0, 0, 255), -1)
            cv.putText(image, color_name, (int(x + point_Xmin-15), int(y +
point_Ymin-15)), cv.FONT_HERSHEY_SIMPLEX, 1, (255, 0, 255), 2)
    return image
```

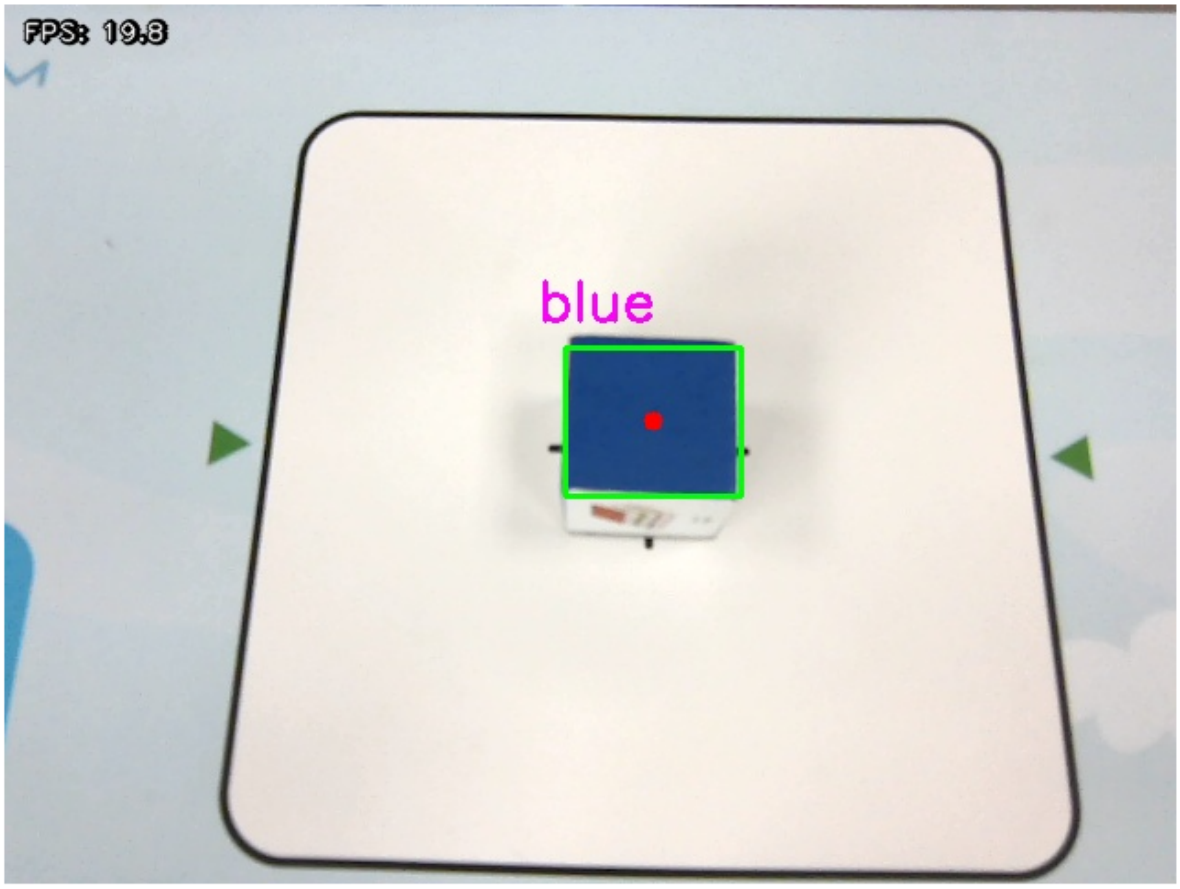
主进程:识别红绿蓝黄颜色。

```
def camera():
    # 打开摄像头 Open camera
    capture = cv.VideoCapture(0)
    capture.set(cv.CAP_PROP_FRAME_WIDTH, 640)
    capture.set(cv.CAP_PROP_FRAME_HEIGHT, 480)
    while capture.isOpened():
        try:
            _, img = capture.read()
            fps.update_fps()
            for key, value in color_hsv.items():
                find_color(img, key, value)
            if model == 'Exit':
                capture.release()
                break
            fps.show_fps(img)
            imgbox.value = cv.imencode('.jpg', img)[1].tobytes()
        except Exception as e:
            capture.release()
            print(e)
            break
```

```
# Start the program
display(controls_box,output)
threading.Thread(target=camera, ).start()
```

Program Click the Run the entire program button on the jupyterlab toolbar, then scroll to the bottom to see the camera component display.

Put the colored side of the building block facing up, and you can mark the color name on the corresponding color building block.



Exit

If you need to exit the program, please click the [Exit] button.

If the color recognition is not accurate, please try the color calibration function first and then try again after the calibration is completed.