Color Recognition

For Orin board users, directly open a web page and enter the IP address:8888 to access jupyter-lab and run directly. For Jetson-Nano board users, you need to first enter the docker container, then enter the following command in docker:

```
cd
jupyter-lab --allow-root
```

Then open a web page and enter the IP address:9999 to access jupyter-lab and run the following program.

This part is mainly to prepare for future functional operations. Main steps:

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

  The upper limit is a Scalar object containing three values: hmin, smin, vmin, representing the minimum values of the three HSV elements;
  The lower limit is also a Scalar object containing three values: hmax, smax, vmax, representing the maximum values of the three HSV elements.
- Use the inRange function to detect whether each pixel of the src image is between lowerb and upperb

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

1. Basic Principles

Commonly used models in digital image processing are the RGB (Red, Green, Blue) model and the HSV (Hue, Saturation, Value) model. RGB is widely used in color monitors and color video cameras, and our usual images are generally RGB models. The HSV model is more in line with the way humans describe and explain colors. HSV color description is natural and very intuitive for humans. Another reason for choosing the HSV model is that RGB channels cannot well reflect the specific color information of objects. Compared to RGB space, HSV space can very intuitively express the brightness, hue, and vividness of colors, making it 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 colors, also known as the Hexcone Model. The parameters of colors in this model are: Hue (H), Saturation (S), and Value (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:

```
#Jetson-Nano users need to enter the docker container to view
~/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))}
# Orin主板HSV参数路径  HSV Parameter path
HSV_path="/home/jetson/dofbot_pro/dofbot_color_identify/scripts/HSV_config.txt"
# Jetson-Nano主板HSV参数路径 Jetson-Nano用户需要进入到docker容器中查看 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!!!")
```

```
# Create widget layout
button_layout  = widgets.Layout(width='200px', height='70px',
align_self='center')
# Output printing
output = widgets.Output()
# exit button
exit_button = widgets.Button(description='Exit', button_style='danger',
layout=button_layout)
# Image widget
imgbox = widgets.Image(format='jpg', height=480, width=640,
layout=widgets.Layout(align_self='center'))
# 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)
```

```
# Get color block
def find_color(image, color_name, hsv_lu):
    (lowerb, upperb) = hsv_lu
    # Set identification area
    point_Xmin=80
    point_Xmax=560
    point_Ymin=20
    point_Ymax=460
    # draw a rectangle
    # cv.rectangle(image, (point_Xmin, point_Ymin), (point_Xmax, point_Ymax),
(105,105,105), 2)
    # Copy the original image to avoid interference during processing
    img = image.copy()
    mask = img[point_Ymin:point_Ymax, point_Xmin:point_Xmax]
    # mask = image.copy()
    # Convert image to HSV
    HSV_img = cv.cvtColor(mask, cv.COLOR_BGR2HSV)
    # filter out elements between two arrays
    img = cv.inRange(HSV_img, lowerb, upperb)
    # Set the non-mask detection part to be all black
    mask[img == 0] = [0, 0, 0]
    # Get structuring elements of different shapes
    kernel = cv.getStructuringElement(cv.MORPH_RECT, (5, 5))
    # morphological closure
    dst_img = cv.morphologyEx(mask, cv.MORPH_CLOSE, kernel)
    # Convert image to grayscale
    dst_img = cv.cvtColor(dst_img, cv.COLOR_RGB2GRAY)
    # Image Binarization Operation
    ret, binary = cv.threshold(dst_img, 10, 255, cv.THRESH_BINARY)
    # Get the set of contour points (coordinates)
    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
        area = cv.contourArea(cnt)
        if area > 1000:
            # Central coordinates
            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
            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
```

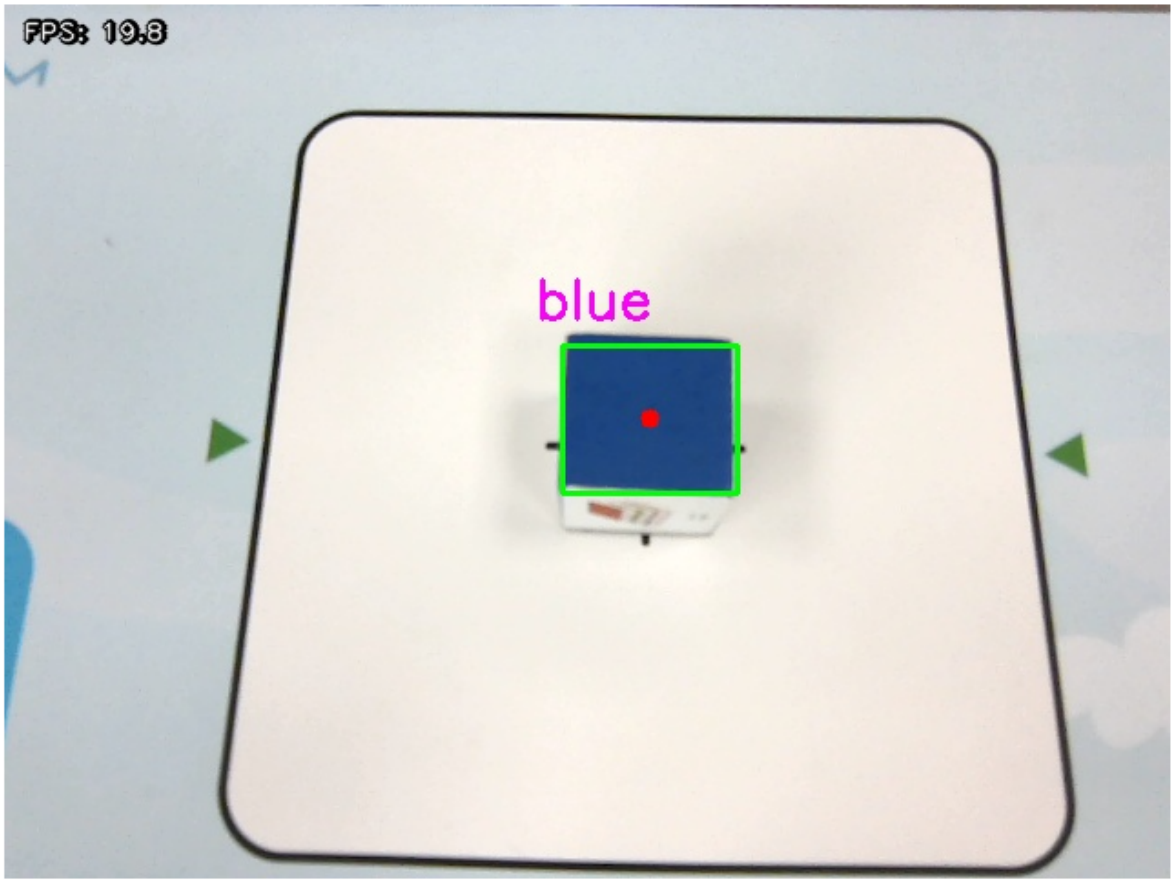
Main process: Recognize red, green, blue, and yellow colors.

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

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

Place the colored side of the building blocks facing up, and the color names will be marked on the corresponding colored blocks.



Exit

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

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