**3.Color recognition**



This part is mainly to prepare for the subsequent functional gameplay. Main steps:

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

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

The lower limit is also a Scalar object, containing three values: hmax, smax, vmax, which represents 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 stored in the mask image, otherwise it is 0.

1. **Basic principles**

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. HSV's color description is natural and very intuitive to people. Another reason for choosing to use the HSV model is that the RGB channel cannot well reflect the specific color information of the object.Compared with RGB space, HSV space can express the lightness, shade, hue, and vividness of colors very intuitively, making it easy to compare colors.

1. **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 parameters of color in this model are: hue (H), saturation (S), and lightness (V).

H: 0 — 180

S: 0 — 255

V: 0 — 255

HSV parameter list:



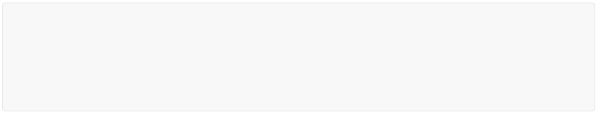
|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **黑** | **灰** | **白** | **红** | **橙** | **黄** | **绿** | **青** | **蓝** | **紫** |  |
| H\_min | 0 | 0 | 0 | 0 | 156 | 11 | 26 | 35 | 78 | 100 | 125 |
| H\_max | 180 | 180 | 180 | 10 | 180 | 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 |  |

1. **Main code**

Code path：/home/dofbot/Dofbot/6.AI\_Visuall/5.Color recognition.ipynb

The following code content needs to be executed according to the actual step. It cannot be run all at once. Running the last unit will directly exit the thread.

#bgr8 to jpeg format



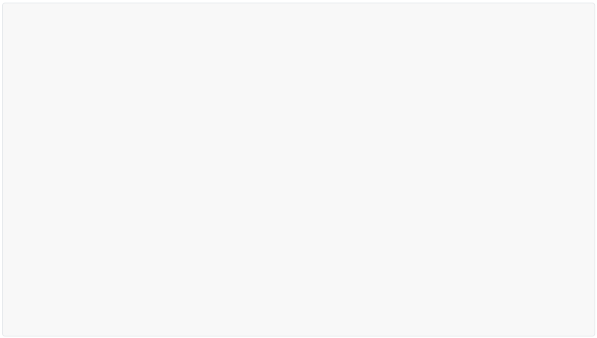
import enum

import cv2

def bgr8\_to\_jpeg(value, quality=75):

return bytes(cv2.imencode('.jpg', value)[1])

#Camera component display



import traitlets

import ipywidgets.widgets as widgets import time

* Thread function operation library import threading

import inspect

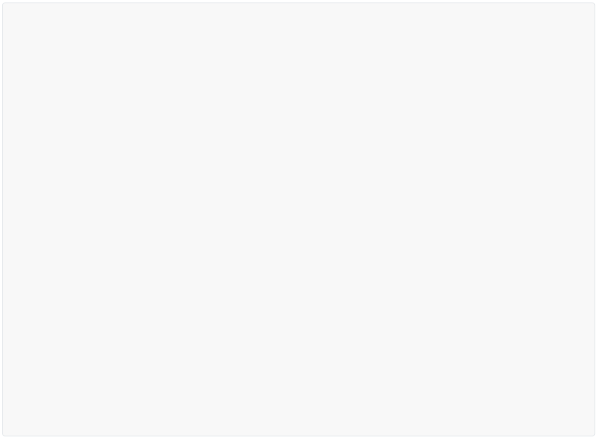
import ctypes

origin\_widget = widgets.Image(format='jpeg', width=320, height=240) mask\_widget = widgets.Image(format='jpeg',width=320, height=240) result\_widget = widgets.Image(format='jpeg',width=320, height=240)

* Create a horizontal box container to place image widgets next to each other image\_container = widgets.HBox([origin\_widget, mask\_widget, result\_widget])
* image\_container = widgets.Image(format='jpeg', width=600, height=500) display(image\_container)

Get hsv value of color

def get\_color(img):



H = []

color\_name={}

img = cv2.resize(img, (640, 480), )

* Convert color image to HSV

HSV = cv2.cvtColor(img, cv2.COLOR\_BGR2HSV)

* Draw a rectangular frame

cv2.rectangle(img, (280, 180), (360, 260), (0, 255, 0), 2)

* Take out the H, S, and V values of each row and column in turn and put them

into the container.

for i in range(280, 360):

for j in range(180, 260): H.append(HSV[j, i][0])

* Calculate the maximum and minimum values of H, S, and V respectively.

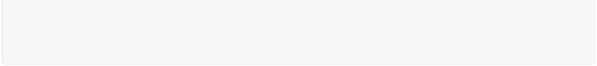
H\_min = min(H);H\_max = max(H)

* print(H\_min,H\_max)
  + Judge color

if H\_min >= 0 and H\_max <= 10 or H\_min >= 156 and H\_max <= 180: color\_name['name'] = 'red'

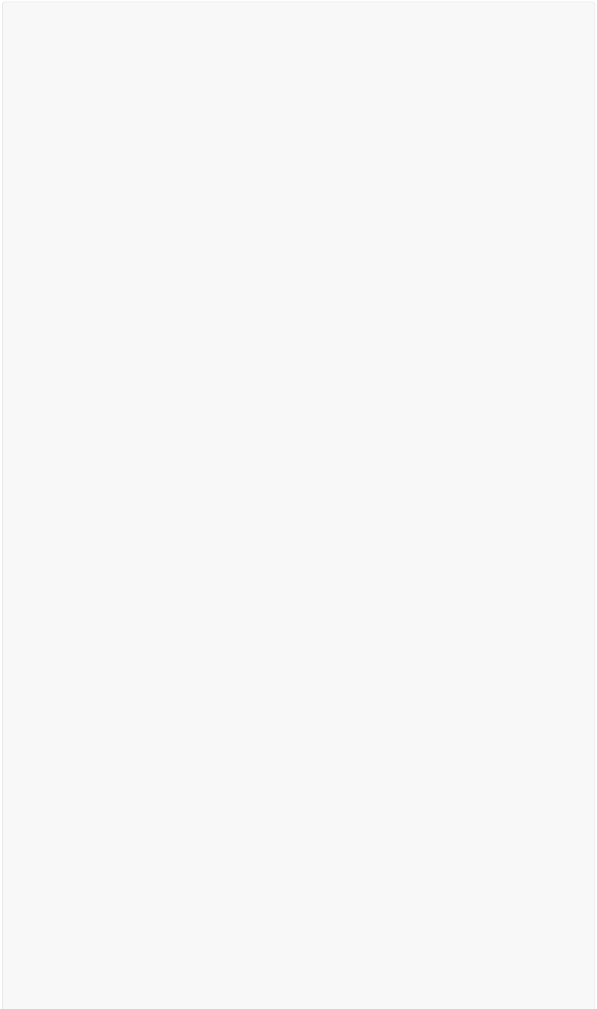
elif H\_min >= 26 and H\_max <= 34: color\_name['name'] = 'yellow' elif H\_min >= 35 and H\_max <= 78: color\_name['name'] = 'green' elif H\_min >= 100 and H\_max <= 124: color\_name['name'] = 'blue'

return img, color\_name



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

import cv2



import numpy as np

import ipywidgets.widgets as widgets

cap = cv2.VideoCapture(0)

cap.set(3, 640)

cap.set(4, 480)

cap.set(5, 30) #Set frame rate

cap.set(cv2.CAP\_PROP\_FOURCC, cv2.VideoWriter.fourcc('M', 'J', 'P', 'G'))

* Red is selected by default, and the program will automatically switch colors based on the color detected in the box.
* red interval

color\_lower = np.array([0, 43, 46])

color\_upper = np.array([10, 255, 255])

def Color\_Recongnize():

while(1):

* get a frame and show Obtain video frames and convert them into HSV

format. Use cvtColor() to convert BGR format into HSV format. The parameter is cv2.COLOR\_BGR2HSV.

ret, frame = cap.read()

frame, color\_name = get\_color(frame)

if len(color\_name)==1:

global color\_lower

global color\_upper

if color\_name['name'] == 'yellow':

color\_lower = np.array([26, 43, 46]) color\_upper = np.array([34, 255, 255])

elif color\_name['name'] == 'red':

color\_lower = np.array([0, 43, 46])

color\_upper = np.array([10, 255, 255])

elif color\_name['name'] == 'green':

color\_lower = np.array([35, 43, 46]) color\_upper = np.array([77, 255, 255])

elif color\_name['name'] == 'blue':

color\_lower=np.array([100, 43, 46])

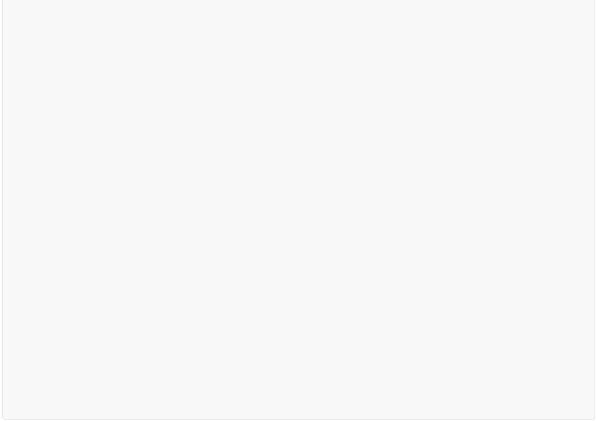
color\_upper = np.array([124, 255, 255])

origin\_widget.value = bgr8\_to\_jpeg(frame) #cv2.imshow('Capture', frame)

* change to hsv model

hsv = cv2.cvtColor(frame, cv2.COLOR\_BGR2HSV)

* get mask Use the inRange() function and the upper and lower bounds of



the blue range in the HSV model to obtain the mask. The blue part of the original video in the mask will be made white and the other parts black.

mask = cv2.inRange(hsv, color\_lower, color\_upper)

#cv2.imshow('Mask', mask)

mask\_widget.value = bgr8\_to\_jpeg(mask)

* detect blue Perform a bitwise AND operation on the mask on the

original video frame, and the white in the mask will be replaced with the real image:

res = cv2.bitwise\_and(frame, frame, mask=mask)

#cv2.imshow('Result', res)

result\_widget.value = bgr8\_to\_jpeg(res)

time.sleep(0.01)

cap.release()

#cv2.destroyAllWindows()

After the program block is run, you will see the camera component display.

