


# Lesson 1 Color Recognition with OpenCV

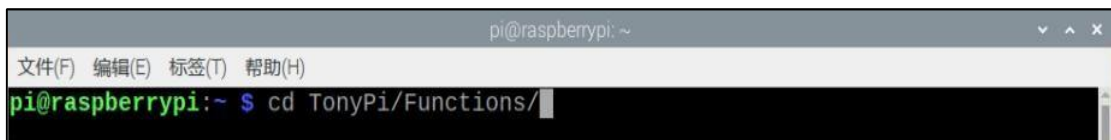
Calling an image through OpenCV is only the first step in image analysis. The position of object also needs to be determined, and then outline its contour and the key to distinguishing the object from the background is the color.

## 1. Operation Steps

1) Turn on TonyPi Pro and connect to VNC.

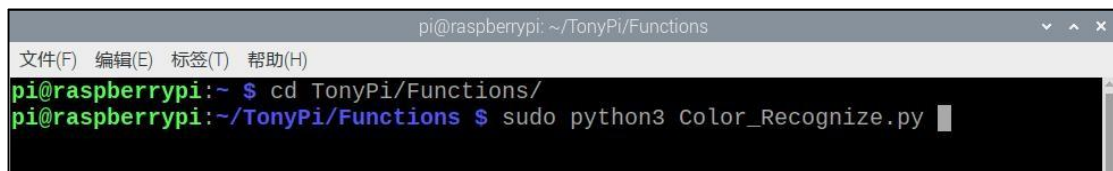
2) Press “Ctrl+Alt+T” or click  icon in the upper left corner to open LX terminal.

3) Enter “cd TonyPi/Functions/” command, and then press “Enter” to come to the category of games programmings.



```
pi@raspberrypi: ~  
文件(F) 编辑(E) 标签(T) 帮助(H)  
pi@raspberrypi:~ $ cd TonyPi/Functions/
```

4) Enter “sudo python3 Color\_Recognize.py” command, and then press “Enter” to start game.

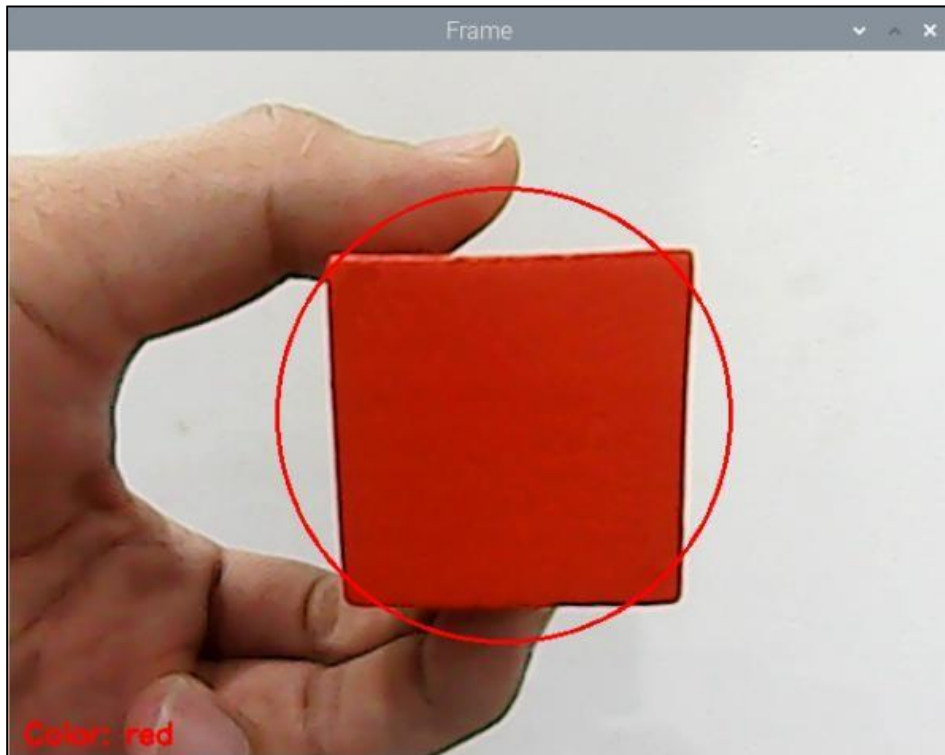


```
pi@raspberrypi: ~/TonyPi/Functions  
文件(F) 编辑(E) 标签(T) 帮助(H)  
pi@raspberrypi:~ $ cd TonyPi/Functions/  
pi@raspberrypi:~/TonyPi/Functions $ sudo python3 Color_Recognize.py
```

5) If you want to exit the game programming, press “Ctrl+C” in the LX terminal interface. If the exit fails, please try it few more times.

## 2. Project Outcome

After starting the game, place a red object in front of camera. Then the object will be circled in the returned image after TonyPi Pro recognizes it and “color: red” is printed.



### 3. Project Analysis

- ◆ Retrieve the image from camera

Through what we learned before, we have mastered how to use OpenCV to process the real-time image from camera. The specific program is as follows:

```
if __name__ == '__main__':  
    from CameraCalibration.CalibrationConfig import *  
    #Load parameter  
    param_data = np.load(calibration_param_path + '.npz')  
    #Obtain parameter  
    mtx = param_data['mtx_array'] dist =  
    param_data['dist_array']  
    newcameramtx, roi = cv2.getOptimalNewCameraMatrix(mtx, dist, (640, 480), 0, (640,  
480))
```

```

mapx, mapy = cv2.initUndistortRectifyMap(mtx, dist, None, newcameramtx, (640, 480),
5)
my_camera = Camera.Camera()
my_camera.camera_open()
print("Color_Recognize Init")
print("Color_Recognize Start")
while True:
    img = my_camera.frame
    if img is not None:
        frame = img.copy()
        frame = cv2.remap(frame, mapx, mapy, cv2.INTER_LINEAR) #Distortion
correction
        Frame = run(frame)
        cv2.imshow('Frame', Frame) key =
        cv2.waitKey(1)
        if key == 27:
            Break
        else:
            time.sleep(0.01)
    my_camera.camera_close()
    cv2.destroyAllWindows()

```

#### ◆ Image binarization

Through the preliminary processing of the image, the information can be effectively used only after contour extraction and position determination. Therefore, the program needs to distinguish the color of the object and the background. The "inRange" function in OpenCV can represent all the pixels in the image as 0 and 1, and display the pixels with a value of 0 in black, and the pixels with a value of 1 in white. This method is the image binarization process. The processing statement is as follow

```

frame_mask = cv2.inRange(frame_lab,
                           (lab_data['red']['min'][0],
                            lab_data['red']['min'][1],
                            lab_data['red']['min'][2]),
                           (lab_data['red']['max'][0],
                            lab_data['red']['max'][1],
                            lab_data['red']['max'][2]))

```

frame\_lab is the image information.

(lab\_data['red']['min'][0],lab\_data['red']['min'][1],lab\_data['red']['min'][2]) is the lower limit of red color.

(lab\_data['red']['max'][0],lab\_data['red']['max'][1],lab\_data['red']['max'][2]) is the upper limit of red color.

The meaning of this statement is to save the image binaryzation information into "frame\_mask" variables. When the L, A, B value of the pixel is between the upper and lower limits of red, it is represented by 1 and the rest of the pixels are represented by 0.

#### ◆ Erode and Dilate

To reduce interference and make image smoother, erode and dilate the grayscale image obtained after binarization in turn. The operation of erosion followed by dilation is called open operation, which can eliminate fine areas of high brightness and separate objects at slim points; for larger objects, it can smooth their boundaries without significant changes in their areas.

The purpose of the erosion process is to remove burrs from the image edges, using the "erode" function of the OpenCV library. The first parameter of this function, "frame\_mask", is the binarized image, and the second parameter represents the kernel for the operation. The kernel shape set in the following function is a rectangle with a 3X3 matrix size.

```

eroded = cv2.erode(frame_mask, cv2.getStructuringElement(cv2.MORPH_RECT, (3, 3)))

```

The dilation process will expand the edges of the image to fill in the non-target pixels

```

dilated = cv2.dilate(eroded, cv2.getStructuringElement(cv2.MORPH_RECT, (3, 3)))

```

at the edges or inside of the target object. It uses the “dilate” function, the first parameter is the image information after erosion processing, and the second parameter is the same as the erosion function.

#### ◆ Find Contour Position

The purpose of the previous operations to image us to get the location of the target object. Next, the function “findcontour” is used to find the contour position of the target object by distinguishing between black and white areas. The specified program is as follow:

```
contours = cv2.findContours(dilated,cv2.RETR_EXTERNAL,cv2.CHAIN_APPROX_NONE)[-2]
```

Save the contour data in “contours” variable. The obtained contours contains all object contours in this color threshold. To obtain the contour of detected object, we custom a function “getAreaMaxContour” that obtain the contour with the largest area to compare contours. The function is as follow:

```
def getAreaMaxContour(contours):  
    contour_area_temp = 0  
    contour_area_max = 0  
    area_max_contour = None  
    for c in contours: #Loop through all contours  
        contour_area_temp = math.fabs(cv2.contourArea(c))#calculate the contour area  
        if contour_area_temp > contour_area_max:  
            contour_area_max = contour_area_temp  
            if contour_area_temp > 50: #The largest contour is valid only if the area is  
                greater than 50.  
                area_max_contour = c  
    return area_max_contour, contour_area_max #Return to the maximum area
```

Firstly, calculate the current contour area “contour\_area\_temp”, and then compare this value with “contour\_area\_max”.

If “contour\_area\_temp” is greater than “contour\_area\_max”, assign the value to “contour\_area\_max”. To filter interference, the maximum area of contour needs to be greater than 50. When it can not meet the condition, a null value is

returned. Based on this logic, loop through all contours detected by `area_max` to obtain the contour with the largest area in the image.

Then, we can call this function to obtain the data of the largest contour. Among them, “`areaMaxContour`” is the data of the largest contour, and `area_max` is the data of the area of the largest contour.

```
areaMaxContour, area_max = getAreaMaxContour(contours)
```

At last, the exact position information is obtained from the maximum contour data, that is, we use “`minEnclosingCircle`” function to obtain the minimum enclosing circle.

```
((centerX, centerY), radius) = cv2.minEnclosingCircle(areaMaxContour_max)
```

Among them, “`centerX`”, “`centerY`” and “`radius`” are the x-axis coordinate and the y-axis coordinate of the centre of the enclosing circle and the radius respectively.

#### ◆ Mark the Position Recognized

At last, mark the recognized object (take “red” as example). Frame the red object in red circle in the image returned by camera, and print the corresponding color “color: red” in the lower-left corner of image.

Circle object and print color information are implemented by using “`circle`” and “`putText`” functions respectively.

```
cv2.putText(img, "Color:
"+detect_color,(10,img.shape[0]-10),cv2.FONT_HERSHEY_SIMPLEX, 0.65, draw_color, 2)

cv2.circle(img, (centerX, centerY), radius, range_rgb["red"], 2)
```