# Part 1 Color Threshold Adjustment

# Part 2 Color Recognition

## 1. Program Logic

Firstly, the program will **acquire the RGB image of the camera**, **scale the image** and **perform Gaussian filtering on the image** so as to **convert the image color space from RGB into LAB**.

Next, **perform image thresholding**, **corrosion**, **dilation**, etc., to **obtain the maximum contour of the target color.**

Lastly, **display the recognition result on the camera returned image**.

The source code of this program is stored in
**/home/jetauto_ws/src/jetauto_example/scripts/color_detect/color_detect.py**

## 2. Operation Steps

1) Start JetAuto, and then connect to ubuntu desktop through NoMachine.

2) Open command line terminal.

3) Enter command "**systemctl stop start_app_node.service**" to enter the catalog of program.

4) Input command "**roscd jetauto_example/scripts/color_detect/**" and press Enter to enter the program directory.

5) Input command "**python3 color_detect.py**", and press Enter to start the game.

6) If want to close this game, please press "**Ctrl+C**". If the game cannot be closed, please press the key again.

## 3. Program Outcome

After the program starts, put the objects within the camera filed of view. When JetAuto recognizes the target, the target object will be circled in the corresponding color, and color name will be printed at the lower left corner. The program can recognize red, blue and green.

## 4. Program Analysis

```
#!/usr/bin/env python3
# encoding: utf-8
# @data:2022/11/07
# @author:aiden
import cv2
import math
import time
import yaml
import numpy as np
import jetauto_sdk.misc as misc
```

```
# Defines a dictionary range_rgb that contains RGB values for red, green, blue, black, and white.
range_rgb = {
    'red': (0, 0, 255),
    'blue': (255, 0, 0),
    'green': (0, 255, 0),
    'black': (0, 0, 0),
    'white': (255, 255, 255),
}

def get_yaml_data(yaml_file): # Reads data from a YAML-formatted file and returns it
    yaml_file = open(yaml_file, 'r', encoding='utf-8') # Open the specified YAML file
    file_data = yaml_file.read() # Reads the entire contents of the file and assigns the contents to the variable file_data.
    yaml_file.close() # Close opened files to free up resources

    data = yaml.load(file_data, Loader=yaml.FullLoader) # Loading YAML data

    return data

# Calling the get_yaml_data() function reads the data from a YAML file that is used to configure the LAB color space
lab_data = get_yaml_data("/home/jetauto/jetauto_software/lab_tool/lab_config.yaml")


# Finds the contour with the largest area in the given list of contours and returns the contour and its area
def getAreaMaxContour(contours):
    contour_area_temp = 0 # Initialize the temporary contour area variable to 0
    contour_area_max = 0 # Initialize the maximum contour area variable to 0.
    area_max_contour = None

    for c in contours:  # Traverse all contours
        contour_area_temp = math.fabs(cv2.contourArea(c))  # Calculate contour area
        if contour_area_temp > contour_area_max: # If the area of the current contour is greater than the previously recorded maximum contour area:
            contour_area_max = contour_area_temp # Updates the maximum contour area to the area of the current contour.
            if contour_area_temp > 50:  # Filtered out some small areas of the contour to minimize interference.
                area_max_contour = c

    return area_max_contour, contour_area_max  #  Returns the largest contour
```

```
color_list = []
detect_color = 'None'
draw_color = range_rgb["black"]

size = (320, 240)



def run(img):

    # Declares the global variables to be used in the function.
    global draw_color
    global color_list
    global detect_color

    img_copy = img.copy() # Copy the input image
    img_h, img_w = img.shape[:2] # Gets the height and width of the image.

    frame_resize = cv2.resize(img_copy, size, interpolation=cv2.INTER_NEAREST)
# Resize the image to the specified size size, using nearest neighbor
interpolation.
    frame_gb = cv2.GaussianBlur(frame_resize, (3, 3), 3) # Perform Gaussian
blurring on the resized image to reduce noise.
    frame_lab = cv2.cvtColor(frame_gb, cv2.COLOR_BGR2LAB)  # Converting images
from BGR space to LAB space

    # Initialize some variables:
    max_area = 0
    color_area_max = None
    areaMaxContour_max = 0

    for i in ['red', 'green', 'blue']:
        # cv2.inRange()：Used to find pixel values within a specified range in
the input image and generate a binary mask.
        # lab_data['lab']['Stereo'][i]['min'] and
lab_data['lab']['Stereo'][i]['max'] denote, respectively, the minimum and
maximum values for a particular color in the LAB color space.
            # These values are usually read from a configuration file and are
used to specify the color range for each color.
        # tuple(): converts the minimum and maximum values read from the
configuration file to tuples, since the cv2.inRange() function needs to accept
tuples as arguments.
        frame_mask = cv2.inRange(frame_lab,
tuple(lab_data['lab']['Stereo'][i]['min']),
tuple(lab_data['lab']['Stereo'][i]['max']))
        # In summary, this line of code is used to filter the converted image
for a range of pixel values based on the minimum and maximum values of the
specified color in the LAB color space, generating a binary mask where pixel
```

```
values within the specified color range are white (255) and those not within
the range are black (0)


        #The erosion operation is a morphological operation in image processing
that is used to reduce or eliminate pixels at the boundaries of objects in an
image. It is based on the principle that a fixed-size structuring element is
applied to each pixel in an image, setting the value of that pixel to the
smallest of its surrounding pixels. The erosion operation can make object
boundaries smoother and can remove small objects or fine features.
        eroded = cv2.erode(frame_mask, cv2.getStructuringElement(cv2.MORPH_RECT,
(3, 3)))# cv2.getStructuringElement() : is the function used to create the
structuring element
        # cv2.MORPH_RECT is one of the constants used in OpenCV to create
rectangular structure elements
        # The cv2.erode() function is used to perform erosion operations on
binary images.


        # The Dilation operation is a morphological operation in image
processing used to enlarge or expand the boundaries of objects in an image. It
is based on the principle that a fixed-size structural element is applied to
each pixel in an image, setting the value of that pixel to the maximum of the
pixels around it.
        dilated = cv2.dilate(eroded, cv2.getStructuringElement(cv2.MORPH_RECT,
(3, 3)))



        contours = cv2.findContours(dilated, cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_NONE)[-2]
        # The cv2.findContours() function finds contours in the dilated mask.
        # cv2.RETR_EXTERNAL Specifies that only external contours are detected
        # cv2.CHAIN_APPROX_NONE Indicates that all contour points are saved
without compression.


        areaMaxContour, area_max = getAreaMaxContour(contours)
        # Call the getAreaMaxContour() function to find the contour with the
largest area and its area.



        # Updates the maximum area max_area, the color corresponding to the
maximum area color_area_max and the contour corresponding to the maximum area
areaMaxContour_max.
        if areaMaxContour is not None:
            if area_max > max_area:
```

```
                max_area = area_max
                color_area_max = i
                areaMaxContour_max = areaMaxContour


    if max_area > 200:

        # The cv2.minEnclosingCircle() function is used to find the smallest
enclosing circle of the contour, returning the center coordinates and radius.
        ((centerX, centerY), radius) =
cv2.minEnclosingCircle(areaMaxContour_max)

        # Maps the value of the circle center coordinate and radius from the
size of the original image to the size of the scaled image.
        centerX = int(misc.val_map(centerX, 0, size[0], 0, img_w))
        centerY = int(misc.val_map(centerY, 0, size[1], 0, img_h))
        radius = int(misc.val_map(radius, 0, size[0], 0, img_w))



        cv2.circle(img, (centerX, centerY), radius, range_rgb[color_area_max],
2)
        # Draws a circle on an image using the cv2.circle() function.
            # Parameters include the image, center coordinates, radius, color,
and line width.

        # Determine the color value based on the color corresponding to the
largest area and add it to the color list for subsequent color judgments
        if color_area_max == 'red':
            color = 1
        elif color_area_max == 'green':
            color = 2
        elif color_area_max == 'blue':
            color = 3
        else:
            color = 0
        color_list.append(color)

        if len(color_list) == 3:  # Multiple judgments



            # Calculates the average of the color list and determines the
detected color based on the average.
            color = int(round(np.mean(np.array(color_list))))
            color_list = []
            if color == 1:
                detect_color = 'red'
                draw_color = range_rgb["red"]
```

```
            elif color == 2:
                detect_color = 'green'
                draw_color = range_rgb["green"]
            elif color == 3:
                detect_color = 'blue'
                draw_color = range_rgb["blue"]
            else:
                detect_color = 'None'
                draw_color = range_rgb["black"]
    else:
        detect_color = 'None'
        draw_color = range_rgb["black"]

    cv2.putText(img, "Color: " + detect_color, (10, img.shape[0] - 10),
cv2.FONT_HERSHEY_SIMPLEX, 0.65, draw_color, 2)
    """ cv2.putText():
        This is the function used in OpenCV to add text to an image.
    img:
        Indicates the image on which the text is to be added.
    "Color: " + detect_color:
        is the content of the text to be added, in the format of "Color:
detected_color".
    detect_color is a string variable indicating the detected color, such as
"red", "green", "blue" or "None".

    (10, img.shape[0] - 10):
        It is the start position of the text, (10, img.shape[0] - 10) means the
coordinate of the bottom left corner of the text.
        (10, img.shape[0] - 10) indicates the position of the text 10 pixels
from the left edge of the image and 10 pixels from the bottom of the image.
    cv2.FONT_HERSHEY_SIMPLEX:
        Indicates the font type of the text. cv2.FONT_HERSHEY_SIMPLEX is a
simple font style.
    0.65:
        Indicates the scaling factor of the text, which controls the size of
the text.
        Here, the text size is 65% of the original size.
    draw_color:
        Is the color of the text, used to draw the text.
        draw_color is a BGR color tuple that changes according to the detected
color.
    2:
        Is the line width of the text, used to control the thickness of the
text. """

    return img
```

```python
if __name__ == '__main__':

    # Create a VideoCapture object to get frames from a camera or video file.
Here, the camera device path is "/dev/astrapro"
    cap = cv2.VideoCapture("/dev/astrapro")
    # cap = cv2.VideoCapture("/dev/usb_cam")

    # This is an infinite loop for constantly reading frames from the camera
and processing them.
    while True:
        # Use the cap.read() method to read a frame.
        ret, frame = cap.read()
        # ret is a boolean value indicating whether the frame was successfully
read.
        # frame is an array of frame images.

        # If the frame image is successfully read, the following operation is
performed.
        if ret:
            # Processing of the read frame image.
            image = run(frame)

            # Use the cv2.imshow() method to display the processed image. The
first argument is the name of the window and the second argument is the image
to be displayed.
            cv2.imshow('image', image)

            # Use the cv2.waitKey() method to wait for the user to press a key.
            key = cv2.waitKey(1)

            # If the user presses the ESC key (ASCII value 27) on the keyboard,
the loop is exited, i.e., the video playing is exited.
            if key == 27:
                break

        # If the frame image read fails, perform the following.
        else:
            # Wait 0.01 seconds, then try to read the frame again
            time.sleep(0.01)

    # Release the resources occupied by the VideoCapture object and close the
camera or video file.
    cap.release()
    # Destroys all created windows, releasing resources related to image
display.
    cv2.destroyAllWindows()
```

## 4.1 Basic Configuration

### ◆ Read Color Threshold Setting File

Acquire the color threshold configuration file through **get_yaml_data** module.

### ◆ Acquire the Image

Read the returned camera image in real time through VideoCapture module.

## 4.2 Process Image

### ◆ Scale

Adopt resize() function in cv2 library to scale the image so as to reduce the computation.

The first parameter "**img_copy**" is the input image, the second parameter "**size**" is the width and length of the image after scaling, and the third parameter "**interpolation**" refers to the interpolation method.

### ◆ Gaussian filtering

Adopt GaussianBlur() function in cv2 library to perform Gaussian filtering on the image to remove the noise from the image.

The first parameter "**frame_resize**" is the input image

The second parameter "**(3, 3)**" is the size of Gaussian convolution kernel, and its height and width must be positive number and odd number

The third parameter **"3"** is the standard deviation of the Gaussian kernel in the horizontal direction.

### ◆ Image Thresholding

Adopt **inRange()** function in cv2 library to perform image thresholding.

The first parameter "**frame_lab**" is the input image.

The second parameter "**tuple(lab_data['lab'][i]['min'])**" is the minimum value of the color threshold.

The third parameter "**tuple(lab_data['lab'][i]['max'])**" is the maximum value of the color threshold.

When the RGB value of a pixel is within the color threshold range, this pixel will be assigned as "**1**", otherwise "**0**".

### ◆ Corrosion and Dilation

Perform corrosion and dilation on the image to smooth the contour edge of the image for better searching for the target contour.

erode() function is used to execute corrosion. The meaning of the parameters in bracket is as follow.

The first parameter "**frame_mask**" is the input image.

The second parameter “**cv2.getStructuringElement(cv2.MORPH_RECT, (3, 3))**” is the structuring element or kernel deciding the nature of the operation.

And the first parameter in the bracket is the kernel shape and the second parameter is the dimension of the kernel.

dilate() function is used for dilation. The meaning of the parameters in the bracket is the same as that of erode() function.

- **Acquire the Maximum Contour**

Call findContours() function in cv2 library to find the maximum contour of the target color in the image.

The first parameter “**dilated**” is the input image.

The second parameter “**cv2.RETR_EXTERNAL**” is the contour retrieving mode.

The third parameter “**cv2.CHAIN_APPROX_NONE**” is the method of contour approximation.

## 4.3 Action Feedback

- **Circle the Target**

Circle the target on the camera returned image through circle module.

The meaning of the parameters in the bracket are as follow.

The first parameter “**img**” is the input image.

The second parameter “**(centerX, centerY)**” is the circle center.

The third parameter “**radius**” is the radius of the circle

The fourth parameter “**range_rgb[color_area_max]**” represents the color.

The fifth parameter “**2**” is the thickness of the line.

- **Print the Color Name**

Print the color name on the camera returned image with putText module.

The meaning of the parameters in the brackets is as follow.

The first parameter “**img**” is the input image

The second parameter “**"Color: " + detect_color**” is the printed content.

The third parameter “**(10, img.shape[0] - 10)**” is the position where the content is printed on the camera returned image.

The fourth parameter “**cv2.FONT_HERSHEY_SIMPLEX**” is the font type.

The fifth parameter “**0.65**” is the font size

The sixth parameter “**draw_color**” is the font color

The seventh parameter “**2**” is the font weight.

# Part 3 QR Code Creating and Recognition

## 1. QR Code Creating

### 1.1 Program Logic

Firstly, **create an instance of QR code tool**, and **set the specific parameter of the QR code**.

Next, **acquire the user's data and put it into the QR code**.

Lastly, **according to these data, generate a QR code picture and display it on the window**.

The source code of this program is stored in
**/home/jetauto_ws/src/jetauto_example/scripts/qrcode/qrcode_creater.py**

```
#!/usr/bin/env python3
# encoding: utf-8
import os
import cv2
import qrcode
import numpy as np

def create_qrcode(data, file_name):

    # Create an Object of the QR Code Tool
    qr = qrcode.QRCode(
        version=1,
        error_correction=qrcode.constants.ERROR_CORRECT_H,
        box_size=5,
        border=4)

    # Generate QR Code
     # Add Data
    qr.add_data(data)
     # Filling data
    qr.make(fit=True)
     # Generate Image
    img = qr.make_image(fill_color=(0, 0, 0), back_color=(255, 255, 255))

    # Display the Picture of Generated QR Code
    opencv_img = cv2.cvtColor(np.asarray(img), cv2.COLOR_RGB2BGR)
    cv2.imshow('img', opencv_img)
    cv2.waitKey(0)

    # Save the Picture
```

```
    cv2.imwrite(file_name, opencv_img)
    print('save', data, file_name)

if __name__ == '__main__':
    file_path = os.getcwd() # It returns the path to the current working
directory
    out_img = file_path + '/myQRcode.jpg' # This line of code constructs a file
path to save the generated QR code image.
    qrcode_text = input("Please enter：") # This line of code prompts the user
to enter the data to be encoded
    create_qrcode(qrcode_text, out_img) # Transfer the user input data and the
file path of the output image as parameters to the function to generate the QR
code image.
```

## 1.2 Operation Steps

1) Start JetAuto, and then connect to ubuntu desktop through NoMachine.

2) Open command line terminal.

3) Enter command "**systemctl stop start_app_node.service**" to stop auto-start program.

4) Input command "**cd jetauto_ws/src/jetauto_example/scripts/qrcode/**" and press Enter to enter the directory where the program is stored.

5) Input command "**python3 qrcode_creater.py**" to start the game program

6) If you want to close this game, you press "**Ctrl+C**" on the terminal. If the game cannot be closed, you can try again.

## 1.3 Program Outcome

1) After the game starts, you need to input some words on the terminal, for example "**hello**".

2) Then press Enter, and the QR code picture containing the input data will pop up.


3) After you close the window, the saved content and path will be printed.

## 1.4 Program Analysis

◆ **Create an Object of the QR Code Tool**

Create the required object with qrcode module, and set the parameters of the QR code.

```
qr = qrcode.QRCode(
        version=1,
        error_correction=qrcode.constants.ERROR_CORRECT_H,
        box_size=5,
        border=4)
```

The meanings of the parameters in the function are as follow.

The first parameter "**version**" is the integer ranging from 1 to 40, which is used to control the size of the QR code. If you want to let the program decide the size automatically, you can set the value as None and adopt **fit** parameter.

The second parameter "**error_correction**" is used to control the error correction function.

1) **ERROR_CORRECT_L** represents that about 7% or less error will be corrected.

2) **ERROR_CORRECT_M** is the default value which means that 15% or less errors can be corrected.

3) **ROR_CORRECT_H** represents 30% or less errors can be corrected.

The third parameter "**box_size**" is used to control the number of pixel contained in each grid in the OR code.

The fourth parameter "**border**" is used to control the number of grid contained in the border, and the default number is 4 (minimum value).

◆ **Generate QR Code**

Acquire and put the data into the QR code through **add_data** and **make** function, then adopt "make_image" function to generate the picture.

```
# Add Data
qr.add_data(data)
# Filling data
qr.make(fit=True)
# Generate Image
img = qr.make_image(fill_color=(0, 0, 0), back_color=(255, 255, 255))
```

The meaning of **make_image** function is as follow.

The first parameter "**fill_color=(0, 0, 0)**" represents the color the picture is filled in, and these value indicates black.

The second parameter "**back_color=(255, 255, 255)**" is the background color, and the background color of this picture is white.

◆ **Display the Picture of Generated QR Code**

Convert the color space of the picture with cvtColor function, then display the picture on the window through imshow function.

```
opencv_img = cv2.cvtColor(np.asarray(img), cv2.COLOR_RGB2BGR)
cv2.imshow('img', opencv_img)
cv2.waitKey(0)
```

◆ **Save the Picture**

Store the generated QR code picture through imwrite function, and print the related function.

```
cv2.imwrite(file_name, opencv_img)
print('save', data, file_name)
```

The meanings of the parameters in imwrite function is as follow.

The first parameter "**file_name**" is the storage path of the picture.

The second parameter "**opencv_img**" is the picture to be stored.

# 2. QR Code Recognition

## 2.1 Program Logic

Firstly, create an instance of **QR code detection**, and **import the network structure and model weights file required by the detection.**

Next, **acquire the camera's returned image and start detecting.**

Lastly, **after the QR code is recognized**, the **QR code and its content will be printed.**

The source code of this program is stored in
**/home/jetauto_ws/src/jetauto_example/scripts/qrcode/qrcode_detecter.py**

## 2.2 Operation Steps

1) Start JetAuto, and then connect to ubuntu desktop through NoMachine.

2) Open command line terminal.

3) Input command "**cd jetauto_ws/src/jetauto_example/scripts/qrcode/**" and press Enter to enter the directory where the program is stored.

4) Input command "**python3 qrcode_detecter.py**" to start the game program.

5) If you need to close this game, you can press "**Ctrl+C**" on the terminal. If the game cannot be exited, please try again.

## 2.3 Program Outcome

After the program starts, JetAuto will recognize the QR code, then frame the code and print the content.

## 2.4 Program Analysis

◆ **Create QR Code Detection Object**

Create the required object with **wechat_qrcode_WeChatQRCode** module whose WeChat scan engine is based on the open-source ZXing engine. ZXing engine is a highly optimized and deeply remodeled and high-performance lightweight QR code reader.

```
# Create QR Code Detection Object
MODEL_PATH = os.path.join(os.path.split(os.path.realpath(__file__))[0],
'opencv_3rdparty')
model1 = os.path.join(MODEL_PATH, 'detect.prototxt')
model2 = os.path.join(MODEL_PATH, 'detect.caffemodel')
model3 = os.path.join(MODEL_PATH, 'sr.prototxt')
model4 = os.path.join(MODEL_PATH, 'sr.caffemodel')

detect_obj = cv2.wechat_qrcode_WeChatQRCode(model1, model2, model3, model4)
```

The meanings of the parameters in **wechat_qrcode_WeChatQRCode()** function is as follow.

The first parameter "**model1**" is the weights file of the detection model.

The second parameter "**model2**" is the caffe framework model required by the detection.

The third parameter "**model3**" is the weights file of the super-resolution model.

The fourth parameter "**model4**" is the caffe framework model required by super-resolution model.

- **Acquire Camera View**

Acquire the designated camera view with VideoCapture() function.

```
# Acquire Camera View
cap = cv2.VideoCapture("/dev/astrapro")
```

- **Recognize QR Code**

Recognize the QR code within the camera view through detectAndDecode function.

```
# Recognize QR Code
        res, points = detect_obj.detectAndDecode(img)
```

The first returned value "**res**" is the data in the QR code, and the second returned value "**points**" is the position of the QR code.

- **Mark the QR Code**

Mark the QR code within the image through line() function.

```
# Mark the QR Code
            for p in [(0, 1), (1, 2), (2, 3), (3, 0)]:
                start = int(pos[p[0]][0]), int(pos[p[0]][1])
                end = int(pos[p[1]][0]), int(pos[p[1]][1])
                cv2.line(img, start, end, color, thick)
```

The meaning of the parameters in line() function is as follow.

The first parameter "**img**" is the picture to be drawn.

The second parameter "**start**" is the starting point of the drawn line

The third parameter "**end**" is the end of the drawn line.

The fourth parameter “**color**” is the color of the line

The fifth parameter “**thick**” is the thickness of the line.

- **Display the Marked Image**

Display the marked image with imshow() function.

```
# Display the Marked Image
        cv2.imshow('qrcode_detect', result_image)
```

# Part 4 AprilTag Recognition

# Part 5 AR Vision