5.3 Face tracking and Color tracking

1. Target location capture principle

1.1 Face recognition

The most basic task of face recognition is "**Face Detection**." You must first "capture" the face to recognize it in the future when compared to the captured new face. We only provide a method for face detection.

The most common face detection method is to use the "Haar Cascade Classifier".

Here we use a pre-training classifier. In OpenCV, static images and real-time video have similar operations on face detection. In general, video face detection only reads images of each frame from the camera, and then Detection is performed using a static image detection method.
Face detection first requires a classifier:
face_cascade=cv2.CascadeClassifier('123.xml')
123.xml is Haar cascading data, this xml can be obtained from data/haarcascades in the OpenCV3 source code.

The actual face detection is then performed by the function face_cascade.detectMultiScale().

We can't pass each frame of the image captured by the camera directly into .detectMultiScale(). Instead, we should first convert the image to a grayscale image because face detection requires such a color space.
(！ Note: Be sure to enter the correct location for 123.xml correctly.)

1.2 color recognition

The principle of color recognition is to classify and mark the HSV color gamut space of each color in each frame image.
The first thing to do is pass the function:
cv2.cvtColor(frame,cv2.COLOR_BGR2HSV)
The RGB is converted into hsv, and the mask is constructed according to the Threshold. After the morphological processing of the expansion corrosion, the mask and the original image are bitwise and operated. After the color is found, a circle is drawn on the outline of the color for labeling.

2. Target tracking algorithm implementation

2.1 face tracking

First, we need to import the relevant packages and then create the camera instance, motion control variables, PID controller instance, PTZ bus steering control instance and display controls that we need to use.
Code as shown below:

Import related packages to create camera instances

```
from jetbot import Camera
from jetbot import bgr8_to_jpeg
import PID
camera = Camera.instance(width=720, height=720)
```

Create related control variables

```
global face_x, face_y, face_w, face_h
face_x = face_y = face_w = face_h = 0
global target_valuex
target_valuex = 2048
global target_valuey
target_valuey = 2048
```

Create a PID control instance

```
xservo_pid = PID.PositionalPID(1.9, 0.3, 0.35)
yservo_pid = PID.PositionalPID(1.5, 0.2, 0.3)
```

Create a PTZ servo engine instance

```
from servoserial import ServoSerial
servo_device = ServoSerial()
```

Create display Control

```
import traitlets
import ipywidgets.widgets as widgets
from IPython.display import display
face_image = widgets.Image(format='jpeg', width=300, height=300)
display(face_image)
```

We load the "Haar" "Cascade Classifier File "123.xml" for Face Detection:
Code as shown below:

Load "Haar" Cascade Classifier

```
import cv2
face_cascade = cv2.CascadeClassifier('123.xml')
```

Yahboom

Then, we can enter the main process, after face recognition obtains the position of the current face, the PTZ controller will track the face through the PID controller.
Here we use the positional PID algorithm:
Code as shown below:

Main process of PTZ movement

```
while 1:
    frame = camera.value
    frame = cv2.resize(frame, (300, 300))
    gray = cv2.cvtColor(frame,cv2.COLOR_BGR2GRAY)
    faces = face_cascade.detectMultiScale( gray )
    if len(faces)>0:
        (face_x, face_y, face_w, face_h) = faces[0]
        # Mark the detected face
        # cv2.rectangle(frame,(face_x,face_y),(face_x+face_h,face_y+face_w),(0,255,0),2)
        cv2.rectangle(frame,(face_x+10,face_y),(face_x+face_w-10,face_y+face_h+20),(0,255,0),2)

        # Proportion-Integration-Differentiation algorithm
        # Input X-axis direction parameter PID control input
        xservo_pid.SystemOutput = face_x+face_h/2
        xservo_pid.SetStepSignal(150)
        xservo_pid.SetInertiaTime(0.01, 0.006)
        target_valuex = int(2048 + xservo_pid.SystemOutput)
        # Input Y axis direction parameter PID control input
        yservo_pid.SystemOutput = face_y+face_w/2
        yservo_pid.SetStepSignal(150)
        yservo_pid.SetInertiaTime(0.01, 0.006)
        target_valuey = int(2048+yservo_pid.SystemOutput)
        # Rotate the gimbal to the PID adjustment position
        servo_device.Servo_serial_double_control(1, target_valuex, 2, target_valuey)
    # Real-time return of image data for display
    face_image.value = bgr8_to_jpeg(frame)
```

Yahboom

The corresponding complete source code is located at:
/home/jetbot/Notebook/11.Face tracking/Face tracking.ipynb

2.2 Color tracking

In the implementation process of color tracking, except for the principle of target capture based on face tracking, the process of implementing the algorithm on the large target is generally consistent.
The difference is:
We can run the code in any of the following cells to set the color to be captured to the target color:

Set to identify red array data

```
color_lower=np.array([0,43,46])
color_upper = np.array([10, 255, 255])
target_valuex = target_valuey = 2048
servo_device.Servo_serial_double_control(1, 2048, 2, 2048)
```

Set to identify yellow array data

```
color_lower=np.array([26,43,46])
color_upper = np.array([34, 255, 255])
target_valuex = target_valuey = 2048
servo_device.Servo_serial_double_control(1, 2048, 2, 2048)
```

Set to identify blue array data

```
color_lower=np.array([100,43,46])
color_upper = np.array([124, 255, 255])
target_valuex = target_valuey = 2048
servo_device.Servo_serial_double_control(1, 2048, 2, 2048)
```

Set to identify green array data

```
color_lower=np.array([35,43,46])
color_upper = np.array([77, 255, 255])
target_valuex = target_valuey = 2048
servo_device.Servo_serial_double_control(1, 2048, 2, 2048)
```

Set to identify orange array data

```
color_lower=np.array([11,43,46])
color_upper = np.array([25, 255, 255])
target_valuex = target_valuey = 2048
servo_device.Servo_serial_double_control(1, 2048, 2, 2048)
```

In the main process of the PTZ shown in the figure below, if the recognition effect is not enough when the ambient light is sufficient, you can try to modify the program:

Gaussian filtering (5, 5) means that the length and width of the Gaussian matrix are both 5 and the standard deviation is 0.

```
frame_=cv2.GaussianBlur(frame,(5,5),0)
hsv=cv2.cvtColor(frame,cv2.COLOR_BGR2HSV)
mask = cv2.inRange(hsv, color_lower, color_upper)
```

Corrosion operation to remove edge hairs

```
mask = cv2.erode(mask, None, iterations=2)
```

Expansion operation

```
mask = cv2.dilate(mask, None, iterations=2)
```

The parameters in the operation function are optimized.

Main process of PTZ movement ¶

```
while 1:
    frame = camera.value
    frame = cv2.resize(frame, (300, 300))
    frame_=cv2.GaussianBlur(frame,(5,5),0)
    hsv=cv2.cvtColor(frame,cv2.COLOR_BGR2HSV)
    mask=cv2.inRange(hsv,color_lower,color_upper)
    mask=cv2.erode(mask,None,iterations=2)
    mask=cv2.dilate(mask,None,iterations=2)
    mask=cv2.GaussianBlur(mask,(3,3),0)
    cnts=cv2.findContours(mask.copy(),cv2.RETR_EXTERNAL,cv2.CHAIN_APPROX_SIMPLE)[-2]
    if len(cnts)>0:
        cnt = max (cnts,key=cv2.contourArea)
        (color_x,color_y),color_radius=cv2.minEnclosingCircle(cnt)
        if color_radius > 10:
            # Mark the detected color
            cv2.circle(frame,(int(color_x),int(color_y)),int(color_radius),(255,0,255),2)
            #Proportion-Integration-Differentiation
            xservo_pid.SystemOutput = color_x
            xservo_pid.SetStepSignal(150)
            xservo_pid.SetInertiaTime(0.01, 0.006)
            target_valuex = int(2048+xservo_pid.SystemOutput)
            # Input Y axis direction parameter PID control input
            yservo_pid.SystemOutput = color_y
            yservo_pid.SetStepSignal(150)
            yservo_pid.SetInertiaTime(0.01, 0.006)
            target_valuey = int(2048+yservo_pid.SystemOutput)
             # Rotate the gimbal to the PID adjustment position
            servo_device.Servo_serial_double_control( 1, target_valuex, 2, target_valuey)
     # Real-time return of image data for display
    color_image.value = bgr8_to_jpeg(frame)
```

The corresponding complete source code is located at:
/home/jetbot/Notebook/12.Color tracking/Color tracking.ipynb