6.3 Face tracking and color tracking

Note: In the image operating system provided by Yahboom, the APP remote control process is enabled by default, in order to avoid multiple occupations of internal resources, causing some functions to fail to operate normally.

Before you running the code of this course, please follow the following to close the APP remote control process.

If you want to permanently close the function of the APP that starts automatically after booting, execute the following command:

```
sudo systemctl disable jetbotmini_start
```

If you want to permanently open the function of the APP that starts automatically after booting, execute the following command:

```
sudo systemctl enable jetbotmini_start
```

If you do not restart Jetbotmini to restart the APP function, execute the following command:

```
sudo systemctl stop jetbotmini_start
sudo systemctl start jetbotmini_start
```

In Section 4.2, we have already understood the background and basic usage of OpenCV. With the previous foundation, the following things are easy to understand. Let's get started:

6.3.1 Face recognition

The most basic task of face recognition is "face detection". You must first "capture" the face to be able to recognize it when comparing it with the new captured face in the future. We only provide the method of face detection here. Students who are interested can build their own classifier and their own face data set. Perform name recognition. The most common method of face detection is to use the "Haar Cascade Classifier". Target detection using a cascade classifier based on Haar features is an efficient target detection method proposed by Paul Viola and Michael Jones in the paper "Rapid Object Detection using a Boosted Cascade of Simple Features" in 2001. This machine learning method is based on a large number of positive and negative images training cascade function, and then used to detect objects in other images. Here, we will use it for face recognition. Initially, the algorithm requires a large number of positive images (face images) and negative images (images without faces) to train the classifier. Then we need to extract features from it. The good news is that OpenCV has trainers and detectors. If you want to train your own object classifier, such as cars, airplanes, etc., you can use OpenCV to create one. Here we are using a pre-trained classifier. In OpenCV, static images and real-time video have similar operations for face detection. In layman's terms, video face detection just reads each frame of image from the camera, and then Use static image detection methods for detection. Face detection first needs a classifier:

```
face_cascade=cv2.CascadeClassifier('haarcascade_profileface.xml')
```

haarcascade_profileface.xml is Haar cascade data, this xml can be obtained from data/haarcascades in OpenCV3 source code. Next, the actual face detection is performed through face_cascade.detectMultiScale(). We cannot directly pass each frame of image obtained by the camera into .detectMultiScale(), but should first convert the image to grayscale, because face detection requires such a color space.

Next, let's implement face recognition. First, we first import the relevant package, and then create the camera instances and display controls that need to be used.

Import related packages and create camera instances

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

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)
face = widgets.Image(format='jpeg', width=300, height=300)
display(face_image)
display(face)
```

Then we load the "Haar" "cascade classifier file "haarcascade_profileface.xml" for face detection

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

Then we can enter the main process for face recognition to get the current face position.

```
while 1:
    frame = camera.value
    frame = cv2.resize(frame, (300, 300))
    frame_face =frame.copy()
    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)
        frame_face = frame_face[face_y:face_y+face_h,face_x:face_x+face_w]
        frame_face = cv2.resize(frame_face,(300,300))
        face.value = bgr8_to_jpeg(frame_face)

    # Send back image data in real time for display
    face_image.value = bgr8_to_jpeg(frame)
```

The corresponding complete source code is located:

/home/jetson/Notebooks/English/10.Face_recognition/Face_recognition.ipynb

6.3.2 Color recognition

In section 4.2.6 HSV color space and color space conversion (RGB-HSV), we have already understood the HSV color gamut space. The principle of color recognition is to use the HSV color gamut space of different colors in each frame of image. To classify and mark it out, the first thing to do is to pass:

```
cv2.cvtColor(frame,cv2.COLOR_BGR2HSV)
```

Convert RGB to hsv, construct a mask according to the threshold value, and perform bitwise AND operation between the mask and the original image after the morphological process of expansion and corrosion. After the color is found, a circle is drawn on the outline of the color for labeling.

In the implementation of color tracking, in addition to the different principles of target capture in face tracking, as long as the process of achieving a large target in the algorithm is roughly the same, I will only briefly talk about the differences below:

We can run the code in any cell below to set the color to be captured as the target color:

Set to recognize red array data

```
color_lower=np.array([0,43,46])
color_upper = np.array([10, 255, 255])
```

Set to recognize yellow array data

```
color_lower=np.array([26,43,46])
color_upper = np.array([34, 255, 255])
```

Set to recognize blue array data

```
color_lower=np.array([100,43,46])
color_upper = np.array([124, 255, 255])
```

Set to recognize green array data

```
color_lower=np.array([35,43,46])
color_upper = np.array([77, 255, 255])
```

Set to recognize orange array data

```
color_lower=np.array([11,43,46])
color_upper = np.array([25, 255, 255])
```

In the main process of the gimbal shown in the figure below, if the recognition effect is not ideal when the ambient light is sufficient, you can try to modify it manually

```
    # 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)
    # Carry out corrosion operation to remove frizz on the edges
    mask = cv2.erode(mask, None, iterations=2)
    # Perform expansion operations
    mask = cv2.dilate(mask, None, iterations=2)
```

The parameters in the operation function are optimized.

Color recognition process

```
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)
     # Send back image data in real time for display
    color_image.value = bgr8_to_jpeg(frame)
```

The corresponding complete source code is located:

/home/jetson/Notebooks/English/11.Color_recognition/Color_recognition.ipynb