Face tracking

Based on face recognition and face positioning, the following is to combine with the robotic arm to start face tracking.

1. Face recognition

The most basic task of face recognition is "face detection". You must first "capture" a face before you can identify it when comparing it with the captured new face in the future. We only provide face detection methods here. Interested students can build their own classifier and their own face data set. Perform name recognition. The most common way of detecting faces is using a "Haar cascade classifier". Object detection using a cascade classifier based on Haar features is an efficient object 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 trains a cascade function based on a large number of positive and negative images, which is 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 (images of faces) 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 both a trainer and a detector. If you want to train your own classifier for objects such as cars, airplanes, etc., you can create one using OpenCV. Here we use a pre-trained classifier. In OpenCV, static images and real-time videos have similar operations for face detection. Generally speaking, video face detection just reads out the image of each frame from the camera, and then Detect using static image detection methods. Face detection first requires a classifier:

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

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

(Note: Be sure to enter the correct location of 123.xml correctly.)

Code path:/root/dofbot_ws/src/dofbot_face_follow/face tracking.ipynb

2. Code block design

- Import header files

```
import Arm_Lib
import cv2 as cv
import threading
from time import sleep
import ipywidgets as widgets
from IPython.display import display
from face_follow import face_follow
```

- Initialize the robot arm position

```
Arm = Arm_Lib.Arm_Device()
joints_0 = [90, 135, 20, 25, 90, 30]
Arm.Arm_serial_servo_write6_array(joints_0, 1000)
```

- Create an instance and initialize parameters

```
#Create instance
follow = face_follow()
#Initialization mode
model = 'General'
```

- Create controls

```
button_layout = widgets.Layout(width='250px', height='50px', align_self='center')
output = widgets.Output()
# Exit the control
exit_button = widgets.Button(description='Exit', button_style='danger',
layout=button_layout)
#Image control
imgbox = widgets.Image(format='jpg', height=480, width=640,
layout=widgets.Layout(align_self='center'))
# Space layout
controls_box = widgets.VBox([imgbox, exit_button],
layout=widgets.Layout(align_self='center'))
# ['auto', 'flex-start', 'flex-end', 'center', 'baseline', 'stretch', 'inherit',
'initial', 'unset']
```

- control

```
def exit_button_Callback(value):
    global model
    model = 'Exit'
# with output: print(model)
exit_button.on_click(exit_button_Callback)
```

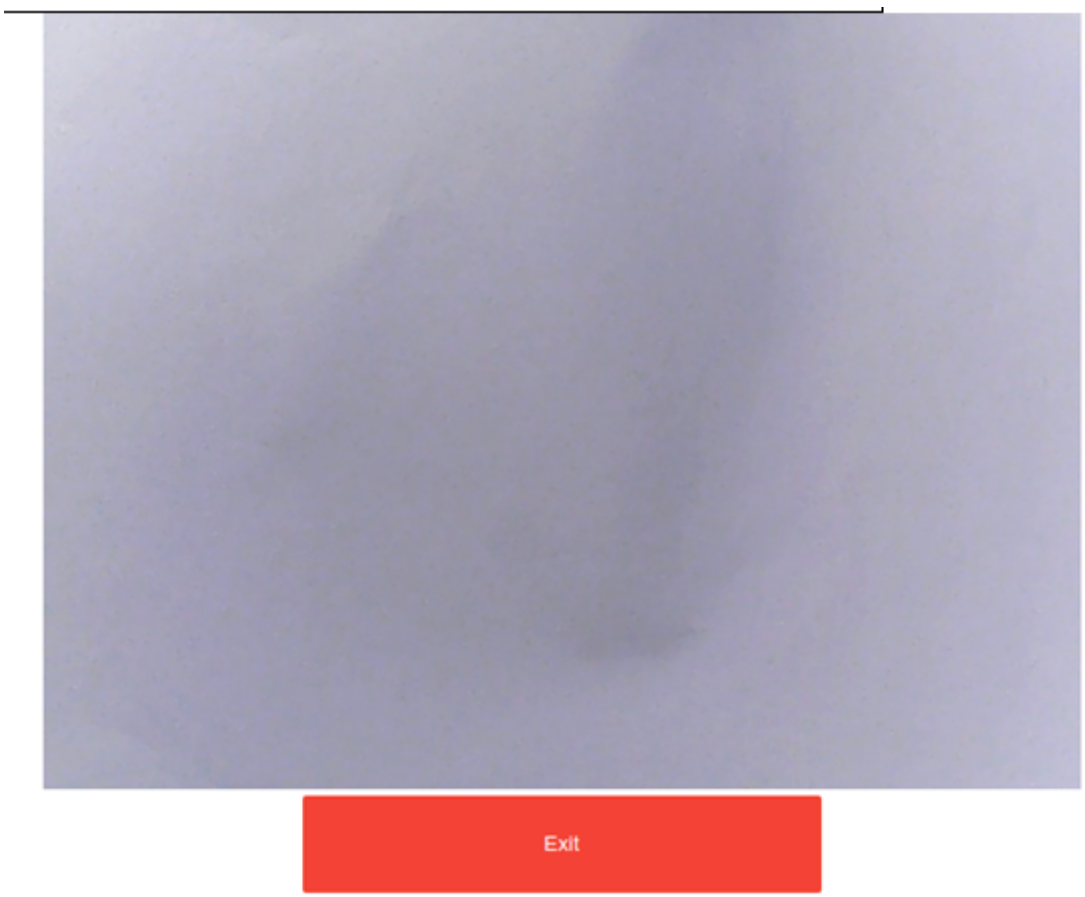
- Main program

```
def camera():
    global model
    #Open camera
    capture = cv.VideoCapture(0)
    # Loop execution when the camera is turned on normally
    while capture.isOpened():
        try:
            # Read each frame of the camera
            _, img = capture.read()
            # Unify image size
            img = cv.resize(img, (640, 480))
            img = follow.follow_function(img)
            if model == 'Exit':
                cv.destroyAllWindows()
                capture.release()
                break
```

```
            imgbox.value = cv.imencode('.jpg', img)[1].tobytes()
        except KeyboardInterrupt:capture.release()
```

- start up

```
display(controls_box,output)
threading.Thread(target=camera, ).start()
```



Then you can see the robotic arm following the human face.

The adjustment of the control algorithm PID is discussed in the PID algorithm section. If you need to adjust, please check the PID algorithm to learn. For detailed face recognition code library, please see

dofbot_ws/src/dofbot_face_follow/face_follow.py