1. OBJECTIVE

In this tutorial, we will use the Python programming language to write facial recognition and QR scanning functions using the OpenCV library. Through this experience, you should master the following skills:

(1) Implement facial recognition and center the face function;

(2) Implement QR Scan function;

(3) Integrate the implemented functions into the program code;

2. COMPONENTS

(1) Windows PC;

(2) A 16GB SD Card;

(3) HDMI Screen, USB keyboard, USB mouse and HDMI Cable;

(4) PIPPY;

3. BASIC KNOWLEDGE OF THE EXPERIMENT

OpenCV (Open-Source Computer Vision Library) is an open-source computer vision and machine learning software library. It includes more than 2500 optimized algorithms, covering various aspects of computer vision and machine learning. These algorithms can be used to detect and recognize faces, identify objects, classify images, perform image analysis, and pattern recognition, and more.

One significant feature of OpenCV is its efficiency and real-time capabilities, which come from its optimized C/C++ implementation. Additionally, OpenCV provides interfaces for multiple languages including Python, Java, and MATLAB, and it can run on various operating systems such as Windows, Linux, Android, and Mac OS.

From a broad application perspective, OpenCV is used in many real-time applications, such as facial

recognition, autonomous vehicles, medical image analysis, motion tracking, and more.

4. EXPERIMENTAL STEPS

Notice:

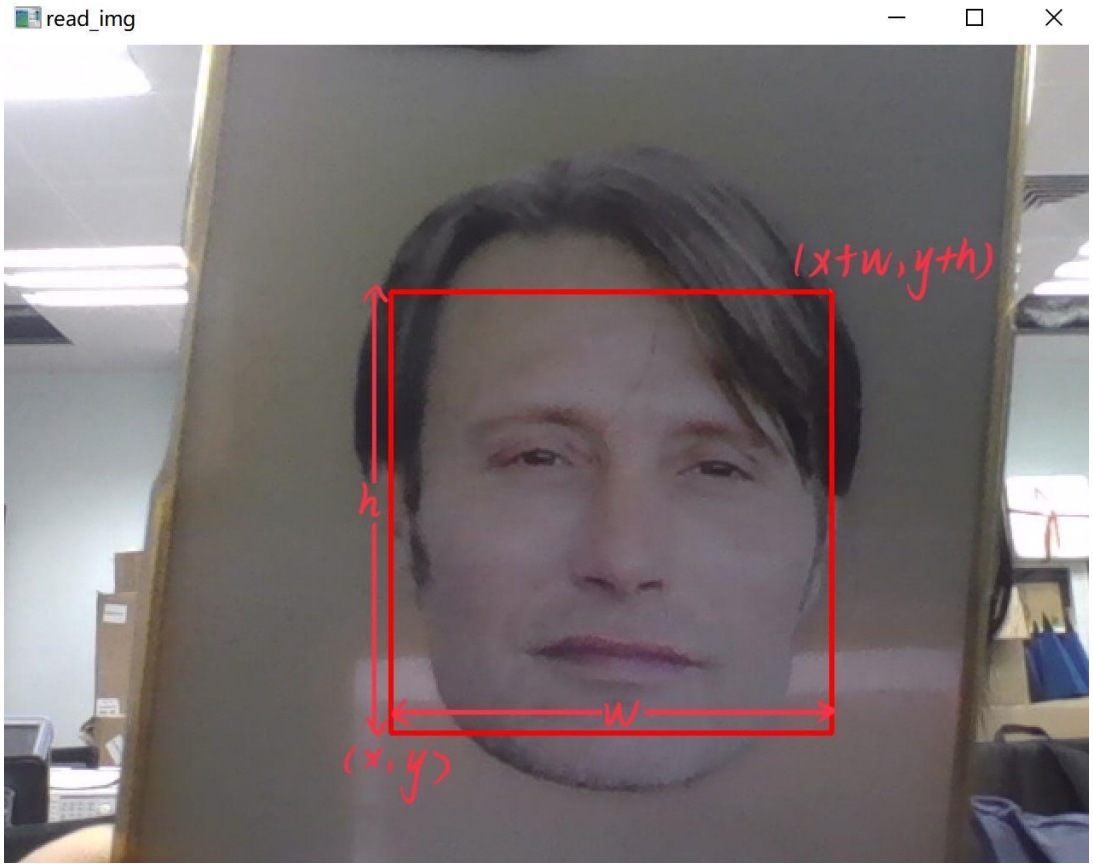
- First, you must clone the PIPPY project from Git Hub. 👉
  https://www.waveshare.com/wiki/PIPPY

4.1.1 Face Detection

1) Install the "numpy" library. You can find the resources and tutorials on the Internet.
2) Create a new Python file. Write the code. Here, `(x,y)` is the coordinate of the start point, which is the lower left corner of the rectangle, and `(x+w, y+h)` is the coordinate of the end point, `w` refers to the width and `h` refers to the height.

```
import os
import cv2

import numpy as np

cap = cv2.VideoCapture(0)

while True:
    ret, frame = cap.read()
    if ret:
        print('Working')
        gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
        face_detect = cv2.CascadeClassifier(cv2.data.haarcascades + 'haarcasc
ade_frontalface_default.xml')
        face = face_detect.detectMultiScale(gray)
        for (x,y,w,h) in face:
           cv2.rectangle(frame,(x,y),(x+w,y+h),color=(0,0,255),thickness=2)
        cv2.imshow('read_img',frame)
        if cv2.waitKey(1) == ord('q'):
            break
```

3) After running the program, you should see the camera turns on and a screen named `read_img` like the one shown below:



Notice:

- To end this process, first click the left mouse to choose the **`read_img`** window, then press "q".

4.1.2 Face Centering

The code of this part is used to let the PIPPY detect whether a human's face is in the center. If not, PIPPY's camera will move. The Python file is named "Face.py".

```
import robot
import datetime
import time
import PIPPY

robotCtrl = PIPPY.PIPPY()
robotCtrl.start()

screenWidth = 640
screenHigh = 480

facecenterX = 1
facecenterY = 1
centerX = 320
centerY = 240
```

```
findfaceError = min(screenWidth, screenHigh) / 5

def track(posX, posY, posW, posH):
    print(posX, posY, posW, posH)
    facecenterX = posX + float(posW/2)
    facecenterY = posY + float(posH/2)
    centerX = float(screenWidth/2)
    centerY = float(screenHigh/2)
    if centerX - findfaceError > facecenterX:
        robotCtrl.moveStart(80, 'no', 'right')
        print('right')
        time.sleep(1)
        robotCtrl.moveStop()
        return 1
    if centerX + findfaceError < facecenterX:
        robotCtrl.moveStart(80, 'no', 'left')
        print('left')
        time.sleep(1)
        robotCtrl.moveStop()
        return 1  # This section of code adjusts the camera left or right to cente
r the face.
    if centerY - findfaceError > facecenterY:
        robotCtrl.moveStart(80, 'no', 'forward')
        print('forward')
        time.sleep(1)
        robotCtrl.moveStop()
        return 1
    if centerY + findfaceError < facecenterY:
        robotCtrl.moveStart(80, 'no', 'backward')
        print('backward')
        time.sleep(1)
        robotCtrl.moveStop()
        return 1 # This section of code adjusts the camera up or down to center th
e face.
    if centerX - findfaceError < facecenterX and centerX + findfaceError > fa
cecenterX:
        if centerY - findfaceError < facecenterY and centerY + findfaceError
 > facecenterY:
            print('stay')
            return 0 #This section of code determines whether the face is centere
d, if it is, movement ceases.
```

Notice:

- Remember that you are trying to **move the camera** instead of the faces:
  - If your face is in the left corner, PIPPY should move left, and vice versa.
  - If your face is at the top of the screen, PIPPY should move backward, and vice versa.

4.2 QR scan

1) Install the "pyzbar" library. You can find the resources and tutorials on the Internet.
2) Write the code. The Python file is named "QR.py"

```
import cv2
import numpy as np
from pyzbar.pyzbar import decode

def decoder(image):
    gray_img = cv2.cvtColor(image,0)
    barcode = decode(gray_img)

    for obj in barcode:
        points = obj.polygon
        (x,y,w,h) = obj.rect
        pts = np.array(points, np.int32)
        pts = pts.reshape((-1, 1, 2))
        cv2.polylines(image, [pts], True, (0, 255, 0), 3)

        barcodeData = obj.data.decode("utf-8")
        if barcodeData:
            return( barcodeData )
```

To further process the scanned QR, just use a variable and assign it with the return value of the decoder function, an example is shown below in 3).

3) The following part is for you to verify the `QR_SCAN` function. Write the following code below the previous code.

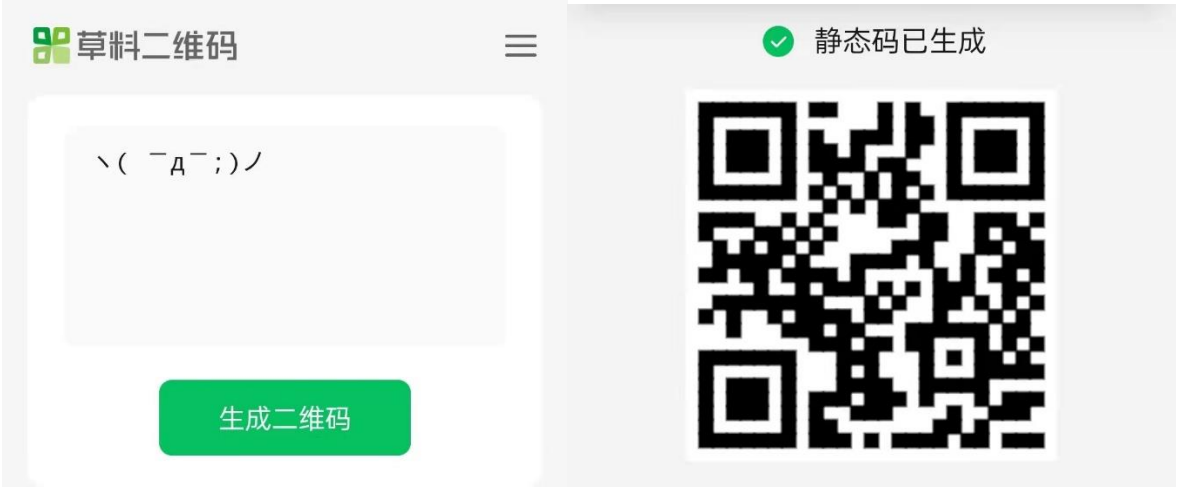
```
if __name__ == "__main__":
    cap = cv2.VideoCapture(0)
    while True:
        ret, frame = cap.read()
        if ret:
            cv2.imshow("Main", frame)
            data = decoder(frame)
```

```
            if data:
                print(data)
        if cv2.waitKey(1) == ord('q'):
            break
```

The whole code structure should be like the figure below:

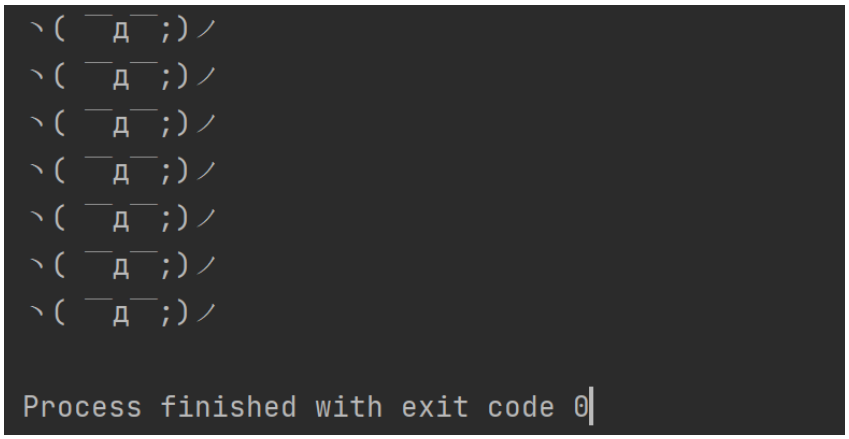
```
import cv2
import numpy as np
from pyzbar.pyzbar import decode

def decoder(image):...


if __name__ == "__main__":
    cap = cv2.VideoCapture(0)
    while True:
        ret, frame = cap.read()
        if ret:
            cv2.imshow("Main", frame)
            data = decoder(frame)
            if data:
                print(data)
        if cv2.waitKey(1) == ord('q'):
            break
```

4) Open the link: https://cli.im/ to generate a QR code. (For example, here I input "ヽ(￣д￣;)ノ" and generate the corresponding QR code.)



5) Run the program again, this time, show the generated QR code to the camera, you should see the

content you input on the terminal screen.

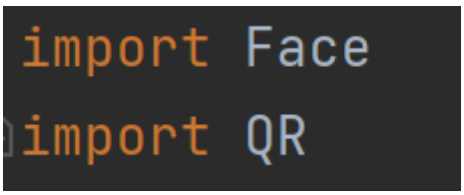


Still, press "q" to end the process.

5. Potential Direction

When you need to run your own code simultaneously with the original PIPPY codes, you are limited since you cannot run two `cv2.VideoCapture(0)` at the same time for only one camera. You can only use the processed image from its original `cv2.VideoCapture(0)` function. The following tutorial shows how to integrate your codes into PIPPY.

1) Import the Python file into `camera_opencv.py`.



2) Add your own function in the `CVThread` class inside the `camera_opencv.py`.

```
def trackFace(self,frame_image):
    gray = cv2.cvtColor(frame_image, cv2.COLOR_BGR2GRAY)
    face_detect = cv2.CascadeClassifier(cv2.data.haarcascades + 'haarcascade
_frontalface_default.xml')
    face = face_detect.detectMultiScale(gray)
    for (x,y,w,h) in face:
       cv2.rectangle(gray,(x,y),(x+w,y+h),color=(0,0,255),thickness=2)
       print('findFace')
       Face.track(x,y,w,h)
    self.pause()
```

```
class CVThread(threading.Thread):
    font = cv2.FONT_HERSHEY_SIMPLEX

    cameraDiagonalW = 64
    cameraDiagonalH = 48
    videoW = 640
    videoH = 480
    Y_lock = 0
    X_lock = 0
    tor = 27
    aspd = 0.005

    def __init__(self, *args, **kwargs):...

    def trackFace(self, frame_image):
        gray = cv2.cvtColor(frame_image, cv2.COLOR_BGR2GRAY)
        face_detect = cv2.CascadeClassifier(cv2.data.haarcascades + 'haarcascade_frontalface_default.xml')
        face = face_detect.detectMultiScale(gray)
        for (x, y, w, h) in face:
            cv2.rectangle(gray, (x, y), (x + w, y + h), color=(0, 0, 255), thickness=2)
            print('findFace')
            Face.track(x, y, w, h)
        self.pause()
```

3) Add your code to the **run** function.

```
elif self.CVMode == 'trackFace':
    self.CVThreading = 1
    self.trackFace(self.imgCV)
    self.CVThreading = 0
```

```
def run(self):
    while 1:
        self.__flag.wait()
        if self.CVMode == 'none':
            robot.robotCtrl.moveStart(speedMove, 'no', 'no')
            self.pause()
            continue

        elif self.CVMode == 'findColor':...

        elif self.CVMode == 'findlineCV':...

        elif self.CVMode == 'watchDog':...

        elif self.CVMode == 'trackFace':
            self.CVThreading = 1
            self.trackFace(self.imgCV)
            self.CVThreading = 0

        pass
```

4) Edit the `commandAct` function

```
elif 'trackFace'== act:
    Camera.modeSelect = 'trackFace'
    Camera.CVMode = 'run'
    pass
```

```
def commandAct(act, inputA):
    global directionCommand, turningCommand, speedMove, posUD
    robot.PIPPY.configPWM(act)
    if act == 'forward':...
    elif act == 'backward':...
    elif act == 'left':...
    elif act == 'right':...
    elif act == 'DS':...
    elif act == 'TS':...

    elif 'wsB' in act:
        speedMove = int(act.split()[1])


    elif 'grab' == act:...

    elif 'trackFace' == act:
        Camera.modeSelect = 'trackFace'
        Camera.CVMode = 'run'
        pass
```

In terms of QR Scan：

```
def findQR(self,frame_image):
    global code
    code = QR.decoder(frame_image)
    if code:
        print(code)
    self.pause()
```

```
class CVThread(threading.Thread):
    font = cv2.FONT_HERSHEY_SIMPLEX

    cameraDiagonalW = 64
    cameraDiagonalH = 48
    videoW = 640
    videoH = 480
    Y_lock = 0
    X_lock = 0
    tor = 27
    aspd = 0.005

    def __init__(self, *args, **kwargs):...

    def trackFace(self, frame_image):...

    def findQR(self, frame_image):
        global code
        code = QR.decoder(frame_image)
        if code:
            print(code)
        self.pause()
```

```
elif self.CVMode == 'findQR':
    self.CVThreading = 1
    self.findQR(self.imgCV)
    self.CVThreading = 0
```

```
def run(self):
    while 1:
        self.__flag.wait()
        if self.CVMode == 'none':
            robot.robotCtrl.moveStart(speedMove, 'no', 'no')
            self.pause()
            continue

        elif self.CVMode == 'findColor':...

        elif self.CVMode == 'findlineCV':...

        elif self.CVMode == 'watchDog':...

        elif self.CVMode == 'trackFace':...

        elif self.CVMode == 'findQR':
            self.CVThreading = 1
            self.findQR(self.imgCV)
            self.CVThreading = 0

        pass
```

```
elif 'QR'== act:
    Camera.modeSelect = 'findQR'
```

```
    Camera.CVMode = 'run'
    pass
```

```
def commandAct(act, inputA):
    global directionCommand, turningCommand, speedMove, posUD
    robot.PIPPY.configPWM(act)
    if act == 'forward':...
    elif act == 'backward':...
    elif act == 'left':...
    elif act == 'right':...
    elif act == 'DS':...
    elif act == 'TS':...

    elif 'wsB' in act:
        speedMove = int(act.split()[1])


    elif 'grab' == act:...

    elif 'trackFace' == act:...

    elif 'QR' == act:
        Camera.modeSelect = 'findQR'
        Camera.CVMode = 'run'
        pass
```

In summary, when you want to add and run your own code in the machine, the following steps are basic processes you need to do:

1) Import the file you are going to use (e.g., a function to realize dancing) into **`camera_opencv.py`**.

2) Complete the functions you want to achieve in the **`CVThread`** class inside.

```
def your_function_name(self,frame_image):          '''
    ##############
    Your code here
    ##############
    self.pause()
```

Notice

- Remember to add **`self.pause()`** at the end when you define a function.
- If you want to use the image captured by the camera, use the **`frame_image`**, and your function imported in 1) should have an argument reserved for **`frame_image`**.

- If you want to use the parameters inside your function, make it a global variable.

3) Add your function declaration in the `run` function.

```
elif self.CVMode == ' your_function_name ':
    self.CVThreading = 1
    self. your_function_name (self.imgCV)
    self.CVThreading = 0
```

4) Edit your content in the `commandAct` function.

```
elif ' Trigger word '== act:
    Camera.modeSelect = ' your_function_name '
    Camera.CVMode = 'run'
    pass
```

Notice:

- If you want to call the function you added, use the Android pad or other communication approaches to send 'Trigger word' (Input anything you want) to the PIPPY's WebSocket, referring to 5.2 from *Tutorial 1*.

6. REFERENCE

[1] Open CV, get started with videos, https://docs.opencv.org/3.4/dd/d43/tutorial_py_video_display.html
[2] PIPPY Waveshare, https://www.waveshare.com/wiki/PIPPY

END OF TUTORIAL