

CS205 C/ C++ Programming – Project

Hand Recognition Based on OpenCV

Name: Wang Yexiang (王烨祥)

SID: 12012529

Name: Tang Xingyu(汤星宇)

SID: 11912217

Name: Tang Linjie(唐麟杰)

SID: 12111204

Part 1 - Analysis

1. OpenCV Introduction

OpenCV is a cross-platform computer vision and machine learning software library distributed under the Apache2.0 license (open source) that runs on Linux, Windows, Android, and Mac OS operating systems. It is lightweight and efficient -- it consists of a series of C functions and a small number of C++ classes. It also provides interfaces to Python, Ruby, MATLAB and other languages and implements many common algorithms in image processing and computer vision.

OpenCV is written in C++ language, it has C++, Python, Java and MATLAB interfaces, and supports Windows, Linux, Android and Mac OS, OpenCV mainly tends to real-time visual applications, and utilizes MMX and SSE instructions when available, Support for C#, Ch, Ruby, and GO is also available.

2. Face recognition and Hand Detection

Face recognition is a kind of biometric identification technology based on facial feature information, and Hand recognition is based on the similar situation. Using a camera or camera to collect images or video streams containing faces, and automatically detect and track faces in the image, and then carry out face recognition on the detected face of a series of related technologies, usually called portrait recognition, face recognition.

Face image acquisition: different face images can be collected through the camera lens, such as static images, dynamic images, different positions, different expressions and other aspects can be very good collection. When the user is in the shooting range of the acquisition device, the acquisition device will automatically search and shoot the user's face image.

Face detection: face detection in practice is mainly used for the pretreatment of face recognition, that is, to accurately calibrate the location and size of the face in the image. Face image contains rich pattern features, such as histogram feature, color feature, template feature, structure feature and Haar feature. Face detection is to pick out the useful information, and use these features to achieve face detection.

.xml files: Different from Databases such as Access, Oracle and SQL Server, DATABASES provide more powerful data storage and analysis capabilities, such as: data indexing, sorting, lookup, correlation consistency, etc., EXTENSIBLE Markup language only stores data. In fact, one of the biggest differences between it and other data representations is that EXTENSIBLE Markup Language is extremely simple, a seemingly trivial advantage that sets it apart.

The simplicity of XML makes it easy to read/write data in any application, which makes XML quickly become a data exchange language (such languages mainly include XML, JSON, etc., commonly used in interface calls, configuration files, data storage, etc.), although different applications also support other data exchange formats. But soon they will all support XML, which means programs can more easily combine information generated on Windows, Mac OS, Linux, and other platforms, and then easily load XML data into programs, analyze it, and output results in XML format.

3. Use Global Variable to Change Mode

Global variables are a programming term derived from the division of variables. Variables are divided into local and global variables, local variables can also be called internal variables. Variables created by an object or function are usually local variables and can only be referenced internally, not by other objects or functions.

In this project, the reason we chose the global variables is that Using global variables takes up more memory (because of their long lifetime), but in today's world of highly configured computers, this should not be a problem unless you are using global variables for large objects, which should be avoided whenever possible. Programs that use global variables run faster (because memory doesn't need to be reallocated), and not much faster either. Namespace pollution of local variables can be avoided without using too many variables. When a global variable has the same name as a local variable, the local variable is used and the global variable is shielded.

So in the every while loop, the first thing is to check and judge the global variables' value, the whole system will jump to the different function modes, and realize the relative functions we want.

Part 2 – Code

1. OpenCV Library and APIs

In this project, the OpenCV Library and APIs used in the demo was listed below:

```

#include <iostream> // for standard I/O
#include <string> // for strings
#include <iomanip> // for controlling float print precision
#include <sstream> // string to number conversion
#include <opencv2/imgproc/imgproc.hpp> // Gaussian Blur
#include <opencv2/core/core.hpp> // Basic OpenCV structures (cv::Mat, Scalar)
#include <opencv2/highgui/highgui.hpp> // OpenCV window I/O
#include <opencv2/imgproc/imgproc_c.h>

#include <opencv2\video\background_segm.hpp>
#include <opencv2\opencv.hpp>
#include "opencv2\objdetect\objdetect.hpp"
#include <math.h>
#include <ctime>
#include <Windows.h>

using namespace cv;
using namespace std;

```

1)Core -- Core component modules

As a Core component, Core must do a lot of things, but also relatively basic. It includes basic data structures, dynamic data structures, plotting functions, array operating-related functions, auxiliary functions and system functions and macros, XML/YML, clustering, and OpenGL interactive operations.

2)Imgproc -- Image processing module

It includes image filtering, geometric image transformation, mixed image transformation, histogram, structure analysis and shape description, motion analysis and target tracking, feature and target detection.

3)Highgui -- Top-level GUI and video I/O

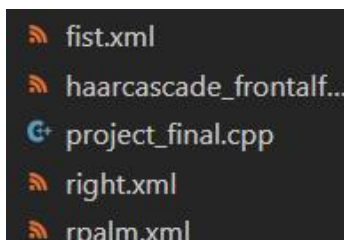
Includes user interface, read/write images and videos, and new QT features.

4)Video -- Video analysis

Including motion analysis and target tracking.

2. Face Recognition and Hand Recognition

First, the .xml files was used in this project to detector the fist, palm, right-direction and face, which was shown in below figures.



```

fist.xml
haarcascade_frontalf...
project_final.cpp
right.xml
rpalm.xml

```

Then .xml file and the corresponding OpenCV interface function are used to load the trained model for gesture recognition and face recognition, indicated below:

```
int main()
{
    //调用摄像头
    VideoCapture cap(0);

    //载入拳头模型
    Fist_Cascade.load("own_project/fist.xml");
    if (Fist_Cascade.empty()) ...

    //载入向右模型
    Rdirection_Cascade.load("own_project/right.xml");
    if (Rdirection_Cascade.empty()) ...

    //载入右手掌模型
    Rpalm_Cascade.load("own_project/rpalm.xml");
    if (Rpalm_Cascade.empty()) ...

    //载入人脸模型
    Face_Cascade.load("own_project/haarcascade_frontalface_alt.xml");
    if (Face_Cascade.empty()) ...
}
```

```
Fist_Cascade.detectMultiScale(img, fists, 1.1, 2, 0 | CASCADE_SCALE_IMAGE, cv::Size(30, 30)); //识别拳头
Rdirection_Cascade.detectMultiScale(img, rdirections, 1.1, 2, 0 | CASCADE_SCALE_IMAGE, cv::Size(30, 30)); //识别向右
Rpalm_Cascade.detectMultiScale(img, rpalms, 1.1, 2, 0 | CASCADE_SCALE_IMAGE, cv::Size(30, 30)); //识别右掌
Face_Cascade.detectMultiScale(img, faces, 1.1, 2, 0 | CASCADE_SCALE_IMAGE, cv::Size(30, 30)); //识别人脸
```

```
//识别人脸
for (int i = 0; i < faces.size(); i++)
{
    //识别人脸 用黄色长方形框起来
    rectangle(img, Point(faces[i].tl().x - 50, faces[i].tl().y - 50), faces[i].br(), Scalar(0, 255, 255), 3);
}
//识别右手掌
for (int i = 0; i < rpalms.size(); i++)
{
    //识别右手掌 用蓝色长方形框起来
    rectangle(img, rpalms[i].tl(), rpalms[i].br(), Scalar(230, 216, 173), 3);
}
```

based on the functions, the next modes can work.

3. Use Global Variable to Change Mode

In the every while loop, the first thing is to check and judge the global variables' value, the whole system will jump to the different function modes, and realize the relative functions we want, which is shown below:

```

while (true)
{
    cap.read(img);

    Fist_Cascade.detectMultiScale(img, fists, 1.1, 2, 0 | CASCADE_SCALE_IMAGE, cv::Size(30, 30));
    Rdirection_Cascade.detectMultiScale(img, rdirections, 1.1, 2, 0 | CASCADE_SCALE_IMAGE, cv::Size(30, 30));
    Rpalm_Cascade.detectMultiScale(img, rpalms, 1.1, 2, 0 | CASCADE_SCALE_IMAGE, cv::Size(30, 30));
    Face_Cascade.detectMultiScale(img, faces, 1.1, 2, 0 | CASCADE_SCALE_IMAGE, cv::Size(30, 30));

    //模式0 开始界面
> if (mode == 0) ...

    //模式1 画轨迹
> if (mode == 1) ...

    //模式2 识别图像
> if (mode == 2) ...

    //模式3 人脸识别自拍
> if (mode == 3) ...

    //模式4 光标动作
> if (mode == 4) ...

    waitKey(25);
}

```

4. Trajectory and Hand Draw Graphics Detection

In this part, we aim to detect the trajectory from the movement of the moving object which in our project is the fist. In addition, we should generate the hand draw graphics using the trajectory. We make the right fist as the detect target. Once we detect the fist, we can use a rectangle to include it in the image. The center of the rectangle is then recorded and plays as a pen on the Mat from OpenCV. The center is stored as a Point in a vector each time there is a image input from the camera. The trajectory therefore is represented by those points. If we use the points project to the computer as the locations of the mouse, we then are able to control the movement of the moue. If we draw a line between the centers in order on the Mat, then we transform a set of points to an image. Since the graphics drawn by hand may not be standard, we have to further process the images to make them recognizable. Here, we draw a line between the start and end point of trajectory to ensure that the graphics are closed. Further more, we have to preprocess the image in order to get suitable sources for detection.

```

//画轨迹部分 img
for (int i = 0; i < fists.size(); i++)
{
    //识别拳头 用紫色长方形框起来
    rectangle(img, fists[i].tl(), fists[i].br(), Scalar(255, 0, 255), 3);

    //找到长方体中心点
    Point point;
    point.x = (fists[i].tl().x + fists[i].br().x) * 0.5;
    point.y = (fists[i].tl().y + fists[i].br().y) * 0.5;

    //显示中心点
    circle(img, point, 5, Scalar(0, 69, 255), FILLED);

    //存点坐标
    centers.push_back({point.x, point.y, i});
}

```

```

for (int i = 1; i < centers.size(); i++)
{
//鼠标控制移动
SetCursorPos(p.x + (-centers[i][0] + centers[i - 1][0]) / 2, p.y + (centers[i][1] - centers[i - 1][1]) / 2);
GetCursorPos(&p);
}

```

```

for (int i = 1; i < centers.size(); i++)
{
//画轨迹 输出到 imgdraw显示
line(imgdraw, Point(centers[i][0], centers[i][1]), Point(centers[i - 1][0], centers[i - 1][1]), Scalar(255, 0, 0), 5);
}

```

Graphics Detection

```

// 连接起始点和终点 封闭图形便于识别形状
line(imgdraw, Point(centers[0][0], centers[0][1]), Point(centers[centers.size() - 1][0], centers[centers.size() - 1][1]), Scalar(255, 0, 0), 5);
flip(imgdraw, show, 1);
// Processing 预处理
Mat imgGray, imgBlur, imgCanny, imgDil, imgErode;
Mat kernel = getStructuringElement(MORPH_RECT, Size(5, 5));

cvtColor(show, imgGray, COLOR_BGR2GRAY); //灰度图像
GaussianBlur(imgGray, imgBlur, Size(3, 3), 3, 0); //高斯模糊
Canny(imgBlur, imgCanny, 25, 75);
dilate(imgCanny, imgDil, kernel);
getContours(imgDil, show, revise);
Recognition_Count++;

//形状判断函数
void getContours(Mat imgDil, Mat img, Mat revise)
{
vector<vector<Point>> contours;
vector<Vec4i> hierarchy;
findContours(imgDil, contours, hierarchy, 0, 2);
// drawContours(img, contours, -1, Scalar(255, 0, 255), 2); //-1 means all
for (int i = 0; i < contours.size(); i++)
{
string objectType;
int area = contourArea(contours[i]);
cout << area << endl;
}
}

```

```

vector<vector<Point>> conPoly(contours.size());
vector<Rect> boundRect(contours.size());
if (area > 1000)
{
    float peri = arcLength(contours[i], true);
    approxPolyDP(contours[i], conPoly[i], 0.038 * peri, true);
    drawContours(img, conPoly, i, Scalar(255, 0, 255), 2);
    // cout << conPoly[i].size() << endl;
    boundRect[i] = boundingRect(conPoly[i]);
    rectangle(img, boundRect[i].tl(), boundRect[i].br(), Scalar(0, 255, 0), 5);
    int objCor = (int)conPoly[i].size();
    cout << "The numbers of angles" << objCor << endl;
    if (objCor == 3)
    {
        objectType = "Tri";
        shape2move = 1;
        drawContours(revise, conPoly, i, Scalar(255, 0, 255), 2);
    }
    else if (objCor == 4)
    {
        float aspRatio = (float)boundRect[i].width / boundRect[i].height;
        cout << aspRatio << endl;
        if (aspRatio > 0.8 && aspRatio < 1.2)
        {
            objectType = "Square";
            shape2move = 2;
            drawContours(revise, conPoly, i, Scalar(255, 0, 255), 2);
        }
        else
        {
            objectType = "Rect";
            shape2move = 3;
            drawContours(revise, conPoly, i, Scalar(255, 0, 255), 2);
        }
    }
    else if (objCor == 5)
    {
        objectType = "Pentagon";
        shape2move = 4;
        drawContours(revise, conPoly, i, Scalar(255, 0, 255), 2);
    }
    else if (objCor >= 7)

```

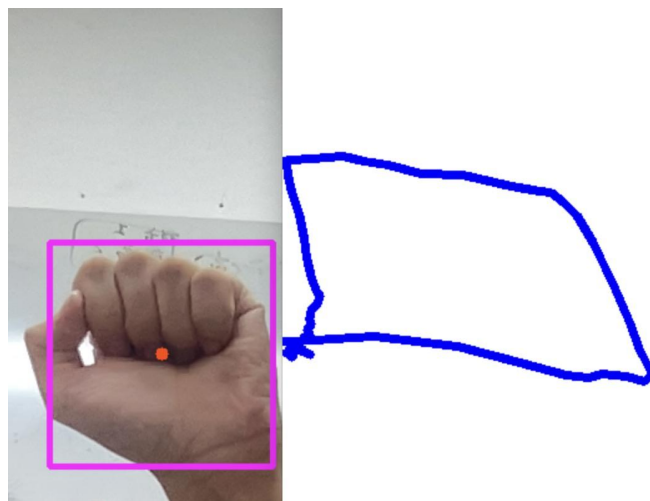
```

{
objectType = "Circle";
shape2move = 5;
drawContours(revise, conPoly, i, Scalar(255, 0, 255), 2);
}
putText(img, objectType, {boundRect[i].x, boundRect[i].y - 5}, FONT_HERSHEY_PLAIN, 1, Scalar(0, 69,
255), 1.5);
}
}
}

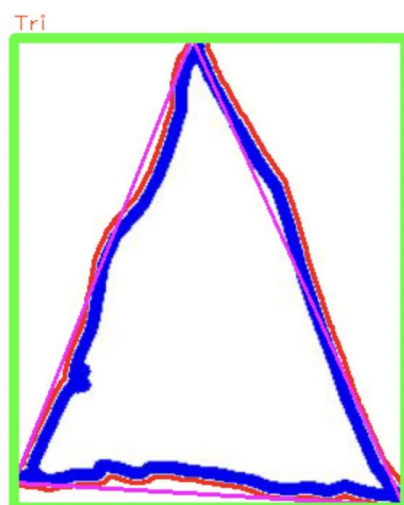
```

The blue line is the line we draw. The purple line is the line we approximate. The red line is the outline detected from the hand draw picture.

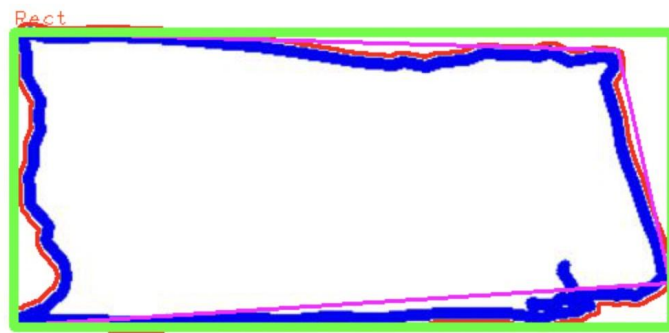
Test case #2: trajectory detection



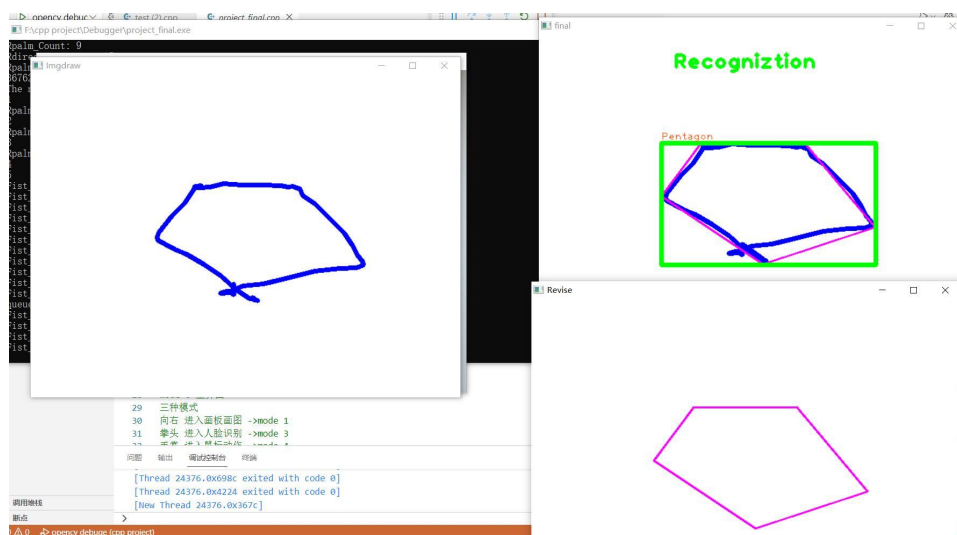
Test case #2: triangle



Test case #3: rectangle



Test case #4: pentagon



Mode 4 : Use the queue saved in Mode3, when mode == 4, check the size of the queue first. If the size equals 0, then mode changes into mode 0.

If the element in the queue is 1 (triangle), we can record the trajectory of the fist and use the SetCursorPos() function to control the mouse. If we open palms, mouse click operation can be realized. If the element in the queue is 2(square), use keybd_event() function to simulate pressing win +shift+ s on the keyboard.

If the element in the queue is 3(rectangle),use mouse_event() function to simulate mouse click.

If the element in the queue is 4(pentagon),use mouse_event() function to simulate mouse double click.

If the element in the queue is 5(circle),use system("C:\\Windows\\system32\\osk.exe") function to open the on-screen keyboard provided by the system.

Part 2 - Code

```

//模式4 光标动作
if (mode == 4)
{
    if (movements.size() != 0)
    {
        if (movements.front() == 1)
        {
            //画轨迹部分 img
            for (int i = 0; i < fists.size(); i++)
            {
                //识别拳头 用紫色长方形框起来
                rectangle(img, fists[i].tl(), fists[i].br(), Scalar(255, 0, 255), 3);

                //找到长方体中心点
                Point point;
                point.x = (fists[i].tl().x + fists[i].br().x) * 0.5;
                point.y = (fists[i].tl().y + fists[i].br().y) * 0.5;

                //显示中心点
                circle(img, point, 5, Scalar(0, 69, 255), FILLED);
                //存点坐标
                centers.push_back({point.x, point.y, i});
            }

            //识别右向
            for (int i = 0; i < rdirections.size(); i++)
            {
                //识别右向 用粉色长方形框起来
                rectangle(img, rdirections[i].tl(), rdirections[i].br(), Scalar(203, 192, 255), 3);
            }

            //识别右手掌
            for (int i = 0; i < rpalms.size(); i++)
            {
                //识别右手掌 用蓝色长方形框起来
                rectangle(img, rpalms[i].tl(), rpalms[i].br(), Scalar(230, 216, 173), 3);
            }
        }
    }
}

```

```

flip(img, Image_y, 1); //将图像沿y轴翻转，即镜像

string size = "queue size: " + to_string(movements.size());
putText(Image_y, size, Point(300, 50), FONT_HERSHEY_PLAIN, 2, Scalar(0, 255, 255), 3);

imshow("Img", Image_y); //摄像头图像
moveWindow("Img", 100, 100);

for (int i = 1; i < centers.size(); i++)
{
    //鼠标控制移动
    SetCursorPos(p.x + (-centers[i][0] + centers[i - 1][0]) / 2, p.y + (centers[i][1] - centers[i - 1][1]) / 2);
    GetCursorPos(&p);
}
cout << "1光标移动" << endl;

//向右识别判断
if (rdirections.size() != 0) //识别到向右 则Rdirection_Count++
{
    Rdirection_Count++;
    std::cout << "Rdirection_Count: " << Rdirection_Count << endl;
}
if (Rdirection_Count == 15) //识别到一定次数后 转换成mode
{
    mode = 4;
    Rdirection_Count = 0; //清空 便于下一次识别
    movements.pop();
    centers.clear();
}
//右手掌识别判断
if (rpalms.size() != 0) //识别到向右 则Rdirection_Count++
{
    Rpalm_Count++;
    std::cout << "Rpalm_Count: " << Rpalm_Count << endl;
}

if (Rpalm_Count == 15) //清空画板
{
    mouse_event(MOUSEEVENTF_LEFTDOWN, 0, 0, 0, 0); //模拟鼠标键按下
    Sleep(10); //要留给某些应用的反应时间
    mouse_event(MOUSEEVENTF_LEFTUP, 0, 0, 0, 0); //模拟鼠标键抬起
    Rpalm_Count = 0;
}
}
if (movements.front() == 2)
{
    keybd_event(VK_LWIN, 0, 0, 0);
    keybd_event(VK_LSHIFT, 0, 0, 0);
    keybd_event(83, 0, 0, 0);
    keybd_event(VK_LWIN, 0, 2, 0);
    keybd_event(VK_LSHIFT, 0, 2, 0);
    keybd_event(83, 0, 2, 0); //快速截图
    cout << "2截图" << endl;
    movements.pop();
}
if (movements.front() == 3)
{
    mouse_event(MOUSEEVENTF_LEFTDOWN, 0, 0, 0, 0); //模拟鼠标键按下
    Sleep(10); //要留给某些应用的反应时间
    mouse_event(MOUSEEVENTF_LEFTUP, 0, 0, 0, 0); //模拟鼠标键抬起
    cout << "3单击" << endl;
    movements.pop();
}
if (movements.front() == 4)
{
    mouse_event(MOUSEEVENTF_LEFTDOWN, 0, 0, 0, 0); //模拟鼠标键按下
    Sleep(10); //要留给某些应用的反应时间
    mouse_event(MOUSEEVENTF_LEFTUP, 0, 0, 0, 0); //模拟鼠标键抬起
    mouse_event(MOUSEEVENTF_LEFTDOWN, 0, 0, 0, 0); //模拟鼠标键按下
    Sleep(10); //要留给某些应用的反应时间
    mouse_event(MOUSEEVENTF_LEFTUP, 0, 0, 0, 0); //模拟鼠标键抬起
    cout << "4双击" << endl;
    movements.pop();
}

```

```

        mouse_event(MOUSEEVENTF_LEFTDOWN, 0, 0, 0, 0); //模拟鼠标键按下
        Sleep(10); //要留给某些应用的反应时间
        mouse_event(MOUSEEVENTF_LEFTUP, 0, 0, 0, 0); //模拟鼠标键抬起
        mouse_event(MOUSEEVENTF_LEFTDOWN, 0, 0, 0, 0); //模拟鼠标键按下
        Sleep(10); //要留给某些应用的反应时间
        mouse_event(MOUSEEVENTF_LEFTUP, 0, 0, 0, 0); //模拟鼠标键抬起
        cout << "4双击" << endl;
        movements.pop();
    }
    if (movements.front() == 5)
    {
        system("C:\\Windows\\system32\\osk.exe"); //打开键盘
        cout << "5虚拟键盘" << endl;
        movements.pop();
    }
}
else
{
    cout << "No Movement in queue" << endl;
    mode = 0;
    destroyAllWindows();
}
}

```

Part 3 - Result & Verification

There are a variety of modes, among which, the global variable mode is used, mode = 0 to jump to the main interface, mode = 1 to jump to the drawing board, mode = 2 to jump to the image recognition module, mode = 3 to jump to the selfie and face screenshot module, mode = 4 to jump to the mouse action. The following describes the specific mode functions.

Mode 0 :Main screen

Brief Description: Three modes

Go right: to artboard and draw and jump mode 1

Fist: into face recognition and jump mode 3

Palm: into mouse action and jump mode 4

Mode 1 :sketchpad

Brief Description: Draw trajectory

Go right: into recognition image and jump mode 2

Fist: trace

Palm: Empty the board and redraw

Mode 2 :Identifies images

Brief Description: The trajectory is fitted to estimate the shape and output the result

Go right: back to the main screen and jump mode 0

Fist: loads the result into a queue for mouse action

Palm: returns mode 1 unsatisfied to redraw/continue adding results to the load queue

The default order is mode 0 -> mode 1 -> mode 2

Mode 3 :Self-portrait face recognition

Brief Description: Recognize the face and then select the screenshot

Go right: back to the main screen and jump mode 0

Palm: Save the screenshot of your face to a folder with your right palm

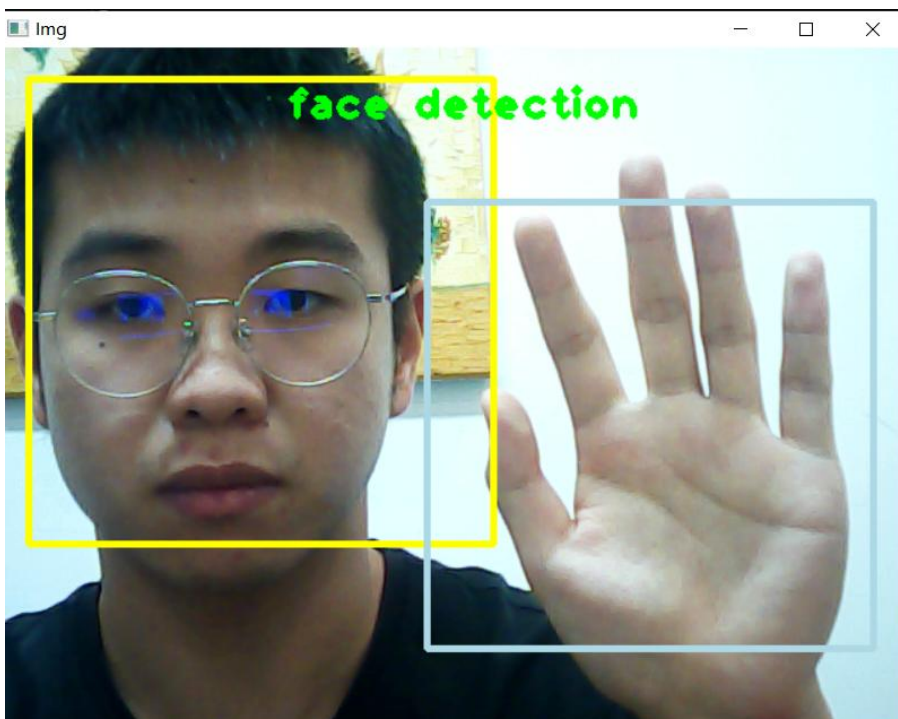
Mode 4 :Cursor action

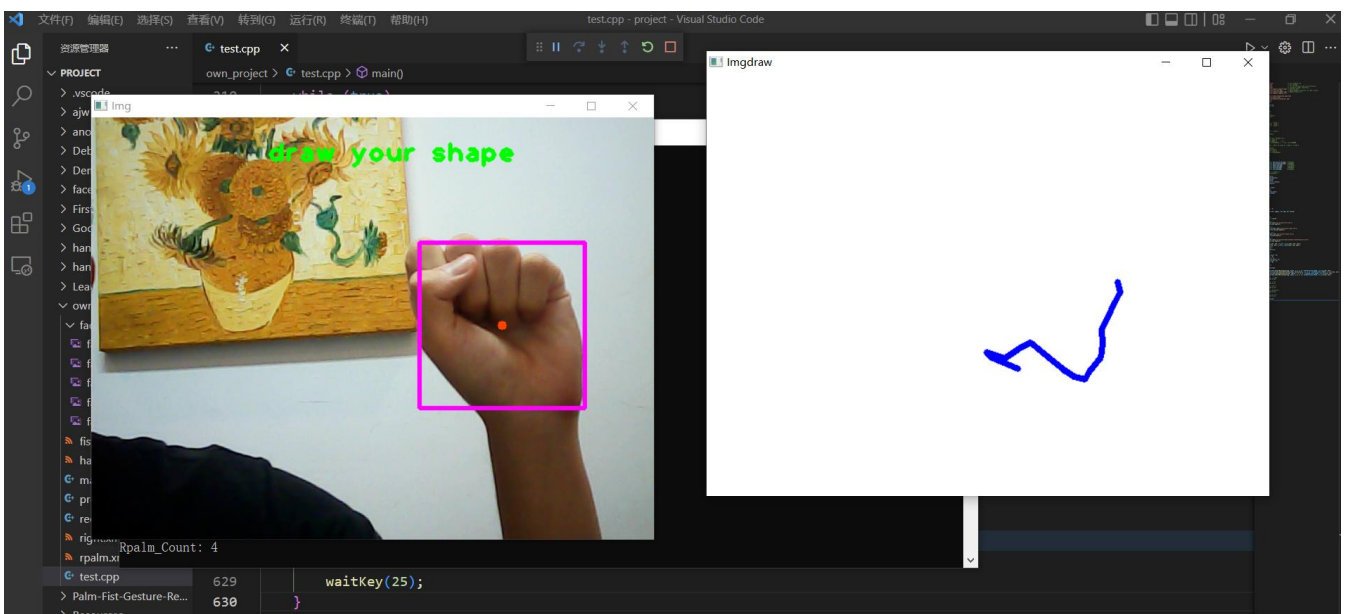
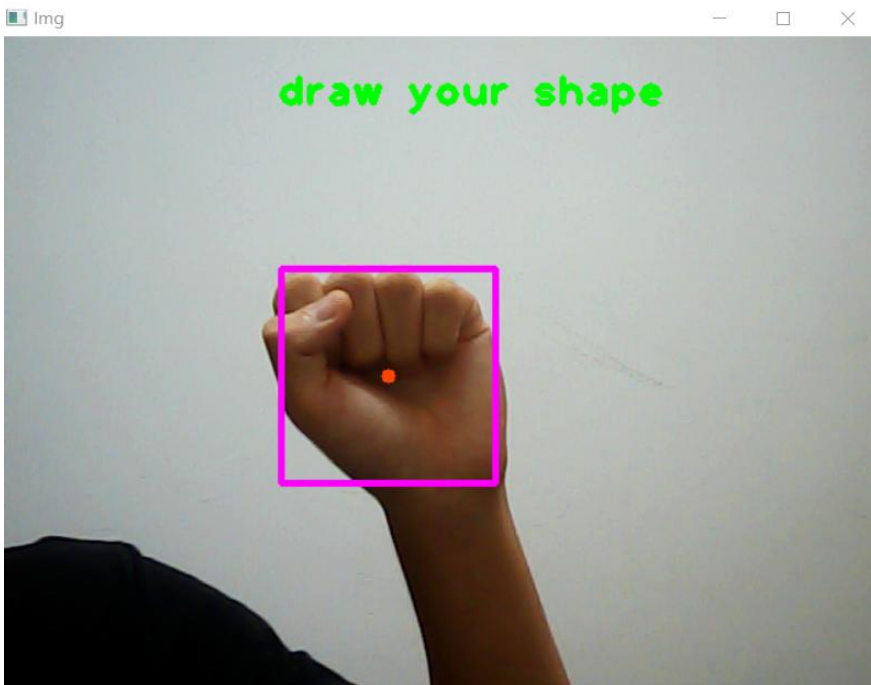
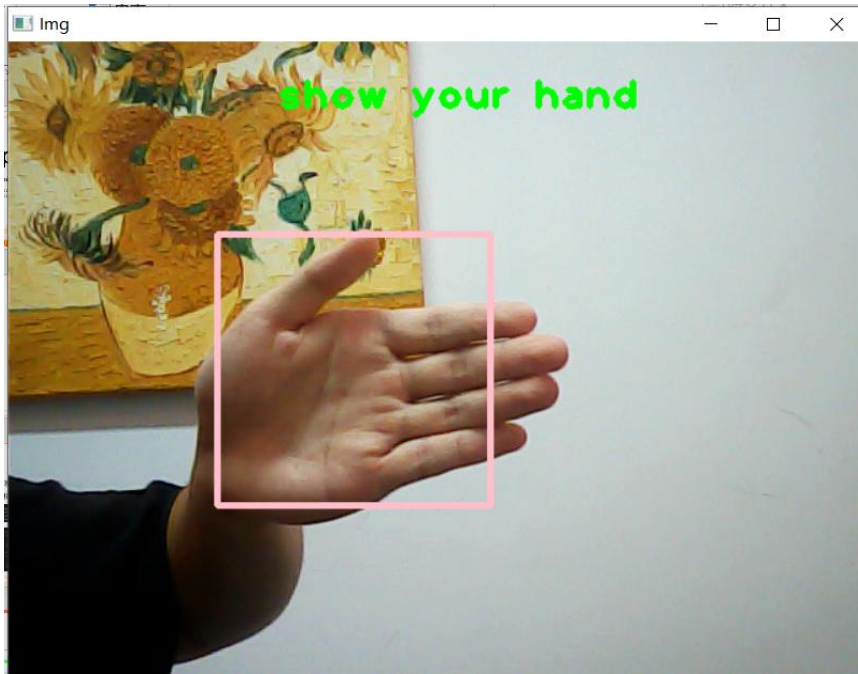
Brief Description:Identify pre-stored actions (graphics) perform different actions according to the different numbers stored

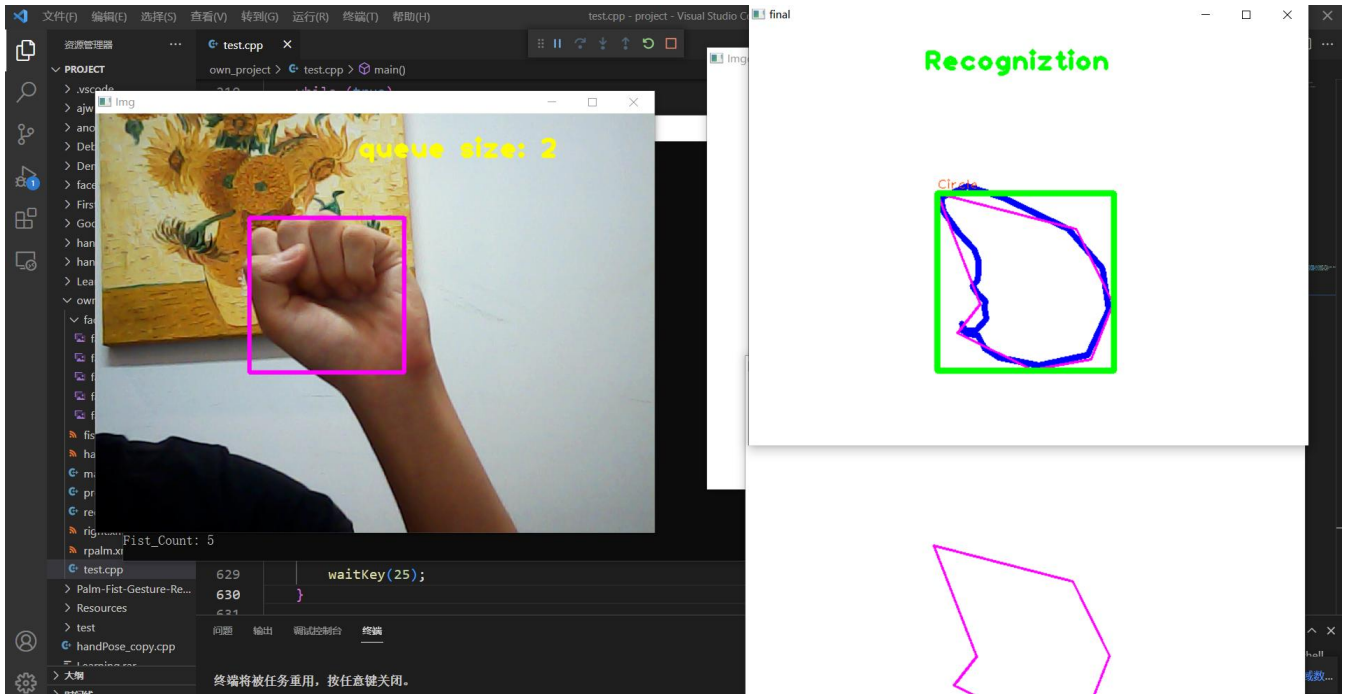
Tri—1, Squa—2, Rect—3, Pentagon—4, Circle—5,

Palm: Next move to the right

And the specific functions' affect was shown below:







Part 4 - Difficulties & Solutions

The first and the most difficult problem is the hand recognition part. At first the color detector was chosen, but the face and hand have the same color, which will cause distortion in detection. And then the training mode is used. However, the training process is difficult and we have no idea to find enough photos to train our model properly and precisely. Finally, through long-time finding and testing, the proper model already trained was used in github.

The point is read from the location of the center of the fist. Once another object is misidentified as the fist, the point can jump from one place to another place far away. So we have calculated the distance between the points; we will draw a line between them. If the distance is larger than the threshold value, the point will not be put into the vector. As a result, it won't draw the line in the Mat.

Since there may be unwanted lines included in the graphics. The function `findContours` from OpenCV is used to extract the outline of the graphics we draw, so that the internal message is discarded.

The outline we extract is also not standard. The lines may not be straight, new corners may appear. To solve this, we use the polygonal approximation method which uses the Douglas-Peucker algorithm to get the corners we want. The threshold value for this algorithm has been set a little large to transform the crooked line resulting from handshake and detection error to a straight line.