SUSTech, 2022 Spring

CS205: C/C++ Programming Language

Project II Report

Authors:

12011517 李子南 12011522 丁泓勋 12011537 陆荻芸

Contributions: 33% each

Contents

1 General Introduction

In this project, we are required to implement a non-contact interactive system based on OpenCV library. Basic steps include getting hand contour from the images read from camera, tracking the moving hand and store its trajectory, analyzing and present specific meanings of the given trajectory and further modifying the trajectories. The following report will introduce our project in 3 main aspects. And our whole process is roughly as the flow chart shown below.

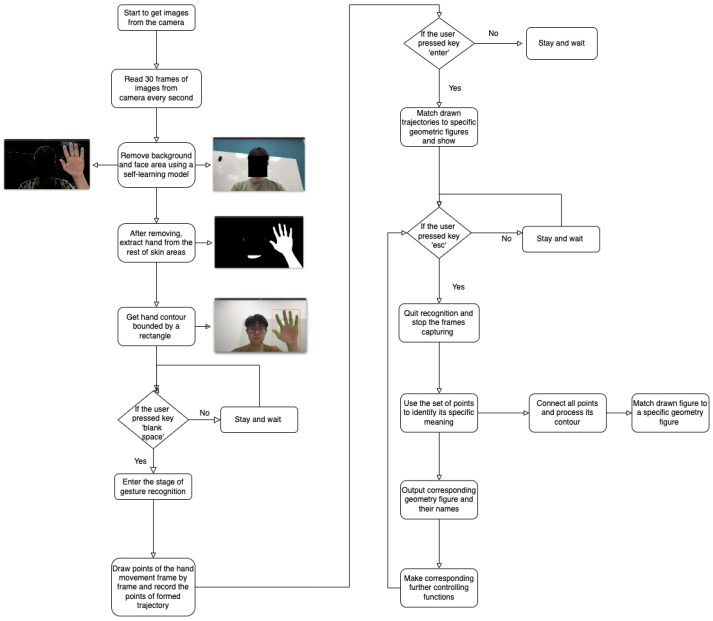


Figure 1: Flow chart of whole process

2 Detail Implementation

2.1 Hand Recognition

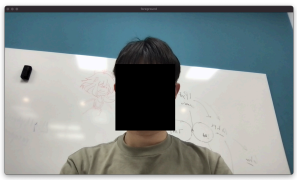
[Sof] To begin with, we read 30 frames of images from camera every second. Based on the images we get, we first removed background and used the color of skin to extract skin areas. Generally, face and hands are two main skin areas that will be captured in camera. Thus a mask is applied to face and the rest of connected skin components are recognized as hand. After basic computations, we locate the coordinates of the central of palm. Further steps will be carried based on the coordinates vary from every frames. More details will be mentioned in the **section** 3 part.

Basic effects of above steps seen pictures below.

2.2 Trajectory Analysis

[Kuk] Points read from the images are stored in structure *vector<Point>* as a member of class *Trajectory* which is used to handle trajectory analysis.

(a) Background removal



(b) Mask on face

(c) Extraction of hand

(d) Hand Contour

Figure 2: Part one result

2.2.1 Matching

Methods $findContours()$, $approxPolyDP()$ are the main methods used here which are encapsulated in OpenCV. The basic idea is to get the corner points of given trajectory to classify. We allowed the identification from lines with 2 corner points to heptagons with 7 corner points. Also circles are supported to be identified.

```
findContours( image: imgDil,  contours: contours,  mode: RETR_EXTERNAL,  method: CHAIN_APPROX_SIMPLE);
double perimeter = arcLength( curve: contours[0],  closed: true);
approxPolyDP( curve: contours[0],  approxCurve: outPutContour,  epsilon: 0.025 * perimeter,  closed: true);
```

Figure 3: Two main methods

We draw lines to connect given set of points and write is to a *Mat* to process it further. Gaussian Blur, Canny Edge Detection and Dilate Operation are applied to this *Mat* in order to get a clearer contour as the drawn trajectory may be very small or in other conditions difficult to recognize. Then 2 methods mentioned previously are applied to the *Mat* and the result set of points, corner points are stored. Also, type of the trajectory is matched through the number of corner points.

2.2.2 Drawing and Other Process

Got the type of the trajectory and corner points, we can recover the graph through drawing lines to connect the points. As for circles, extra computation is required using 3 coordinates we have. What's more, we need to clear the *vector* member variable each time before we receive a new set of points. If it is not empty, it means that we should enter the matching and drawing stage.

Pictures of process mentioned above is shown below.

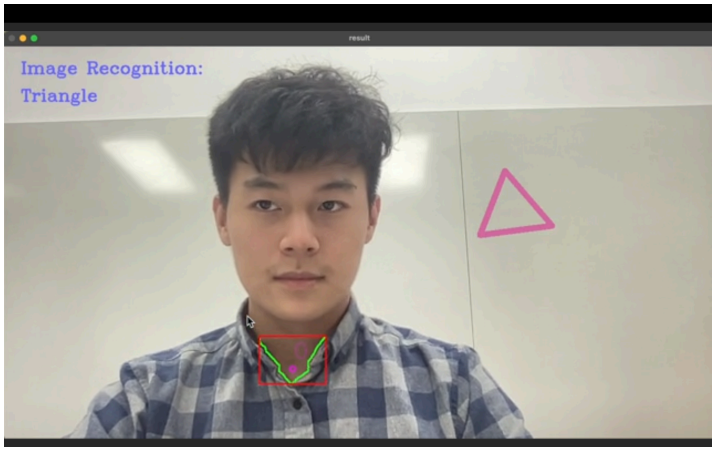


Figure 4: An example of drawing result

2.3 Further Control

According to requirements, further controlling functions should be applied to validate our identification of specific trajectories. Here we choose to relate our identification to taking screenshots. Three options are provided. If a triangle is drawn, screenshot of gray image will be taken. If a quadrangle is drawn, a erode screenshot will be taken. And for a drawn circle, screenshot of the original image will be taken. Corresponding effects seen pictures shown below.

3 Innovations

3.1 Outlier Removal

As many unexpected perturbation may occur when drawing using hand in reality, outlier may arise in trajectories. In order to reduce the effect of outlier, we make a specific judge. For every point we get, we draw circles of assigned radius, if number of points in the circle area is less than 2, then we mark it as a outlier and ignore it.

3.2 Multiple Types of Recognition

In consideration of usability, we develop auto recognition mode and manual recognition mode.

In auto recognition mode, background is gradually deleted by KNN Algorithm. It is updated in real time and runs fast. Users' skin area is separated using default skin color in both HSV and YCRCB color space.

In manual recognition mode, background is removed by using user defined captured, user can refresh their captured at any time. System will collect users' skin color(mean value of certain areas) and separate the skin area.

3.3 Choice of recognizing specific trajectories

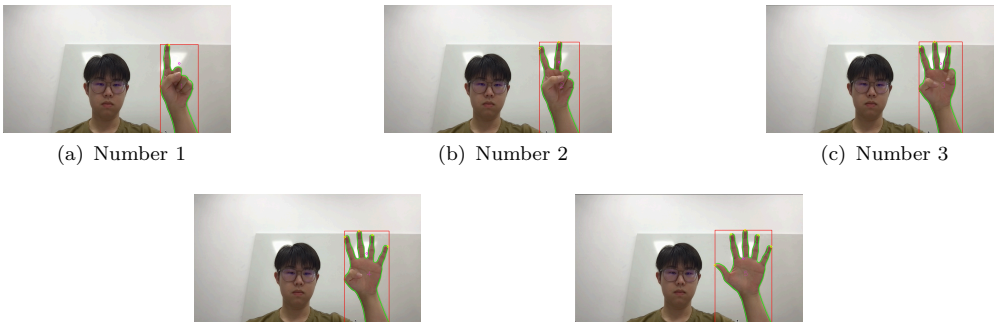
Under real circumstances, we don't always want our trajectories to be identified with geometric meanings. Therefore we provide options of whether you want to identify your drawn

trajectory or not. In detail implementation, you can press blank space to start drawing and press it again to indicate your stop. Once you stop, the drawn part is shown on the screen. And if you press enter key now, the trajectory will be recognized and the result will be display.

3.4 Finger Recognition

Not only support recognizing the hand area, we can also recognize the number of fingers. We use the convexhull function to find convex point in the area and exclude the wrong point by the angle between points.

Concrete implementation is shown below.



(a) Number 1 (b) Number 2 (c) Number 3

(d) Number 4 (e) Number 5

Figure 5: Part one result

4 Difficulties and Solutions

4.1 Limitations of Hardware

When we read from the camera and detect our hand's movement, problems arise due to the limitations of hardware. We observed that our camera may not capture exactly number of frames as expected and points of trajectory are sparse. Thus we experimented on both computer with Intel chip and M1 chip. We discovered that we can draw more fluently with M1 chip.

4.2 Limitations of OpenCV Library

In the process of development, we found that we must use absolute path to read model or image when using Opencv on macOS, which brought us a lot of troubles.

Another problem is the color space of image. When input images are in different color space, many functions of Opencv will process with no error but wrong answer. Such case makes it harder to find the bugs.

5 Conclusion

In this project, we implement hand recognition via background and face removal and skin area extraction. And the trajectory of hand drawing is recorded and analyzed. Our system can recognize specific shapes such as circle, triangle and rectangle and so on. On the basis of recognizing the shape, we modify the shape to a more standard shape. Additionally, user can capture different screenshots by using the graphic.

References

[Kuk] Kukeoo. 计算机视觉 *OpenCV*【七：应用之形状与轮廓检测】. URL: https://blog.csdn.net/Kukeoo/article/details/116328341?ops_request_misc=%5C%257B%5C%2522request%5C%255Fid%5C%2522%5C%253A%5C%252216301592681678026403418 0%5C%2522%5C%252C%5C%2522scm%5C%2522%5C%253A%5C%252220140713.130102334.pc%5C%255Fall.%5C%2522%5C%257D&request_id=163015926816780264034180&biz_id=0&utm_medium=distribute.pc_search_result.none-task-blog-2~all~first_rank_ecpm_v1~rank_v29-1-116328341.pc_search_ecpm_flag%5C&utm_term=conPoly%5C%5Bi%5C%5D.size%5C%28%5C%29&spm=1018.2226.3001.4187.

[Sof] Pierfrancesco Soffritti. *Handy, hand detection with OpenCV*. URL: https://pierfrancesco-soffritti.medium.com/handy-hands-detection-with-opencv-ac6e9fb3cec1.