

C++ Project Report

小组成员	贡献比	学号
唐昕宇	33.33%	12011439
徐春骥	33.33%	12011408
刘锦润	33.33%	12011216

C++ Project Report

第一部分：Windows下OpenCV的安装和配置

第二部分：项目使用的介绍

 软件的使用

 各个窗口的介绍

第三部分：识别手的轮廓：手部识别与手势识别

 将用户的手从图像中剥离

 如何找到用户的皮肤颜色

 二值化

 删除用户的脸

 背景去除

 手和手指检测

 手部轮廓

 手指识别(Bonus)

第四部分：轨迹的检测，绘制与存储

第五部分：轨迹修正与识别

 1. 图像预处理

 2. 轮廓检测

 3. 形状的检测与拟合

 4. 检测结果的打印与绘制

 5. 结果测试

第六部分：根据轨迹控制鼠标事件

第七部分：特色功能实现

 1. 用户图形界面部分

 1.1 首页以及引导页面

 遇到的问题及解决：

 1.2 画笔颜色及粗细设置

 1.3 摄像头参数设置

 2. 手部识别以及手势识别优化

 3. 键盘使能

总结与展望

第一部分：Windows下OpenCV的安装和配置

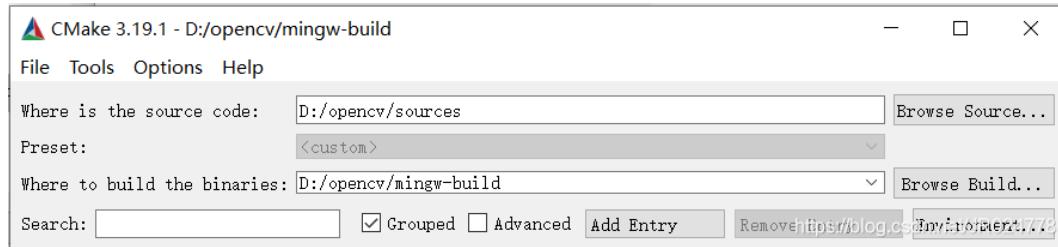
- 从官网下载OpenCV到D盘中

> 此电脑 > Data (D:) > opencv

名称	修改日期	类型	大小
build	2022/5/9 21:35	文件夹	
Learning OpenCV 3 Computer vision in C++	2022/5/8 14:49	WPS PDF 文档	43,793 KB
README.en	2022/5/8 14:49	Markdown File	1 KB
README	2022/5/8 14:49	Markdown File	1 KB

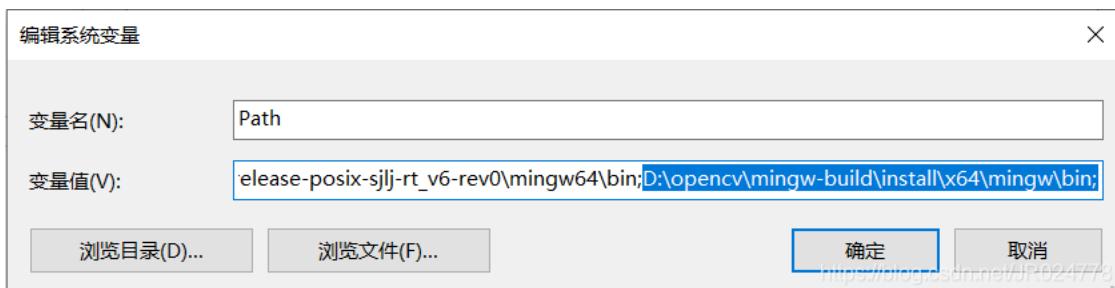
- 生成CMake文件

- 新建文件夹D:/opencv/MinGW-build，打开CMake(gui)，点击Browse Source...和Browse Build...选择源文件目录和生成文件目录。分别选择D:/opencv/sources和D:/opencv/MinGW-build，如下图：



- 点击Configure，在弹出的窗口选择类型为MinGW Makefiles.

- 打开cmd进入mingw-build目录，使用 mingw32-make install 命令
- 配置环境变量，将D:/opencv/mingw-build/install/x64/mingw/bin加入环境变量中的path中。



- 遇到的问题与解决方法

- 因为懒得下CMake，起初我们从网上找到已经包含了别人已经编译好的OpenCV文件夹，将其中lib子文件夹中的文件复制到mingw/lib中，但是在CLion建立一个使用OpenCV的工程时，CMakeList文件不能正常工作。
- 最终发现是因为我们并没有改写CMake文件-----它使得CMakeList在工作时需要寻找一个并不存在我们电脑中的文件夹，然后报错。

```

38 # Import target "opencv_ml" for configuration "Release"
39 set_property(TARGET opencv_ml APPEND PROPERTY IMPORTED_CONFIGURATIONS RELEASE)
40 set_target_properties(opencv_ml PROPERTIES
41   IMPORTED_IMPLIB_RELEASE "${_IMPORT_PREFIX}/opencv_old/build/lib/libopencv_ml452.dll.a"
42   IMPORTED_LOCATION_RELEASE "${_IMPORT_PREFIX}/opencv_old/build/bin/libopencv_ml452.dll"
43 )
44

```

```

38 # Import target "opencv_ml" for configuration "Release"
39 set_property(TARGET opencv_ml APPEND PROPERTY IMPORTED_CONFIGURATIONS RELEASE)
40 set_target_properties(opencv_ml PROPERTIES
41   IMPORTED_IMPLIB_RELEASE "${_IMPORT_PREFIX}/x64/mingw/lib/libopencv_ml452.dll.a"
42   IMPORTED_LOCATION_RELEASE "${_IMPORT_PREFIX}/x64/mingw/bin/libopencv_ml452.dll"
43 )
44

```

第二部分：项目使用的介绍

软件的使用

现在您将拥有四个黑色窗口。要使用程序执行以下操作序列：

0.在开始界面，按2可以查看指令集，按1可以正式开始程序。

1.不要在画面中显示你的手，按下键盘上的B键。它将开始背景移除的过程。

2.移动你的手，让它完全覆盖窗口中显示的两个称为输出的紫色矩形。按下键盘上的S键。这将抽样你的皮肤颜色，并开始过程的手检测。

- 3.伸出食指，按下T键，可以进行图形的绘画。
- 4.画好轨迹之后，按下A键，开始分析轨迹类型并自动规范化。
- 5.当你想要关闭程序按下键盘上的esc键。

各个窗口的介绍

- 1.output窗口显示应用程序的输出。手中心的紫色数字代表软件检测到的被举起手指的数量。
- 2.foreground窗口显示了背景移除操作的输出。从这里你可以很容易地看到背景移除是否被正确地执行。如果没有，按B重新校准。
- 3.handMask窗口显示了整个二值化过程的输出。这是用来检测手和数手指的图像。
- 4.handDetection窗口显示手和手指检测过程的输出。这里显示的数字对应数组中点的索引(如下所示)。手中心的紫色数字代表软件检测到的被举起手指的数量。
- 5.“Integrated Camera属性”窗口，可以更改摄像头的属性，例如亮度、对比度等。

第三部分：识别手的轮廓：手部识别与手势识别

将用户的手从图像中剥离

从复杂性和可靠性的角度来看，我们解决这个问题的方法是从用户皮肤的颜色开始分割手。这个想法很简单，即找到用户皮肤的颜色，并将其作为二值化图像的阈值。（识别到的部分（即手）的像素设为1，背景和面部的像素设为0）

如何找到用户的皮肤颜色

为了找到皮肤的颜色，我们决定在屏幕上指定两个特定的区域作为“样本区域”。用户必须在框架中移动他的手，以便它覆盖两个样本区域。当按下键盘上的S键时，程序保存样本区域中包含的图像，对颜色进行平均，然后用这两个平均的颜色作为低阈值和高阈值来找到用户的皮肤。

这个解决方案非常简单，可以通过多种方式进行改进。例如，准确度可以通过取更多的样本来提高。

```
class SkinDetector {  
public:  
    SkinDetector();  
  
    void drawSkinColorSampler(Mat input);  
    void calibrate(Mat input);  
    Mat getSkinMask(Mat input);  
  
private:  
    int hLowThreshold = 0;  
    int hHighThreshold = 0;  
    int sLowThreshold = 0;  
    int sHighThreshold = 0;  
    int vLowThreshold = 0;  
    int vHighThreshold = 0;  
  
    bool calibrated = false;  
  
    Rect skinColorSamplerRectangle1, skinColorSamplerRectangle2;  
  
    void calculateThresholds(Mat sample1, Mat sample2);  
    void performOpening(Mat binaryImage, int structuralElementShape, Point  
structuralElementSize);  
};
```

二值化

帧的二值化是使用OpenCV的inRange函数完成的。运算符简单地将低阈值和高阈值之间的所有像素设为1，将所有其他像素设为0。

经过二值化后，由于误报，图像产生了一些噪声。为了清理图像并去除假阳性，应用了一个开放算子，带有一个3x3的圆形结构元素。

如果手的部分在二值化后脱落(有时手指从手上脱落)，也可以进行扩张。

删除用户的脸

用户的皮肤现在是我们的二值图像中的对象。除了用户的手，用户的脸也是我们的对象的一部分。这显然是不可取的。

为了防止我们的算法提取用户的脸，在图像二值化之前，通过在用户的脸上面画一个黑色矩形来检测和删除用户的脸。利用OpenCV提供的人脸分类器解决了人脸检测问题。

```
void FaceDetector::removeFaces(Mat input, Mat output) {
    vector<Rect> faces;
    Mat frameGray;

    cvtColor(input, frameGray, CV_BGR2GRAY);
    equalizeHist(frameGray, frameGray);

    faceCascadeClassifier.detectMultiScale(frameGray, faces, 1.1, 2, 0 |
    CASCADE_SCALE_IMAGE, Size(200, 150));

    for (size_t i = 0; i < faces.size(); i++) {
        rectangle(
            output,
            Point(faces[i].x, faces[i].y),
            Point(faces[i].x + faces[i].width + 50, faces[i].y + faces[i].height
+ 50),
            Scalar(0, 0, 0),
            -1
        );
    }
}
```

背景去除

在这一点上，程序正在工作，但由于场景条件的不可预测性，它不是特别可靠。简单地改变光照或者背景的颜色与用户皮肤的颜色过于相似，可能会产生很多误报。为了解决这个问题，我们决定在我们的过程中添加一个额外的步骤：在二值化之前去除背景。

在这种方法中，手必须一直移动，否则它被归类为背景，然后被去除。

当应用程序启动时，第一帧被保存为引用。然后对于每个新帧，我们简单地迭代帧中的每个像素，并将其与参考帧中对应的像素进行比较。如果当前帧中的像素与参考帧中的对应像素有一定的差异，则不删除该像素，否则将该像素归为背景像素并删除。

```

class BackgroundRemover {
public:
    BackgroundRemover(void);
    void calibrate(Mat input);
    Mat getForeground(Mat input);

private:
    Mat background;
    bool calibrated = false;

    Mat getForegroundMask(Mat input);
    void removeBackground(Mat input, Mat background);
};


```

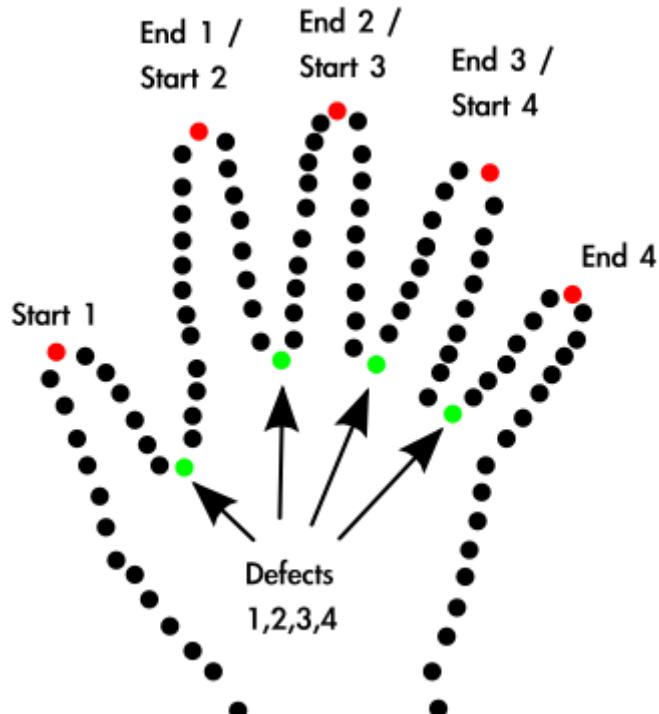
手和手指检测

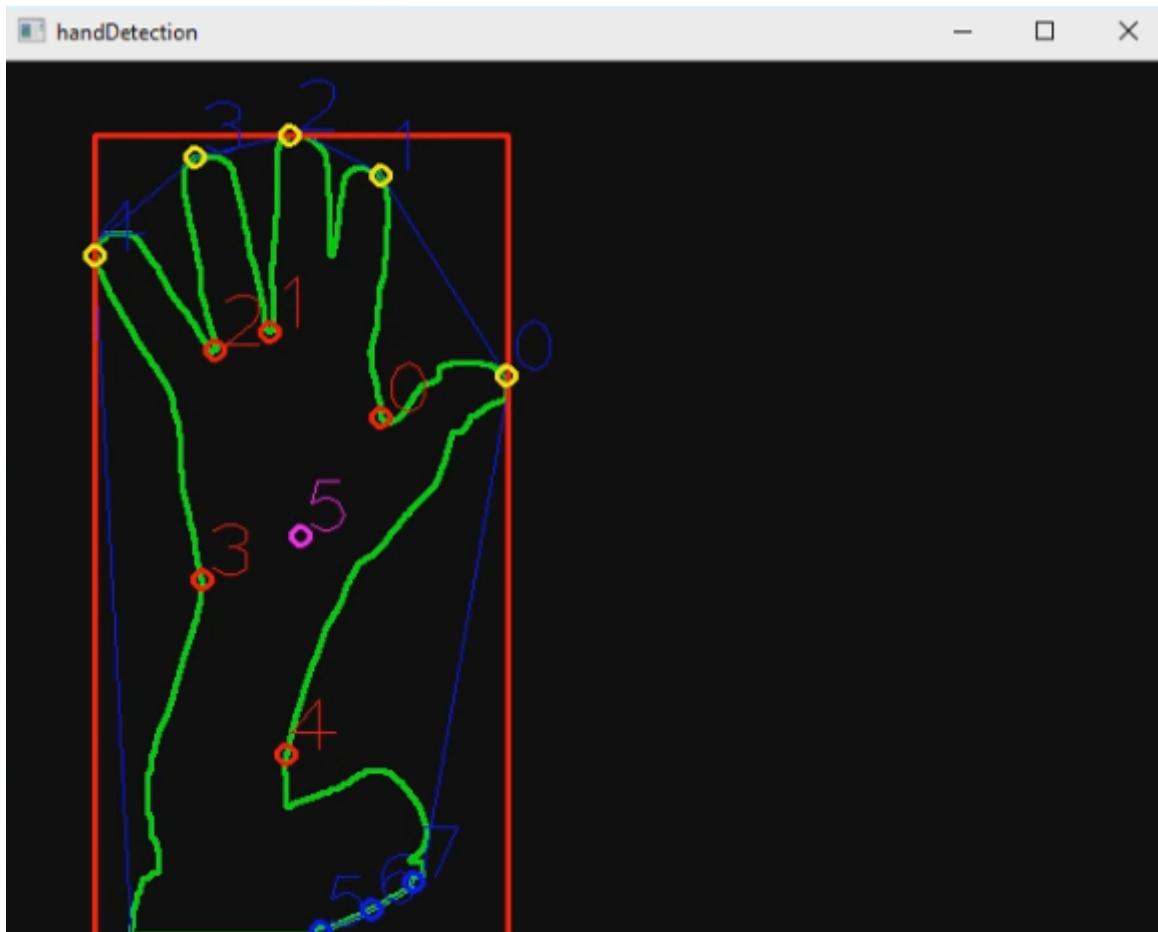
手部轮廓

现在我们有了二值图像，我们使用OpenCV的函数findContours来获得图像中所有对象的轮廓。从中选择面积最大的轮廓线。如果图像的二值化处理是正确的，这个轮廓应该是我们的手。

手指识别(Bonus)

利用convexityDefects函数，我们可以得到手部轮廓的所有凹陷，并将它们保存在另一个数组中。这是指两根手指之间的最低点。我们可以使用这两个数组来假设图像中被举起的手指的数量。





第四部分：轨迹的检测，绘制与存储

- 在第二部分中我们成功识别了手部，并得到了一些标志点例如指尖，虎口的位置信息。我们可以选择一个固定的标志点进行轨迹的绘制。
- 在我们的实现中，我们选择这些标志点的最高点作为“画笔”绘制轨迹。
- 将这些点记录下来，以供轨迹修正与识别

```
void printTrack(Point curPoint, Mat &srcImage)
{
    if (state == DRAWTRACK && curPoint.x <= 250)
    {
        circle(srcImage, curPoint, 0, Scalar(dgBlue, dgGreen, dgRed), thick
+ 1);
        if (Track.empty() || curPoint != Track.back())
            Track.push_back(curPoint);
    }
}
```

- 问题与解决方法
 - 视频画面刷新很快，使得我们记录的轨迹中会出现大量的重复的点。而且有时由于我们识别的标志点并非十分稳定，会出现抖动现象，有可能记录到与我们手运动反向相反的点。
我们可以通过判断当前点是否在一定范围内与前点处于相同位置，如果是便不向记录轨迹的集合传入该点，这样可以有效的减少冗余的点并起到一定的降噪作用
 - 当手与脸的位置接近时，我们的算法很容易混淆目标失去对手部的追踪，导致大量离群点的出现。
我们可以忽略出现在右半部分（脸所在的部分）的点。

- 结果展示



第五部分：轨迹修正与识别

函数shape_analysis进行实现，向该函数传入元素类型为Point的vector数组，该数组为轨迹纠正之后的点集。前面所做的纠正仅将临近的点进行删除，在形状拟合时仍会进行进一步纠正。首先将点按序连接，得到的图形为初步的轨迹图像，对此图像进行形状与轮廓的检测。

1. 图像预处理

为了能找到图像的边缘以及图像的轮廓，需要对上面得到的图像进行预处理。轮廓就是连接所有具有相同颜色或灰度值的连续点，因此需要先利用opencv的cvtColor函数将图像转为灰度图像。同时利用GaussianBlur函数对图像进行高斯模糊，进行阈值处理，去除噪声点，提高轮廓提取的可行性。

```
Mat picGray; //转为灰度
cvtColor(pic, picGray, COLOR_BGR2GRAY);
Mat picBlur; //添加高斯模糊
GaussianBlur(picGray, picBlur, Size(3,3), 3, 0);
```

为了提高轮廓检测的精确性，通过进行坎尼边缘检测将图像转化为二值图。最后进行膨胀操作，使边缘间没有空隙，成为完整的封闭图。

```
Mat picCanny; //坎尼边缘检测
Canny(picBlur, picCanny, 25, 75);
Mat picDil; //进行膨胀操作
Mat kernel = getStructuringElement(MORPH_RECT, Size(3,3));
dilate(picCanny, picDil, kernel);
```

操作完成即结束图像的预处理，将预处理完成使图像作为函数参数传入，对该图像的轮廓检测与形状拟合在函数getCounter中完成。

2. 轮廓检测

在getCounter的函数中，contours为二维vector容器，由于我们一次只画一个图像，因此外层永远只有一个元素。内层每个vector存放的是一个形状轮廓点的集合。而hierarchy变量类型为vector<Vec4i>，即为一个每个元素包含了4个int变量的向量。外层仍然永远为1，这四个int型变量分别是该轮廓的后一个轮廓，前一个轮廓，父轮廓以及内嵌轮廓的索引编号。若没有则int为-1。可以看到虽然我们只有一个图形，但仍然使用了二维vector，这是因为可以直接使用opencv自带的findContours函数。

```
vector<vector<Point>> contours;
vector<Vec4i> hierarchy;
findContours(picDil, contours, hierarchy, RETR_EXTERNAL, CHAIN_APPROX_SIMPLE);
```

3. 形状的检测与拟合

为了决定形状拟合时所允许的误差，首先要先进行图像周长的检测，由于前面进行轨迹纠正并绘制了新的图形，这里默认图像一定闭合。因此后面一个参数为true。

```
float perimeter = arcLength(contours[i], true);
```

利用opencv自带的函数approxPolyDP，找出轮廓的多边形拟合曲线。参数一为输入的点集；参数二为输出的点集；参数三为指定精度，这里利用得到的图形周长控制精度；参数四表示是否闭合，同上，恒为true。

```
approxPolyDP(contours[i], conPoly[i], 0.05 * perimeter, true);
```

图形形状的判断即通过拟合后获得的拐点数，拐点是几即为几边形图形。

```
int num = (int)conPoly[i].size();
```

至此我们已经完成形状的检测，并在原来轨迹的基础上进行了拟合，即纠正了原来的轨迹。

4. 检测结果的打印与绘制

首先在指定位置打印检测的结果

```
putText(pic_after, shape, {boundRect[i].x, boundRect[i].y - 5},
FONT_HERSHEY_DUPLEX, 0.75, Scalar(0, 69, 255), 2);
```

而指定的位置即在轨迹附近，轨迹的位置我们利用矩形框来定位，在刚刚得到拟合的输出点集后，找到包含了该轨迹的矩形，将是什么图像的文本信息打印在该矩形的上方。

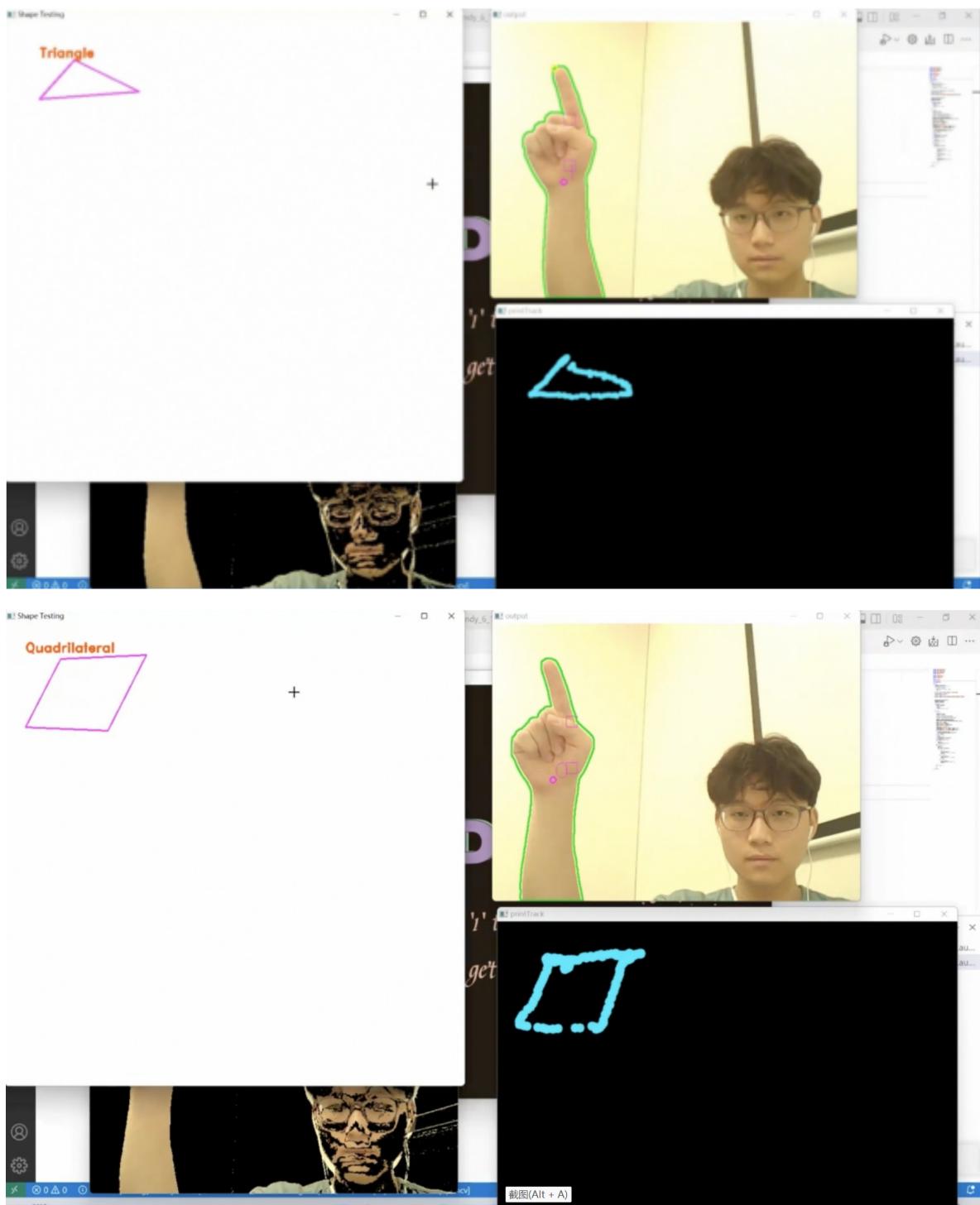
```
vector<Rect> boundRect(contours.size());
boundRect[i] = boundingRect(conPoly[i]);
```

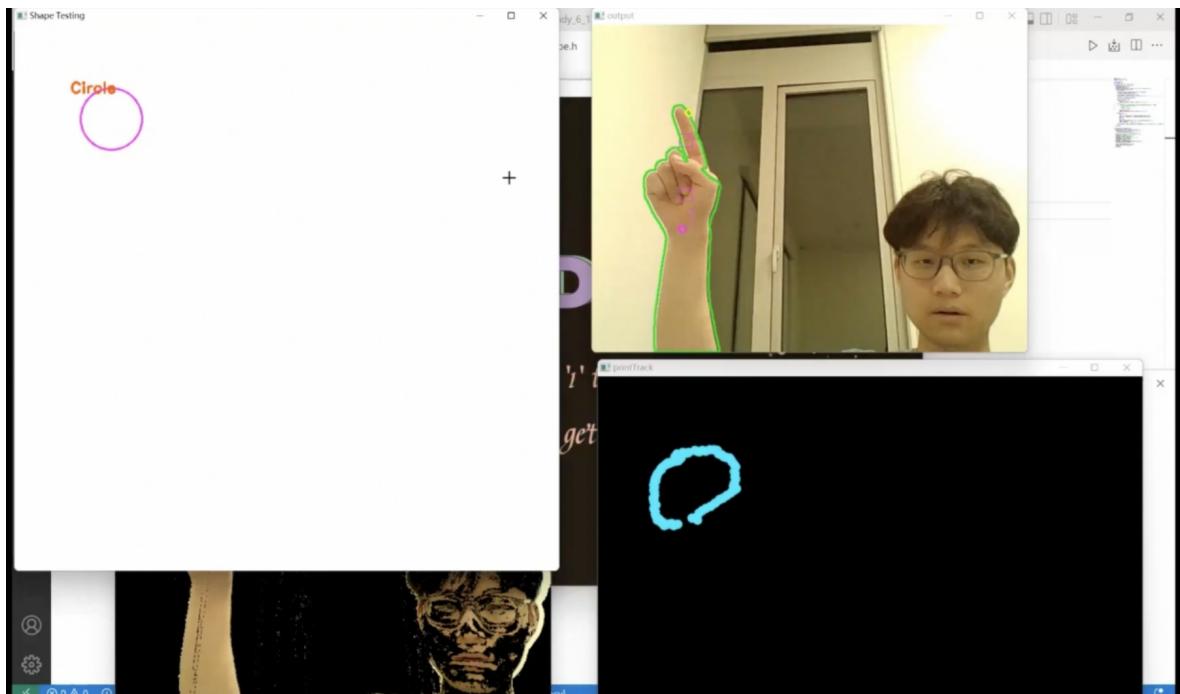
最后则是利用opencv的drawContour函数，可以将拟合得到的节点集连接进行绘制。

```
drawContours(pic_after, conPoly, i, Scalar(255, 0, 255), 2);
```

而在绘制中需要注意的是，该函数只是将拐点进行了连接，对于三角形已经四边形的识别是正确可行的，而当拐点数目较多时，我们将其识别为圆形，则不能仅仅是把拐点进行连接了。在识别为圆形的情况下，我们找到第0个点与第n/2个点，确定圆的中心以及半径，直接画出纠正出的圆形。

5. 结果测试





第六部分：根据轨迹控制鼠标事件

```
GetCursorPos(&Cursor);  
SetCursorPos(Cursor.x + 3, Cursor.y);  
Sleep(10);
```

根据识别到的不同的轨迹，我们让鼠标光标进行不同方向的移动，识别到四边形让光标左移，识别到三角形让光标右移，识别到圆形让光标向下移动。

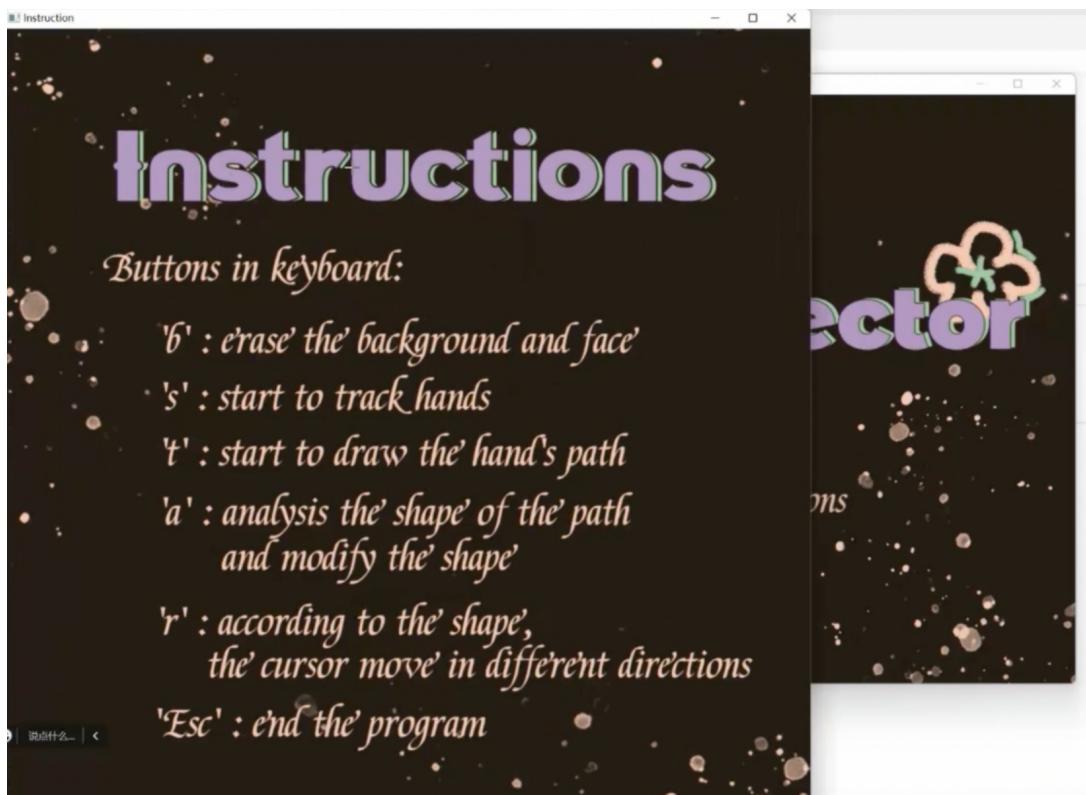
为了让用户更好的观察到光标的移动，我们设置了循环移动。每次移动单位长度后进行一次暂停，使光标的移动更为清晰。

第七部分：特色功能实现

1. 用户图形界面部分

1.1 首页以及引导页面

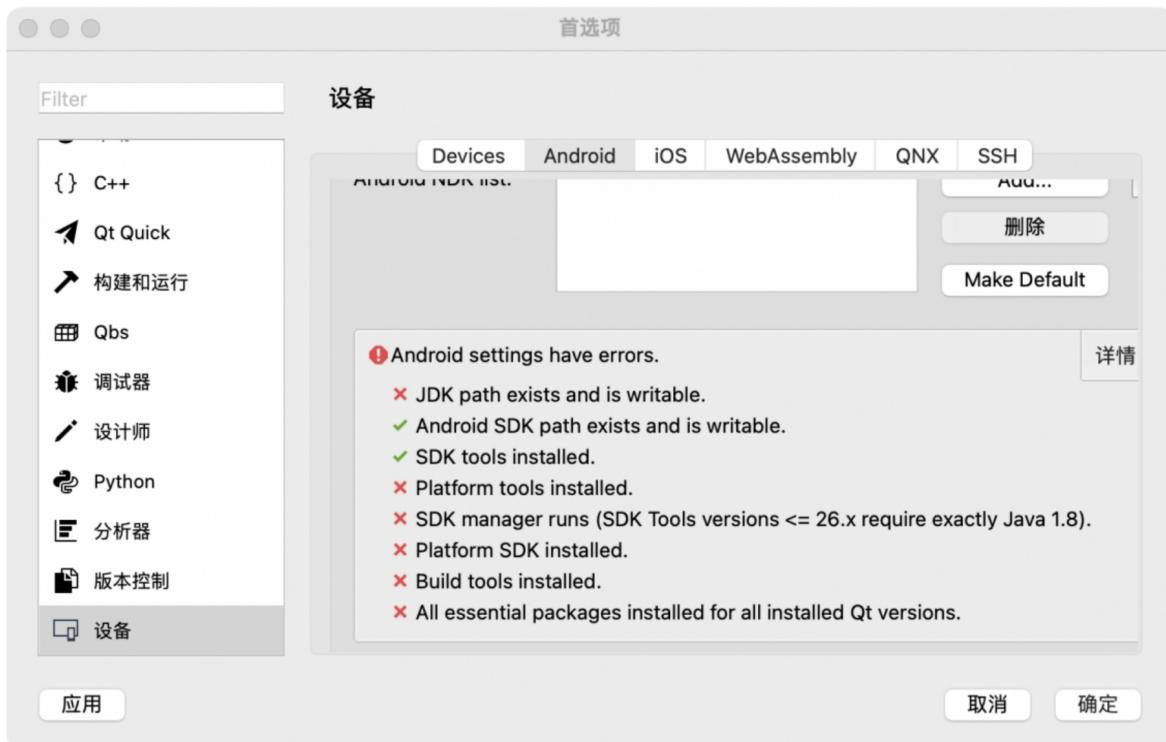
为了更好的用户体验，我们想将这次的项目尽可能的完整，因此在正式进入项目功能之前设计了首页以及操作说明。如下图展示，



遇到的问题及解决：

由于一开始想使用按钮功能，探索很长时间qt的使用，在过程中遇到一系列问题，比如：

1. 安装时编译器的配置问题：



该问题发现可以通过重新建项目工程，修改编译选项解决。

2. 安装好后路径的添加和插件的使用问题：

```
1 #include "mainwindow.h"
2 #include <opencv2/opencv.hpp>    ⚠ In included file: 'arm_neon.h' file not found
3
4 #include <QApplication>
5 #include <QPushButton>
6 #include <QLocale>
```

该问题发现可以通过取消clang插件的运行解决。

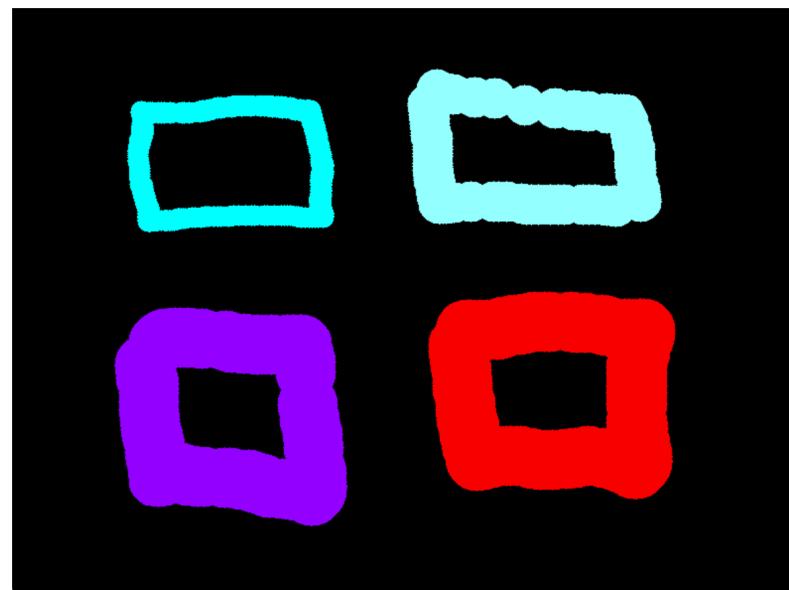
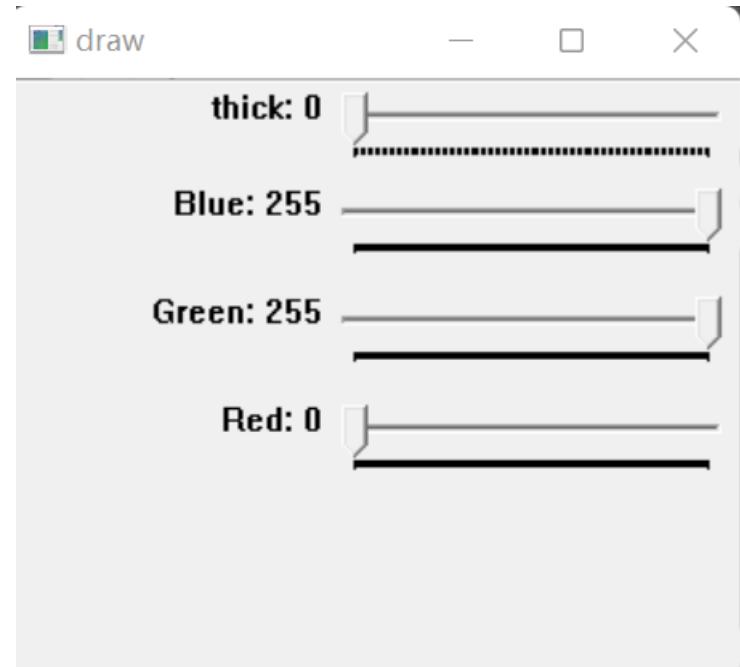
3. 运行时发现的版本问题：

```
⚠ dylib (/opt/homebrew/Cellar/opencv@3/3.4.16_3/lib/libopencv_ximgproc.3.4.dylib) was built for newer macOS version (12.0) than being linked (11.0)
⚠ dylib (/opt/homebrew/Cellar/opencv@3/3.4.16_3/lib/libopencv_ximgproc.dylib) was built for newer macOS version (12.0) than being linked (11.0)
⚠ dylib (/opt/homebrew/Cellar/opencv@3/3.4.16_3/lib/libopencv_xobjdetect.3.4.16.dylib) was built for newer macOS version (12.0) than being linked (11.0)
⚠ dylib (/opt/homebrew/Cellar/opencv@3/3.4.16_3/lib/libopencv_xobjdetect.3.4.dylib) was built for newer macOS version (12.0) than being linked (11.0)
⚠ dylib (/opt/homebrew/Cellar/opencv@3/3.4.16_3/lib/libopencv_xobjdetect.dylib) was built for newer macOS version (12.0) than being linked (11.0)
⚠ dylib (/opt/homebrew/Cellar/opencv@3/3.4.16_3/lib/libopencv_xphoto.3.4.16.dylib) was built for newer macOS version (12.0) than being linked (11.0)
⚠ dylib (/opt/homebrew/Cellar/opencv@3/3.4.16_3/lib/libopencv_xphoto.3.4.dylib) was built for newer macOS version (12.0) than being linked (11.0)
⚠ dylib (/opt/homebrew/Cellar/opencv@3/3.4.16_3/lib/libopencv_xphoto.dylib) was built for newer macOS version (12.0) than being linked (11.0)
⚠ 10 duplicate symbols for architecture arm64
⚠ linker command failed with exit code 1 (use -v to see invocation)
Error while building/deploying project test (kit: Qt 6.3.0 for macOS)
```

而由于时间成本问题，该问题发现之后发现可能需要重新安装，我们考虑直接用opencv已有的功能替代，因此利用了键盘，实现了上面的界面。

1.2 画笔颜色及粗细设置

为增强用户体验，创建可视化窗口调节画笔的颜色与粗细



1.3 摄像头参数设置

通过使用opencv自带的VideoCapture类，获取摄像头参数并进行修改。通过修改摄像头画面的对比度，亮度等参数，可以更好地更灵敏地识别出手，方便之后轨迹的识别与检测。

```
VideoCapture videoCapture(0);  
videoCapture.set(CAP_PROP_SETTINGS, 1);
```



2.手部识别以及手势识别优化

见第二部分，识别手的轮廓：手部识别和手势识别。

3.键盘使能

使用opencv命名空间cv中的waitkey函数，读取键盘输入，从而控制功能的进行。

```
int key = waitKey(1);

if (key == 27) // esc
    break;
else if (key == 'b')
    ...
else if (key == 's')
    ...
else if (key == 't')
{
    ...
}
else if (key == 'a')
{
    ...
}
else if (key == 'r')
{
    ...
}
```

总结与展望

本次project的实现基于本学期对C/C++课程所教授的知识以及我们对OpenCV的了解。由于事件原因以及知识的掌握程度仍有进步的空间，因此我们仅仅使用了关于opencv和c++的一些比较基础的知识，project的很多功能仍可以继续提升。

例如，project对特定物体的识别还不够精确，未来可以使用更高级的算法（包括但不限于人工智能的深度学习理论）来使得对物体识别精确度和准确性的进一步提高；project中对于背景的要求也较高，比如背景在每一次识别之间必须保持静止，在未来我们仍可对这一部分进行更实用的拓展。

经过此次project的学习与完成，我们收获了许多知识和技能，我们也将继续对C/C++程序设计的学习，将其变成日实用的工具。