

CS205 C/C++ Programming - Group-Project II

Project: HCI

Group Members: 肖煜玮(12012902)、杨家鉴(12012711)

Part 1 - Analysis

1. 状态转换机

下面给出整个程序的状态机图：

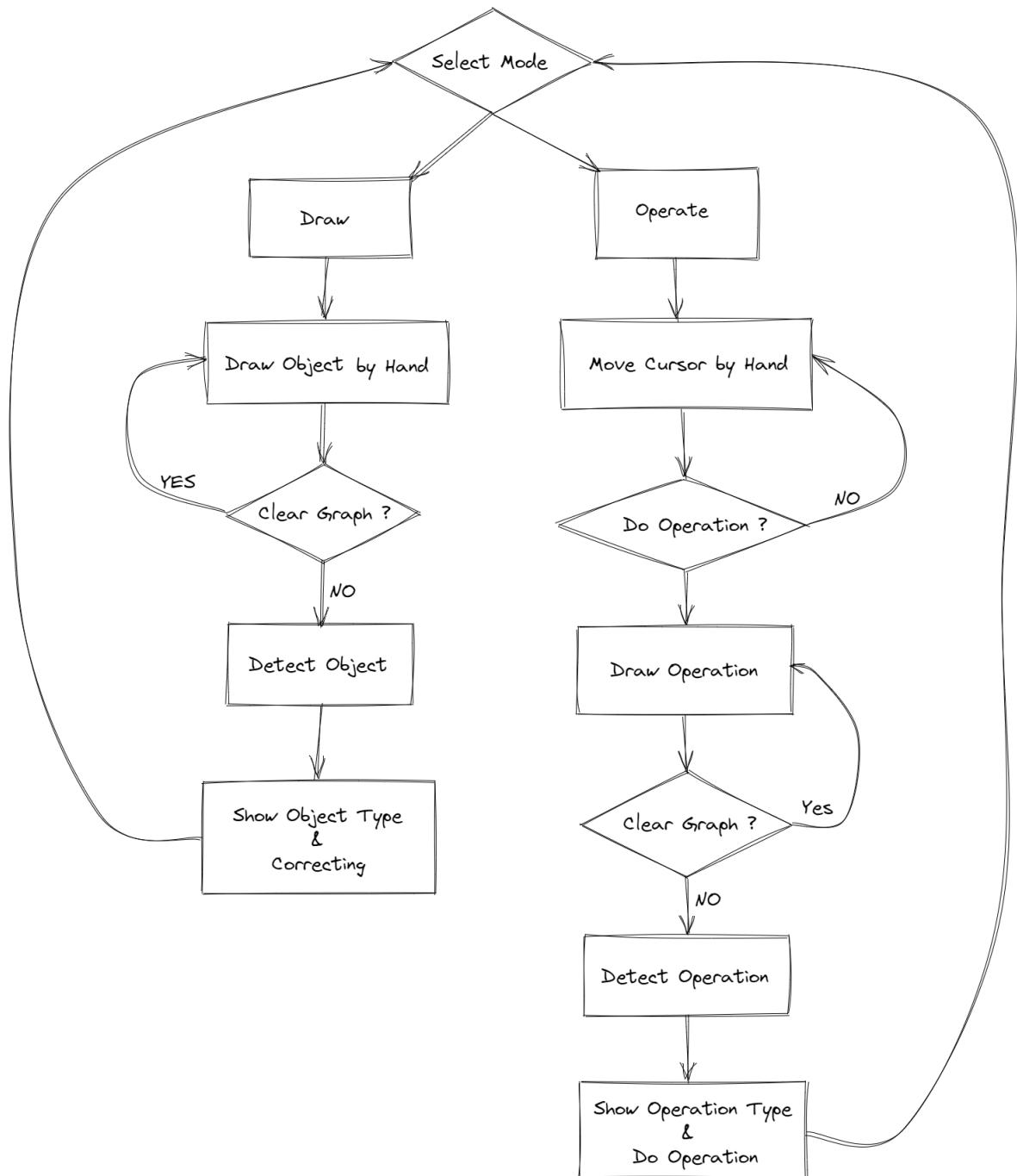


图2 | 状态转换机

2. Image IO (10 points)

(1) Mat类:

OpenCV引入C++接口，提供Mat类用于存储数据，Mat类用来保存矩阵类型的数据信息，包括向量、矩阵、灰度或彩色图像等数据。Mat类分为矩阵头和指向存储数据的矩阵指针两部分。矩阵头中包含矩阵的尺寸、存储方法、地址和引用次数等。利用自动内存管理技术很好的解决了内存自动释放的问题，当变量不再需要时立即释放内存。

```
//创建Mat类
cv::Mat a; //创建一个名为a的矩阵头
a = cv::imread("test.jpg"); //向a中赋值图像数据，矩阵指针指向像素数据
cv::Mat b=a; //复制矩阵头，并命名为b
```

(2) imread()和imwrite():

OpenCV提供了imread函数来读取图像，并提供了3个函数重载。imwrite函数来保存图片。在项目中，当用户按下键盘上'K'时，将画面当前帧保存到桌面。

```
//imread函数原型
Mat imread( const String& filename, int flags )
/*第一个参数 filename: 表示图像的路径。
第二个参数 flags: 表示读取图像的方式。
IMREAD_UNCHANGED = -1, 表示读取原图, 不进行任何改变
IMREAD_GRAYSCALE = 0, 表示以灰度图方式读取原图
IMREAD_COLOR = 1, 表示以RGB方式读取原图
默认不加 flags 的话, 表示不做改变读取原图。*/
//imwrite函数
bool imwrite(const string& filename, InputArray img, const std::vector<int>&
params = std::vector<int>())
/*第一个参数const String& filename表示需要写入的文件名, 必须要加上后缀, 比如“123.png”, 或带文件路径的图像名称。
第二个参数InputArray img表示Mat类型的图像数据。
第三个参数: const std::vector<int>&类型的params, 表示为特定格式保存的参数编码, 它有默认值
std::vector<int>()*/
```

3. Locate Object (20 points) & Track Target (20 points)

(1) 大致思路:

先对原始图像进行中值滤波操作{medianBlur}，将图像的每个像素用邻域(以当前像素为中心的正方形区域)像素的中值代替，达到平滑处理的效果，方便后续对图像进行轮廓检测以及色彩变换。利用{inRange}函数将在两个阈值内的像素值设置为白色(255)，而不是阈值区间内的像素值设置为黑色(0)，更关键的是可以同时针对多通道进行操作。

```

/*
src: 输入要处理的图像，可以为单通道或多通道。
lowerb: 包含下边界的数组或标量。
upperb: 包含上边界数组或标量。
dst: 输出图像，与输入图像src 尺寸相同且为CV_8U 类型。该函数输出的dst是一幅二值化之后的图像。
*/
void inRange(InputArray src, InputArray lowerb,
              InputArray upperb, OutputArray dst);

```

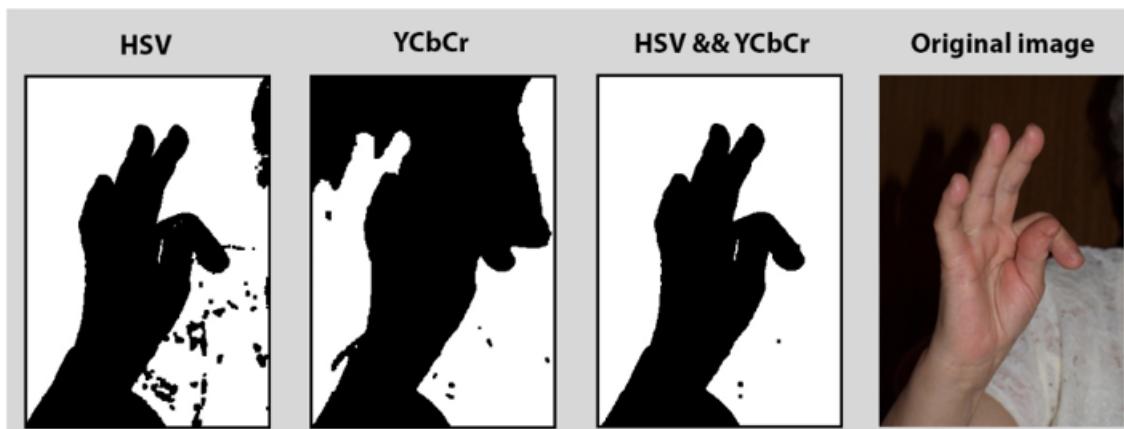
(2) 不同色彩空间的对比:

HSV: RGB通道并不能很好地反映出物体具体的颜色信息，而相对于RGB空间，HSV空间能够非常直观的表达色彩的明暗，色调，以及鲜艳程度，方便进行颜色之间的对比。

YCrCb: 在人脸检测中也常常用到YCrCb空间，因为一般的图像都是基于RGB空间的，在RGB空间里人脸的肤色受亮度影响相当大，所以肤色点很难从非肤色点中分离出来，也就是说在此空间经过处理后，肤色点是离散的点，中间嵌有很多非肤色，这为肤色区域标定(人脸标定、眼睛等)带来了难题。如果把RGB转为YCrCb空间的话，可以忽略Y(亮度)的影响，因为该空间受亮度影响很小，肤色会产生很好的类聚。这样就把三维的空间降为二维的CrCb，肤色点会形成一定得形状。

(3) 优化实现:

将原RGB色彩空间的图片分别转换{cvtColor}到HSV通道和YCrCb通道，再通过bitwise_or () 函数将两个通道的图像进行“或”运算，这样就能将两种色彩空间的优势结合到一起，得到更好的检测效果。



```

//使用HSV色彩空间—便于色彩对比
cvtColor(imgBlur, imgHSV, COLOR_BGR2HSV);
Scalar lower_HSV(hmin, smin, vmin);
Scalar upper_HSV(hmax, smax, vmax);
inRange(imgHSV, lower_HSV, upper_HSV, range_HSV);
erode(range_HSV, range_HSV, kernel);
dilate(range_HSV, range_HSV, kernel);

//用YCrCb空间识别—手和脸都会检测到
cvtColor(imgBlur, imgYCrCb, COLOR_BGR2YCrCb);
Scalar lower_YCrCb(Ymin, CRmin, CBmin);
Scalar upper_YCrCb(Ymax, CRmax, CBmax);
inRange(imgYCrCb, lower_YCrCb, upper_YCrCb, range_YCrCb);
erode(range_YCrCb, range_YCrCb, kernel);
dilate(range_YCrCb, range_YCrCb, kernel);

//图像处理

```

```

bitwise_or(range_HSV, range_YCrCb, img_final);
medianBlur(img_final, img_final, 3);
erode(img_final, img_final, kernel_combine);
dilate(img_final, img_final, kernel_combine);

```

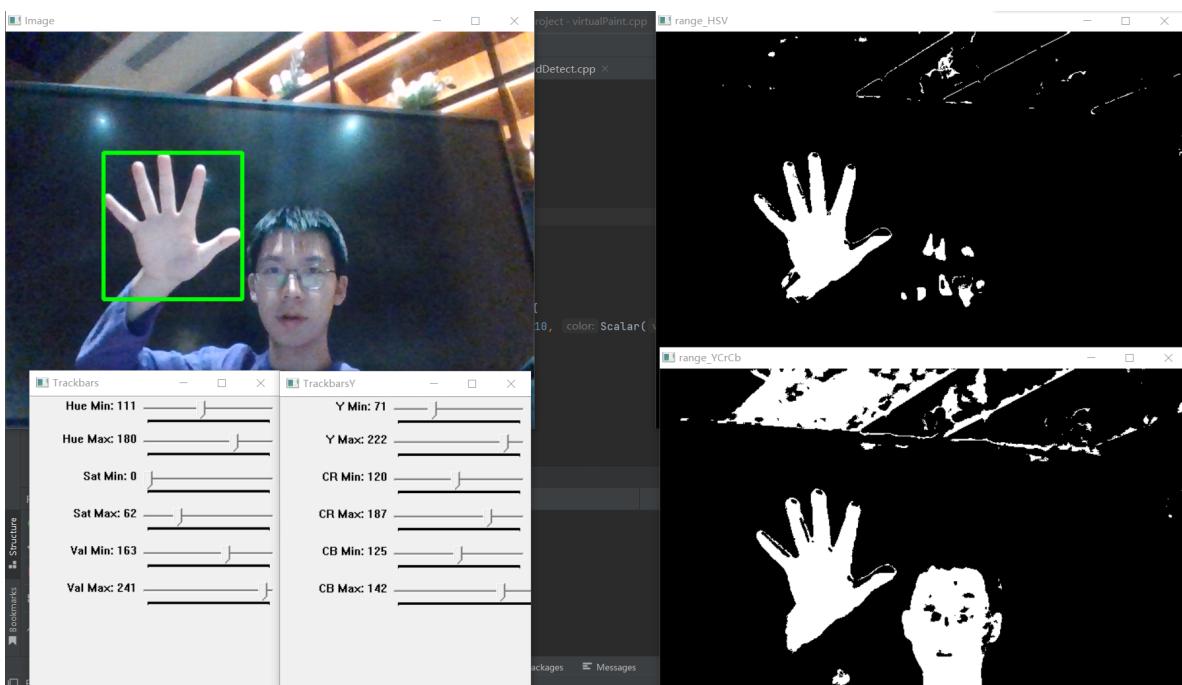


图1 | HSV和YCrCb颜色空间比较

4. Draw Certain Shapes (10 points)

(1) 大致思路

点击start按键后，程序记录绘画起始点，每隔10帧获取一次手部位置（矩形左上角），并push_back进一个存放Point的vector容器。直到检测到回到出发点位置，结束绘画。

(2) 开始和结束的判断——具体实现

在实际开发中，我们遇到了非常多的问题，比如怎么才算开始，怎么算结束，在开始时由于手的抖动造成结束误判等一系列问题。解决办法分别为：见下面注释

```

//初次开始画图，设置起点
//监听鼠标事件判断开始画图;
if (!start_draw) {
    startPoint = point;
    points.push_back(point);
    start_draw = true;
}
//如果在画图过程中回到了起点，则结束画图，清空点集
//当前点的坐标和起点坐标在横纵15像素内时，视为回到起点，画图结束;
//当画图开始后，只有当画满80个点后，才能判断结束，避免开始时抖动误判
else if (((abs(point.x - startPoint.x) < 15) && ((abs(point.y - startPoint.y) < 15))) &&
         points.size() > 80) {
    start_draw = false;
    end_draw = true;
    begin_draw = false;
}

```

```
    }
    else { //将点加入点集
        points.push_back(point);
    }
}
```

5. Detect Shape (10 points)

(1) 大致思路

创建一个角点向量集合{conPoly}，仅存储轮廓中的角点，然后调用{arcLength}函数获取轮廓周长。利用{approxPolyDP}函数生成逼近曲线，其中的角点存在conPoly中，然后对conPoly进行判断，若conPoly中点的数量等于3，则为三角形；为4，则为四边形；为5，则为五边形；若大于5，则为圆。

(2) Douglas-Peucker算法

调用OpenCV库中{approxPolyDP}函数——生成逼近曲线：该函数采用**Douglas-Peucker算法**（也称迭代终点拟合算法）。可以有效减少多边形曲线上点的数量，生成逼近曲线，简化后继操作。

经典的 Douglas-Peucker 算法描述如下：

在曲线首尾两点 A, B 之间连接一条直线 AB，该直线为曲线的弦；

得到曲线上离该直线段距离最大的点 C，计算其与 AB 的距离 d；

比较该距离与预先给定的阈值 threshold 的大小，如果小于 threshold，则该直线段作为曲线的近似，该段曲线处理完毕。

如果距离大于阈值，则用 C 将曲线分为两段 AC 和 BC，并分别对两段曲线进行 1~3 的处理。

当所有曲线都处理完毕时，依次连接各个分割点形成的折线，即可以作为曲线的近似。

```
//函数原型
void approxPolyDP(InputArray curve, OutputArray approxCurve, double epsilon,
bool closed)
/*
curve: 输入的二维点集，可以为 vector 或 Mat 类型。
approxCurve: 多边形逼近的结果，其类型应该和输入的二维点集类型一致。
epsilon: 逼近的精度，设定的原始曲线与近似曲线之间的最大距离，即上文提到的的阈值。
closed: 如果为真，则近似的曲线为封闭曲线（第一个顶点和最后一个顶点相连），否则，近似的曲线不封闭。
*/
```

通过调整epsilon的值，可以有效检测出所画图形的中拐点的个数，最终通过拐点个数来检测图形形状。经过反复调试，最终确定以{0.03*arcLength}作为参数检测的准确度最高！

(3) 实现代码

```
//      通过角点数量识别图形
int objectType;
int objCornerPoints = (int) conPoly[0].size();
if (objCornerPoints == 3) {
    objectType = 3;
} else if (objCornerPoints == 4) {
    objectType = 4;
```

```

} else if (objCornerPoints == 5) {
    objectType = 5;
} else if (objCornerPoints > 5) {
    objectType = 6;
} else {
    objectType = 7;
}
//      operator binding
if (objectType == 3){
    obj = "Triangle";
    operationType = "Click";
}
else if(objectType == 4){
    obj = "Rectangle";
    operationType = "DoubleClick";
}
else if(objectType == 5){
    obj = "Pentagon";
    operationType = "MidClick";
}
else if(objectType == 6){
    obj = "Circle";
    operationType = "RightClick";
}
else{
    obj = "Bad Object";
    operationType = "None";
}

```

6. Shape Modify (10 points)

(1) 三角形的修正:

如果步骤4中检测到的几何图形为多边形{Triangle}, 则直接调用{minEnclosingTriangle}函数获得原始点的最小包裹三角形。

```

/*
points:要被包裹的点集
triangle: 得到的三角形的轮廓
*/
void minEnclosingTriangle(InputArray points,OutputArray triangle);

```

(2) 矩形的修正:

如果步骤4中检测到的几何图形为多边形{Rectangle}, 则直接调用{}函数获得原始点的最小包裹三角形。

(3) 圆的修正:

如果步骤4中检测到的几何图形为**Circle**, 则调用{**minEnclosingCircle**}函数寻找包裹轮廓的最小圆。该函数能得到最小包裹圆的圆心和半径。

```
/*
points: 输入的二维点集, 可以是 vector 或 Mat 类型。
center: 圆的输出圆心。
radius: 圆的输出半径。
*/
void minEnclosingCircle(InputArray points, Point2f& center, float& radius);
```

底层原理:

- 1.遍历所有点, 找出最左边、最右边、最上边、最下边的四个点, 分别用A、B、C、D来表示。
- 2.求出包围这四个点的最小圆C1的圆心和半径;
- 3.第一次迭代 (k=0)。遍历所有点, 检查是否存在点出界, 即不在圆C1界内也不在边界上。如果没有出界点, 则求出的圆就是最终所求。如果有出界点, 转到第四步。
- 4.假设出界点中距离圆C1的圆心最远的点为E, 则依次尝试以下四种组合: (1)A/B/C/E (2)A/B/D/E
(3)A/C/D/E (4)B/C/D/E。求出该组合中四个点的最小包围圆。假设组合(1)A、B、C、E的最小包围圆是C2, 然后检测剩下的点D不在C2内, 比如说不在, 那就算组合 (2), 算出A、B、D、E四个点的最小包围圆C3, 检测剩下的点C, 发现在圆C3内, 那么记录下点E和圆C3的圆心、半径。
- 5.第二次迭代 (k=1)。遍历所有点, 检查是否存在出界点, 即不在圆C3界内也不在边界上。如果没有出界点, 则求出的圆C3就是最终所求。如果有出界点, 则转到下一步。
- 6.假设出界点中距离圆C3的圆心最远的点为F, 则依次尝试以下四种组合: (1)A/B/D/F (2)A/B/E/F
(3)A/D/E/F (4)B/D/E/F (这里已经没有C什么事了, 它已经在上一轮被淘汰了)。依次处理这四个组合, 求出该组合中四个点的最小包围圆。假设组合(1)求出圆C4, 检测点E在圆C4内, 则么记录下点F和圆C4的圆心、半径。
- 7.第三次迭代 (k=2)。接下来重复第5步、第6步, 直到发现遍历完所有点结果都在最新求出的圆界内, 则该圆就是最终所求的圆, 退出迭代。

7. Mouse Control (20 points)

(1) 鼠标追踪

```
void Let_Mouse_Move(int x, int y) {
    double fScreenWidth = ::GetSystemMetrics(SM_CXSCREEN) - 1; // 获取屏幕分辨率宽度
    double fScreenHeight = ::GetSystemMetrics(SM_CYSCREEN) - 1; // 获取屏幕分辨率高度
    double fx = x * (65535.0f / fScreenWidth);
    double fy = y * (65535.0f / fScreenHeight);
    INPUT Input = { 0 };
    Input.type = INPUT_MOUSE;
    Input.mi.dwFlags = MOUSEEVENTF_MOVE | MOUSEEVENTF_ABSOLUTE;
    Input.mi.dx = fx;
    Input.mi.dy = fy;
    SendInput(1, &Input, sizeof(INPUT));
}
```

(2) 鼠标左键、右键、中键、双击

```
void Let_Mouse_Click() {
    INPUT Input = { 0 };
    Input.type = INPUT_MOUSE;
    Input.mi.dwFlags = MOUSEEVENTF_LEFTDOWN | MOUSEEVENTF_LEFTUP;
    SendInput(1, &Input, sizeof(INPUT));
}

void Let_Mouse_Rightclick() {
    INPUT Input = {0};
    Input.type = INPUT_MOUSE;
    Input.mi.dwFlags = MOUSEEVENTF_LEFTDOWN | MOUSEEVENTF_ABSOLUTE;
    SendInput(1, &Input, sizeof(INPUT));
    Input.mi.dwFlags = MOUSEEVENTF_LEFTUP | MOUSEEVENTF_ABSOLUTE;
    SendInput(1, &Input, sizeof(INPUT));
    Input.mi.dwFlags = MOUSEEVENTF_RIGHTDOWN | MOUSEEVENTF_ABSOLUTE;
    SendInput(1, &Input, sizeof(INPUT));
    Input.mi.dwFlags = MOUSEEVENTF_RIGHTUP | MOUSEEVENTF_ABSOLUTE;
    SendInput(1, &Input, sizeof(INPUT));
    cout << "Right click" << endl;
}

void Let_Mouse_MidClick() {
    INPUT Input = {0};
    Input.type = INPUT_MOUSE;
    Input.mi.dwFlags = MOUSEEVENTF_MIDDLEDOWN | MOUSEEVENTF_ABSOLUTE;
    SendInput(1, &Input, sizeof(INPUT));
    Input.mi.dwFlags = MOUSEEVENTF_MIDDLEUP | MOUSEEVENTF_ABSOLUTE;
    SendInput(1, &Input, sizeof(INPUT));
    cout << "Middle click" << endl;
}

void Let_Mouse_DoubleClick() {
    INPUT Input = {0};
    Input.type = INPUT_MOUSE;
    Input.mi.dwFlags = MOUSEEVENTF_LEFTDOWN | MOUSEEVENTF_ABSOLUTE;
    SendInput(1, &Input, sizeof(INPUT));
    Input.mi.dwFlags = MOUSEEVENTF_LEFTUP | MOUSEEVENTF_ABSOLUTE;
    SendInput(1, &Input, sizeof(INPUT));
    Input.mi.dwFlags = MOUSEEVENTF_LEFTDOWN | MOUSEEVENTF_ABSOLUTE;
    SendInput(1, &Input, sizeof(INPUT));
    Input.mi.dwFlags = MOUSEEVENTF_LEFTUP | MOUSEEVENTF_ABSOLUTE;
    SendInput(1, &Input, sizeof(INPUT));
    cout << "Double click" << endl;
}
```

Part 2 - Code

详见项目压缩包中.cpp文件

Part 3 - Result & Verification

1. Image IO

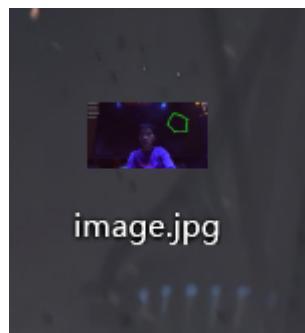


图3 | 储存到桌面的某一帧

2. Locate Object & Track Target



图4 | 识别手掌并跟踪

3. Draw Certain Shapes

PS: 绿色为画图轨迹；紫色为轮廓连线

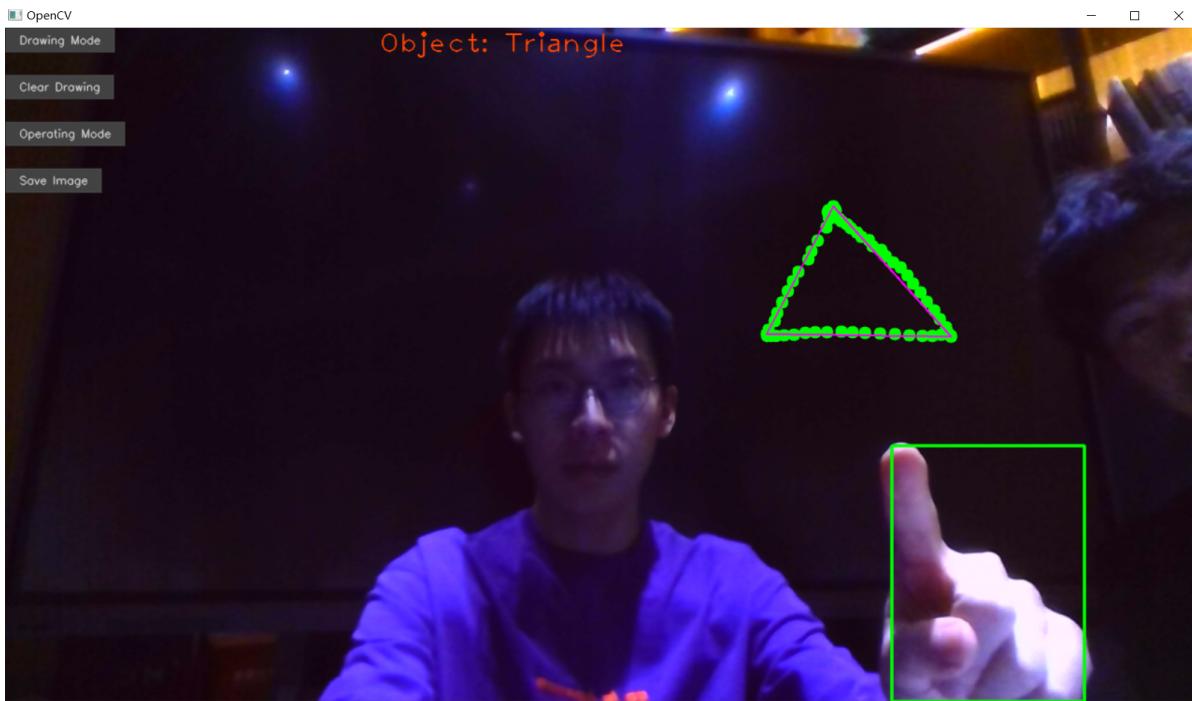


图5 | 三角形检测

4. Detect Shape

PS：在图像正上方显示检测出的图形的名称！



图6 | 五边形检测

5. Shape Modify

PS：**红色**为修正后图形！

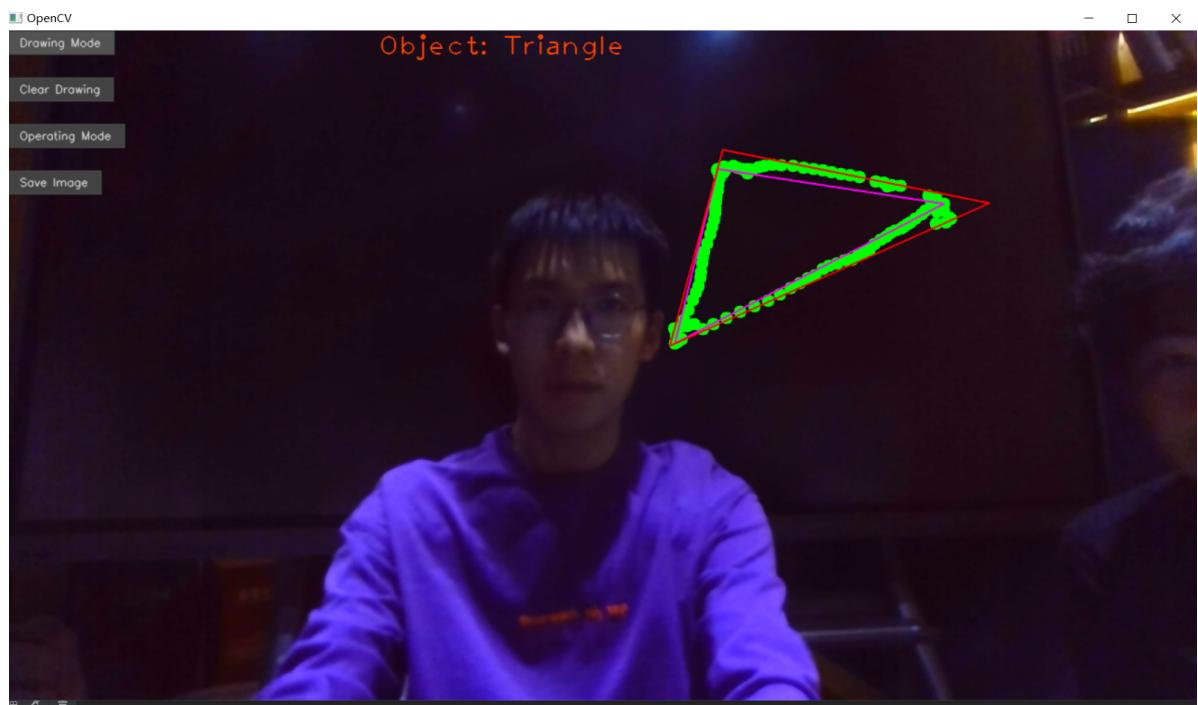


图7 | 三角形的修正

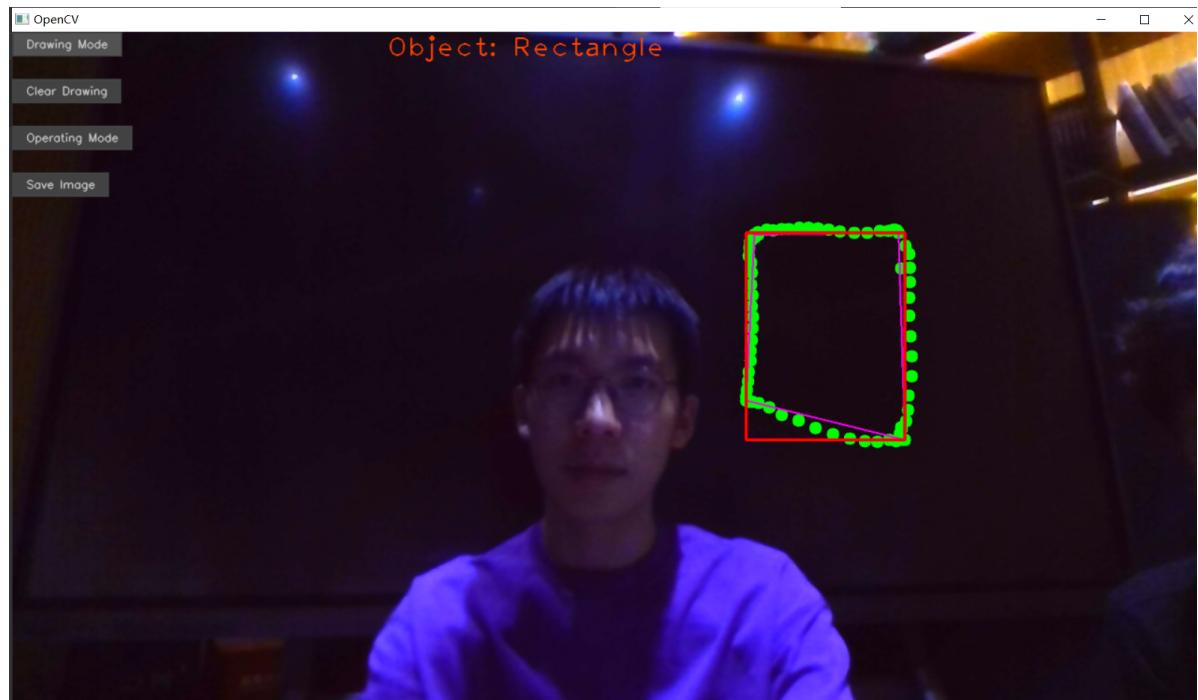


图8 | 矩形的修正

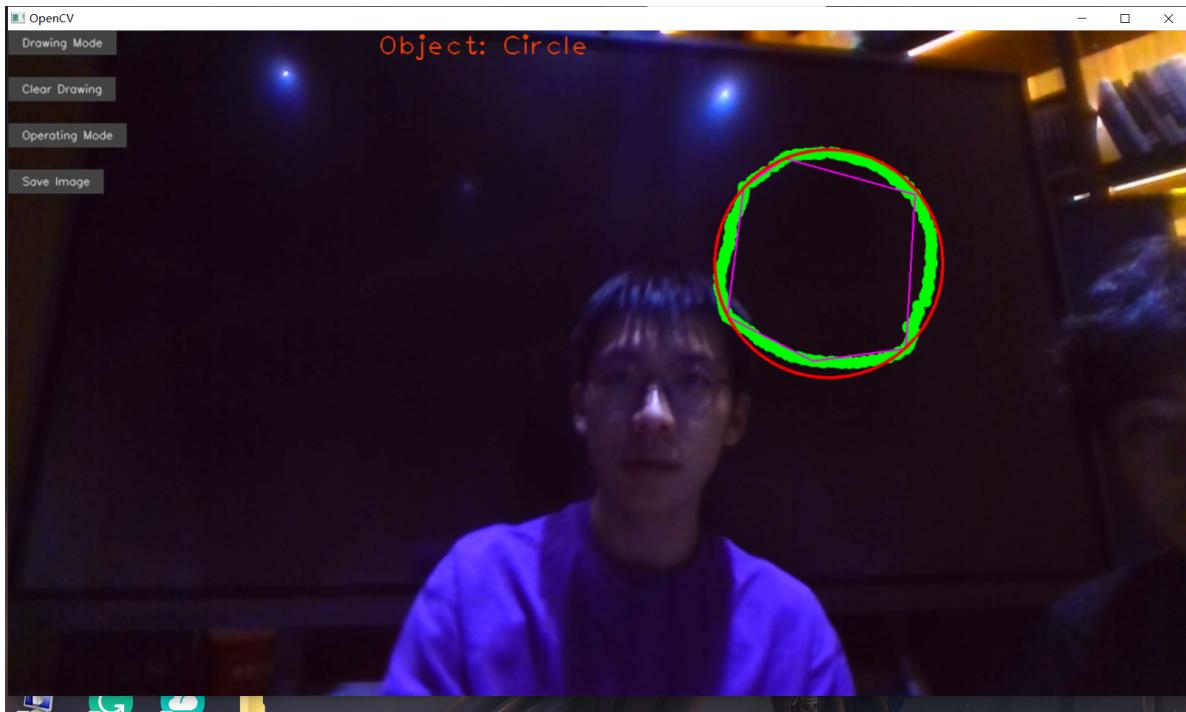


图9 | 圆的修正

Part 4 - Difficulties & Solutions

1. New way to Install OpenCV——利用Vcpkg来管理第三方库

由于尝试了所给文档中的所有安装方法，均已失败告终，于是在自主摸索下，找到了一个似乎更加方便的安装OpenCV的方法。

(1) 配置CMake和winMG

和其他配置方法一样，先要配CMake和winMG的环境变量，这里不多赘述。

(2) 配置Vcpkg

- 使用git命令克隆一个当前版本下来，或者直接下载压缩包。

```
//cmd指令  
git clone https://github.com/microsoft/vcpkg
```

- 编译Vcpkg

Vcpkg大量使用的psl脚本，所以官方强烈推荐使用PowerShell而不时CMD命令行来执行各种操作。尽管在使用的时候兼容CMD，但是在编译这一步，请使用PowerShell。

如果指令执行不成功，可以直接在窗口中打开文件夹双击**bootstrap-vcpkg.bat**

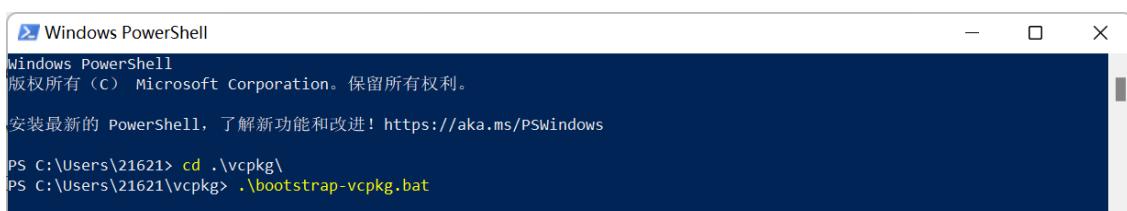


图10 | PowerShell指令

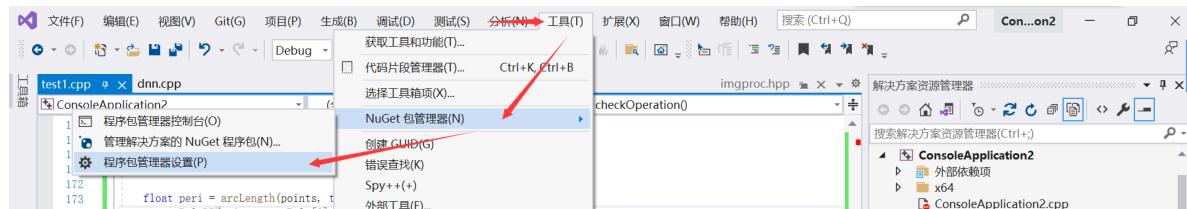
使用Vcpkg安装OpenCV库

```
//cmd指令: 在vcpkg目录下  
.vcpkg.exe install opencv:x64-windows
```

Vcpkg和visual studio的集成

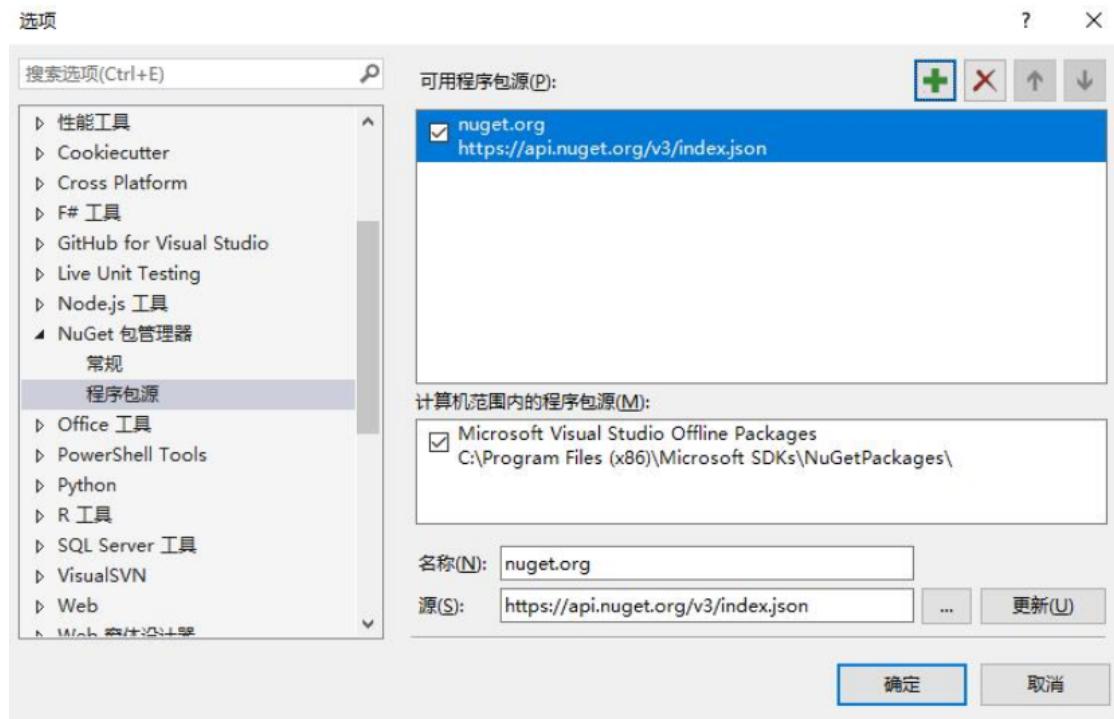
```
//cmd指令: 集成到工程  
.vcpkg integrate project
```

这时候会在“./vcpkg/scripts/buildsystems”目录下，生成nuget配置文件，打开Visual Studio，点击菜单“工具->NuGet包管理器->程序包管理器设置”，进入设置界面，点击“程序包源”。

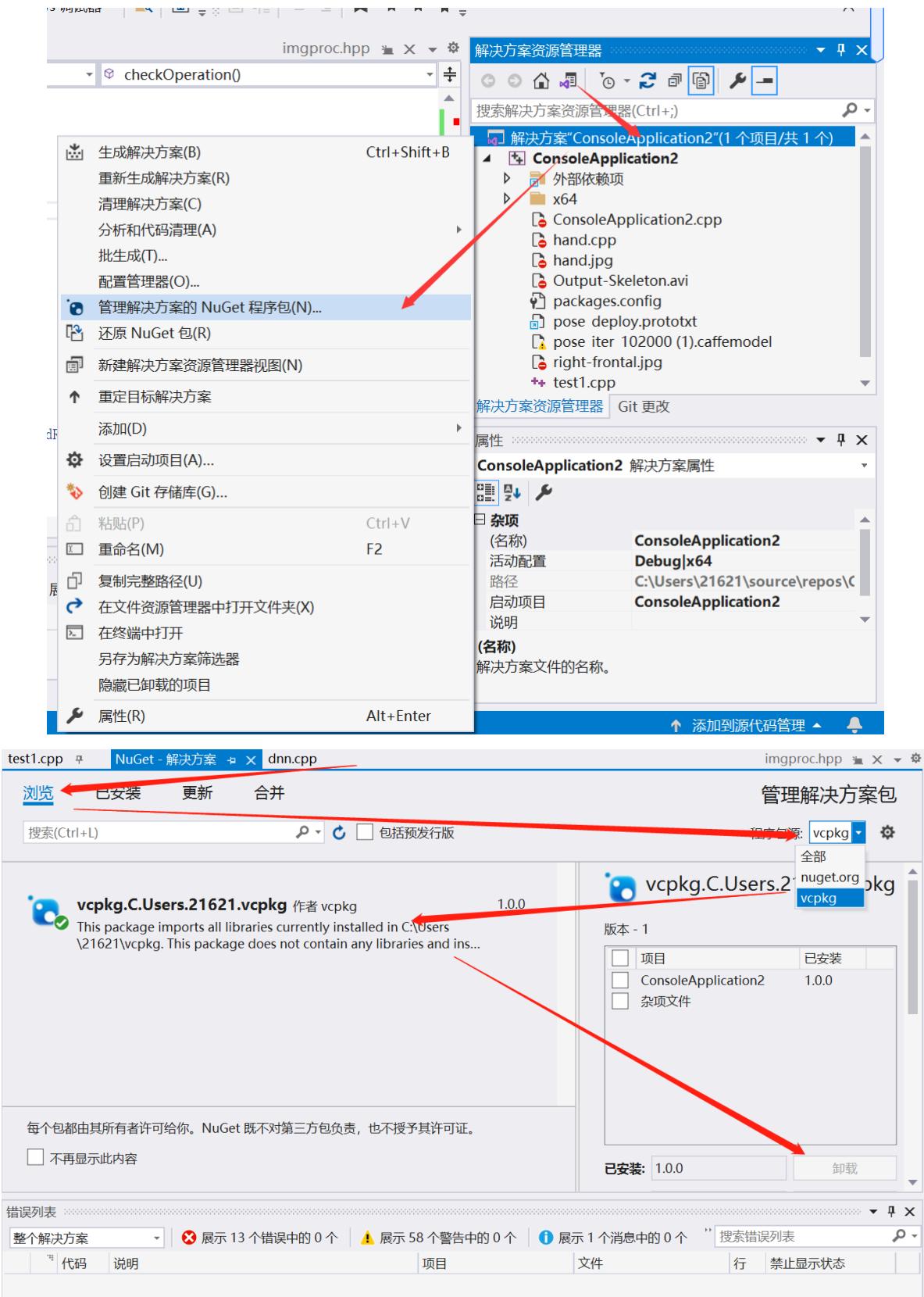


点击“加号”增加一个源。修改源的名字为vcpkg。在“源”的选项中点击右侧的“...”选择vcpkg目录下的“scripts\buildsystems”目录，然后点击右侧的“更新按钮”。点击“确定”，关闭设置对话框。

用Visual Studio 打开一个工程或解决方案。右键点击需要设置的工程，选择“管理NuGet程序包”。在右上角的“程序包源”中选择刚刚设置的“vcpkg”。这样在“浏览”选项卡中就可以看到“vcpkg.H.Repos.vcpkg”。点击最右侧的“安装”。这样就可以集成到某个工程了。



用Visual Studio 打开一个工程或解决方案。右键点击需要设置的工程，选择“管理NuGet程序包”。在右上角的“程序包源”中选择刚刚设置的“vcpkg”。这样在“浏览”选项卡中就可以看到“vcpkg.H.Repos.vcpkg”。点击最右侧的“安装”。这样就可以集成到某个工程了。



2. Few Resources to Refer

由于国内外有关C++开发OpenCV的资料十分有限，导致很多功能的实现有些时候要找好多的资料，甚至有些时候需要查看python的代码，再结合源码才能完成一些相关功能，可以说整个过程十分艰辛，但却十分充实。在完成本次Project的过程中，不仅是完成了一个project，还掌握了OpenCV的基础用法。在项目过程中，既有遇到让人捧腹大笑的BUG，也有人机交互带给我们的成就感，可谓收获颇多。

在学习OpenCV的时候，有参考一个4h的视频教程，值得推荐。[4h上手C++版Opencv哔哩哔哩bilibili](#)

