山东大学　计算机科学与技术　学院

计算机视觉 课程实验报告

学　号　： 201700301042	姓名： 陈佳睿	班级： 17 智能

实验题目：基于直方图的目标跟踪

实现基于直方图的目标跟踪：已知第 t 帧目标的包围矩形，计算第 t+1 帧目标的矩形区域。

选择适当的测试视频进行测试：给定第 1 帧目标的矩形框，计算其它帧中的目标区域。

实验过程中遇到和解决的问题：

（记录实验过程中遇到的问题，以及解决过程和实验结果。可以适当配以关键代码辅助说明，但不要大段贴代码。）

首先，实验的答题思路是对于视频的每一帧图像都进行直方图计算

然后通过定义不同帧的差距计算方式来进行图像矩形框的跟随

先看对于图像的直方图计算实现

opencv 中对于直方图的计算实现函数接口详解：

C++: void calcHist(const Mat* images, int nimages, const int* channels, InputArray mask, OutputArray hist, int dims, const int* histSize, const float** ranges, bool uniform=true, bool accumulate=false)

参数详解：

onst Mat* images：输入图像

int nimages：输入图像的个数

const int* channels：需要统计直方图的第几通道

InputArray mask：掩膜，，计算掩膜内的直方图 ...Mat()

OutputArray hist:输出的直方图数组

int dims：需要统计直方图通道的个数

const int* histSize：指的是直方图分成多少个区间，就是 bin的个数

const float** ranges： 统计像素值得区间

bool uniform=true::是否对得到的直方图数组进行归一化处理

bool accumulate=false：在多个图像时，是否累计计算像素值得个数

图像直方图将统计结果分布于一系列预定义的 bin 中

最难理解的是第 6，7，8 个参数 dims、histSize 和 ranges

channels — 维度通道序列

第一幅图像的通道标号从 0~image[0].channels( )–1。Image[0]表示图像数组中的第一幅图像，

channels（）表示该图像的通道数量。

同理，图像阵列中的第二幅图像，通道标号从 image[0].channerls( )开始，到

image[1].channels( )−1 为止；

第三、四幅图像的通道标号顺序依此类推；也就是说图像阵列中的所有图像的通道根据图像排列顺序，排成一个通道队列。

至此，基本搞懂了 opencv 对于直方图计算的函数封装逻辑，开始考虑视频读入问题。

1.先通过使用 calHist 函数实现对于视频中的每一帧图像计算直方图，方便后续功能实现的调用

初始化在 calHist 函数中会用到的变量，用于存放视频中图像的信息

包括了 RGB 图像的通道数，对于每一个维度直方图 bin 的数量，每一个维度数值的取值范围

以及对于得到的直方图数组是否进行归一化处理等参数

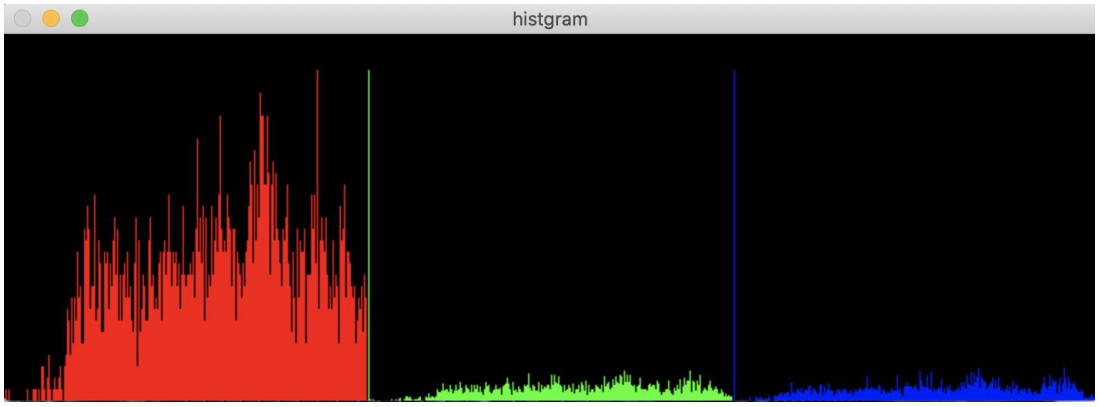
```
//计算图像的直方图(红色通道部分)
cv::calcHist(&rectImage, nimages, channels: &channels[0], mask: cv::Mat(), outputHist_red, dims, histSize: &histSize[0], ranges: &ranges[0], uni, accum);
//计算图像的直方图(绿色通道部分)
cv::calcHist(&rectImage, nimages, channels: &channels[1], mask: cv::Mat(), outputHist_green, dims, histSize: &histSize[1], ranges: &ranges[1], uni, accum);
//计算图像的直方图(蓝色通道部分)
cv::calcHist(&rectImage, nimages, channels: &channels[2], mask: cv::Mat(), outputHist_blue, dims, histSize: &histSize[2], ranges: &ranges[2], uni, accum);
```

直接使用统计出来的数值在窗口中进行打印展示，有时候如果某个数据点的值很高，他就占满

整个打印的窗口，所以通过使用 minmaxLoc 函数，得到某帧图像中像素的最大值和最小值

调节参数，使得最后打印在窗口中的像素值在纵轴上依然保持比窗口的 height 要小

最后在实际应用中，手动选择的兴趣点图像　　并展示三通道直方图



2.接下来使用 VideoCapture 类新建一个对象 cap 来对视频进行逐帧读取

在 getObject()函数中通过 setMouseCallback()函数动态回调

```
void Mouse_areaSelection(int event, int x, int y, int flags, void *ustc);
```

函数来定义选择感兴趣区域的鼠标事件

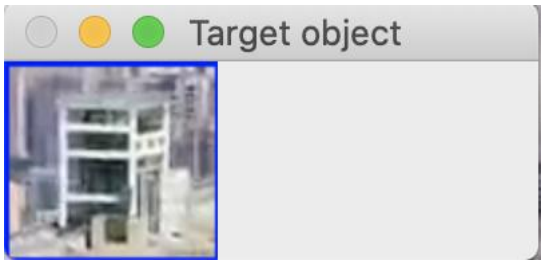
对于 Mouse_areaSelection 函数

定义 bool leftbuttom_downflag　作为标志来定义鼠标左键是否按下

按下为 true　并在该值为 true 的时候　定义矩形框开始的左上角点的坐标 startPoint(x,y)

在鼠标移动过程中动态更新矩形框的右下角终止位置 endPoint(x,y)

并在鼠标左键按下并移动的过程中，不再用 cap 读入新的图像
并在锁定的该框选帧上根据 startPoint 和 endPoint 来动态更新矩形框的形状
并最终在鼠标左键抬起，即标识符 bool leftbuttom_downflag = false 的时候
使用 Rect()函数绘制框定的矩形区域
赋值给全局的 Mat 对象 objectImage 用于记录感兴趣的图像特征并用于后续的图像相似度计算
传递一个全局 int 类型标识符 finish 来记录是否已经完成感兴趣区域的框选



这样 imshow("interImage",image);
在每次运行程序的时候 调用 getObject()函数可以得到框选区域 并展示

这样我们就得到了 target object 图像的宽高
开始编写 tracing 函数用于根据我们在第 t 帧框选图像来计算出在 t+1 帧中该框选物体应该在图像中的大体位置
为了简化计算，我只是选取我框定图像周围的像素进行搜索和直方图对比计算
移动的单位用框定图像的 width 和 height 来衡量
但是考虑到主题兴趣点的景物可能在图像中的尺寸占比会比较大
所以需要进行边界控制，对于超出每一帧图像的长宽的值约束在图像内部

```
if((preStart.x+width)<image.cols)
    preEnd.x = preStart.x + width;
else
    preEnd.x = image.cols-1;
if((preStart.y+height)<image.rows)
    preEnd.y = preStart.y+height;
else
    preEnd.y = image.rows-1;
```

这里开始涉及对于框选图像和检索区域图像相似度的计算，根据设置的相似度衡量的阈值
来判定是否在这一帧的图像上框选出这个区域作为预测
所以使用 opencv 封装好的 compareHist 函数进行基于直方图的图像相似度计算

操作步骤：

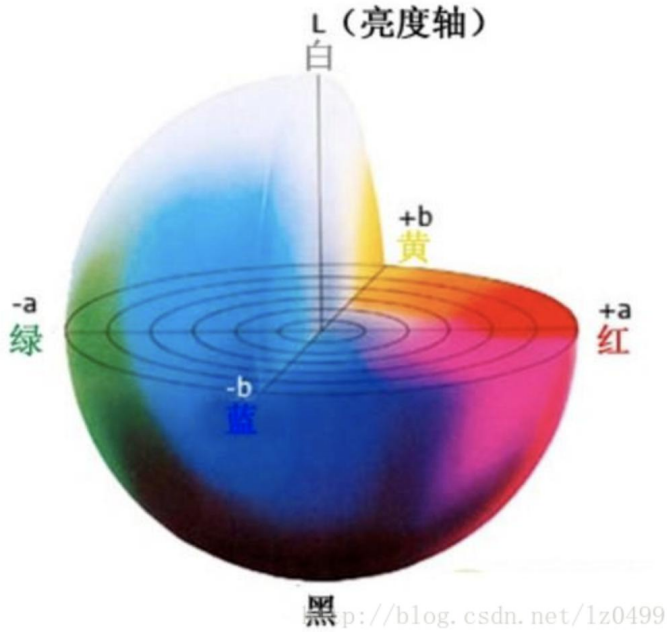
1. 载入图像(灰度图或者彩色图)，并使其大小一致；
2. 若为彩色图，增进行颜色空间变换，从RGB转换到HSV，若为灰度图则无需变换；
3. 若为灰度图，直接计算其直方图，并进行直方图归一化；
4. 若为彩色图，则计算其彩色直方图，并进行彩色直方图归一化；
5. 使用相似度公式，如相关系数、卡方、相交或巴氏距离，计算出相似度值。

以上是计算图像相似度的一般步骤
因为我的测试视频都为彩色视频 所以按照上面的流程是要把图像转换为 HSV 空间表示
但是我没明白为什么一定要在计算相似度的时候把原图像的 RGB 转换到 HSV
然后我做了一次对比实验
发现对于同样的视频，对于同样的区域进行框选
使用 RGB 和 HSV 之间存在着极大的表现差异
不进行转换，在 RGB 下进行框选之后，对于阈值参数进行多次调校
视频没能在框选的相邻区域找到对于设定阈值比较接近的区域并框选显示
而转换为 HSV 之后，即使在较小的区域上，也能在较为合理的范围内找到相似度较高的区域
（也有可能是我实验做错了）
对于此种现象我能找到比较合理的解释是：

RGB 比较颜色之间的相似度时,存在很大的问题,不建议直接使用,因为往往一个通道的一点改变,会导致最后融合在一起的颜色发生巨大变化,而如果三个通道的同时改变,却只会使最后的明暗发生变化,色调并不会产生巨大变化.

继续查，发现还有一种不常用的颜色空间的表示方法 LAB

Lab 是一种不常用的色彩空间。它是在 1931 年国际照明委员会（CIE）制定的颜色度量国际标准的基础上建立起来的。1976 年，经修改后被正式命名为 CIELab。它是一种设备无关的颜色系统，也是一种基于生理特征的颜色系统。这也就意味着，它是用数字化的方法来描述人的视觉感应。Lab 颜色空间中的 L 分量用于表示像素的亮度，取值范围是[0,100],表示从纯黑到纯白；a 表示从红色到绿色的范围，取值范围是[127,-128]；b 表示从黄色到蓝色的范围，取值范围是[127,-128]。下图所示为 Lab 颜色空间的图示；



2.RGB转Lab颜色空间

RGB颜色空间不能直接转换为Lab颜色空间，需要借助XYZ颜色空间，把RGB颜色空间转换到XYZ颜色空间，之后再把XYZ颜色空间转换到Lab颜色空间。

RGB与XYZ颜色空间有如下关系：

$$\begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = \begin{bmatrix} 0.412453 & 0.357580 & 0.180423 \\ 0.212671 & 0.715160 & 0.072169 \\ 0.019334 & 0.119193 & 0.950227 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix} \quad (1)$$

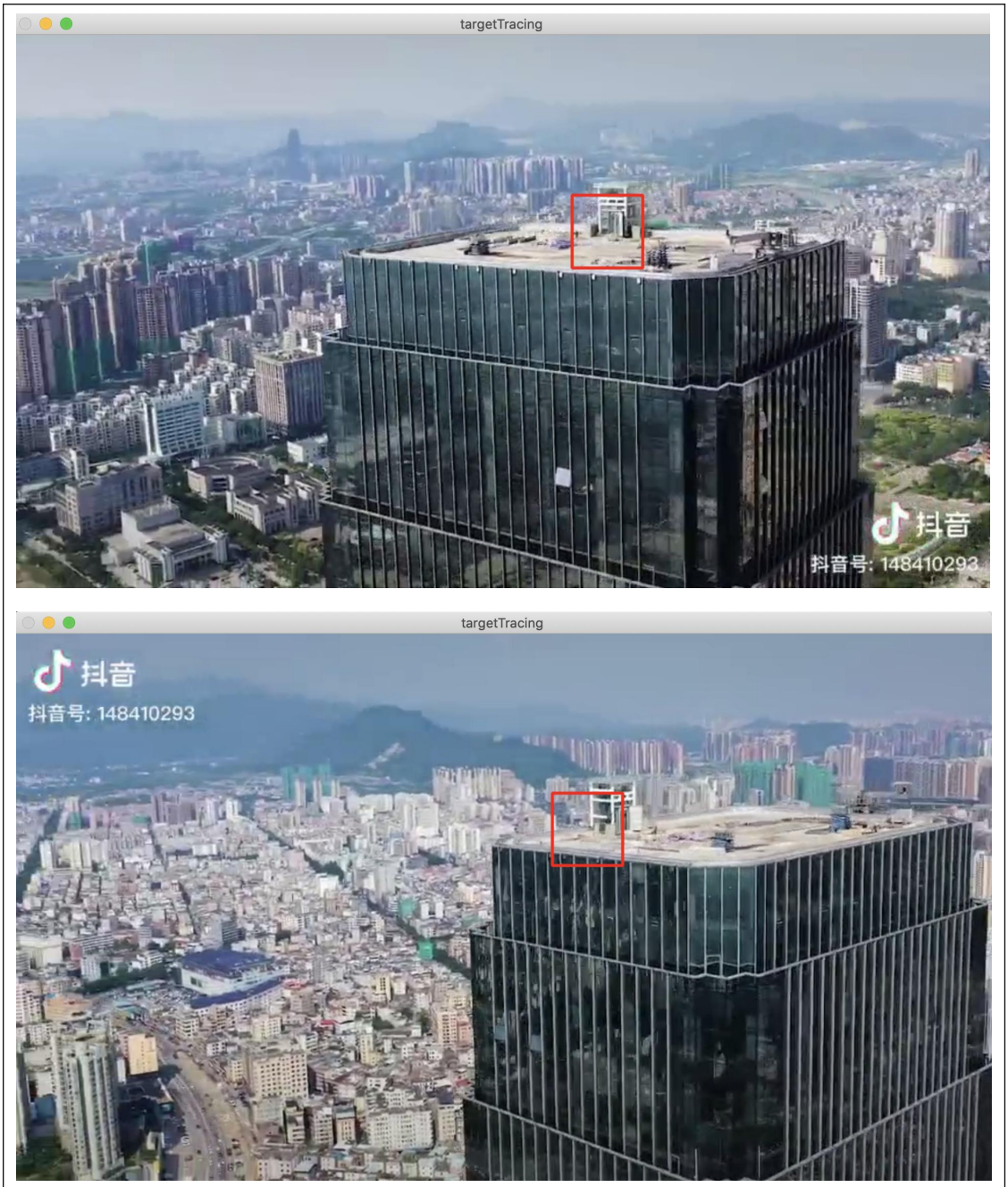
$$\begin{bmatrix} R \\ G \\ B \end{bmatrix} = \begin{bmatrix} 3.240479 & -1.537150 & -0.498535 \\ -0.969256 & 1.875992 & 0.041556 \\ 0.055648 & -0.204043 & 1.057311 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} \quad (2)$$

仔细观察式（1），其中 X = 0.412453 * R + 0.412453 *G+ 0.412453B ；各系数相加之和为0.950456，非常接近于1，我们知道R/G/B的取值范围为[0,255]，如果系数和等于1，则X的取值范围也必然在[0,255]之间，因此我们可以考虑等比修改各系数，使其之和等于1，这样就做到了XYZ和RGB在同等范围的映射。这也就是为什么代码里X,Y,Z会分别除以0.950456、1.0、1.088754。

$$\begin{aligned} L^* &= 116 f(Y/Y_n) - 16 \\ a^* &= 500[f(X/X_n) - f(Y/Y_n)] \\ b^* &= 200[f(Y/Y_n) - f(Z/Z_n)] \end{aligned}$$

$$f(t) = \begin{cases} t^{1/3} & \text{if } t > (\frac{6}{29})^3 \\ \frac{1}{3}\left(\frac{29}{6}\right)^2 t + \frac{4}{29} & \text{otherwise} \end{cases}$$

但是如果只是在 opencv 程序中使用的话
只需要改 cvtColor 的第三个参数即可
相比 HSV 颜色空间在兴趣点环绕视频中的表现
LAB 颜色空间在主体景物的观景角度发生变化时仍然能够较好地框住
最终的结果展示：


targetTracing
抖音
抖音号: 148410293

targetTracing
抖音
抖音号: 148410293

结果分析与体会：
这次的实验整体来说还是比较困难
但是得益于 opencv 库函数的强大
很多繁琐的算法实现都不需要自己重新再去从头实现了
极大提高了开发的效率
Ps：使用直方图为基础的物体追踪方法给我的感觉效果还在可以接受的范围内但是对于在视频中移动速度较高，或者说角度变换导致的图像特征改变仍然表现的比较差。