山东大学 计算机科学与技术 学院

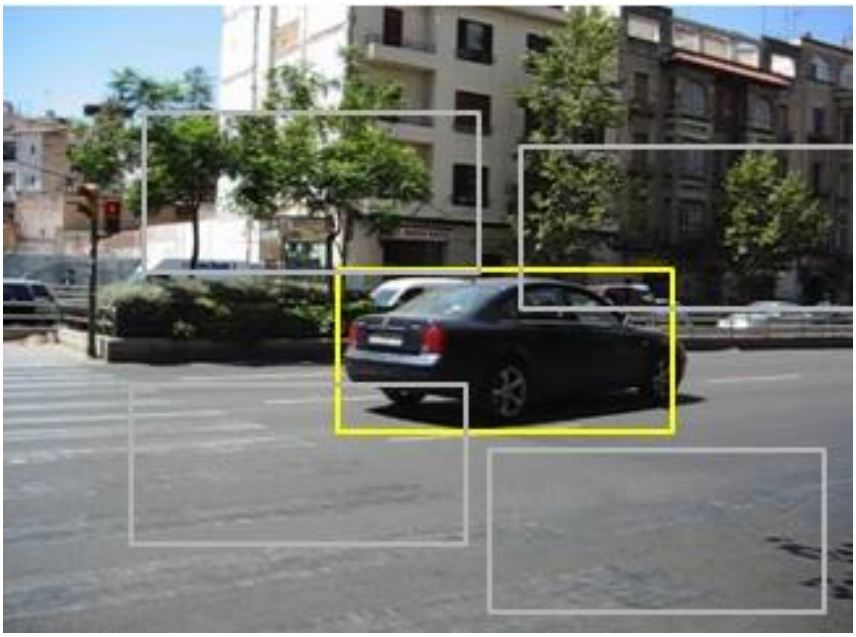
计算机视觉 课程实验报告

学号：201822130233	姓名： 李云龙	
实验题目：图像统计特征		

实验过程中遇到和解决的问题：

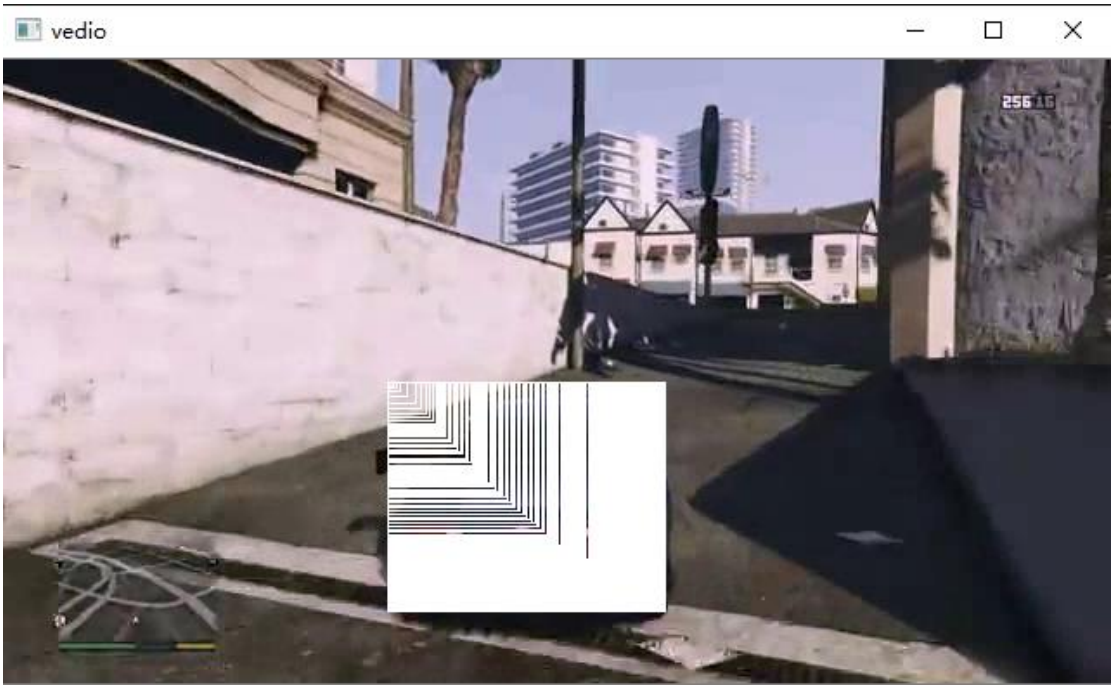
（记录实验过程中遇到的问题，以及解决过程和实验结果。可以适当配以关键代码辅助说明，但不要大段贴代码。）

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。



第 1 步：设置鼠标事件，打开视频播放，手动划出目标区域。由于是在视频里实现目标跟踪，初始的目标框需要用户手动标明而不是参数给定，因此用 VideoCapture 播放视频，并提前绑定鼠标事件，当左键按下时，视频暂停，鼠标移动时显示目标框，最后松开鼠标后，根据开始点与结束点的位置划出目标区域。由于开始画框后要保持画面静止，视频不再输入到图像中，此时如果直接对图像画框，那么在鼠标移动过程中会留下多个矩形框的痕迹，因此选择在图像的克隆体上画，画的过程中每当鼠标移动要画新框时，克隆体克隆自原图像，矩形框就会消失。

直接在原图上画框：



用克隆的临时图像画框：

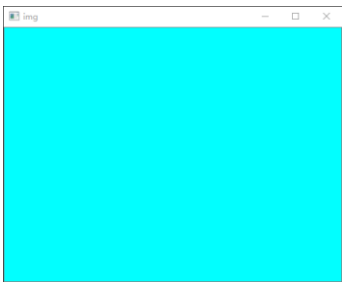
```
if (event == EVENT_MOUSEMOVE && canDraw)
{
    endPoint = Point(x, y);
    roi = Rect(startPoint, endPoint);
    copyImage = image.clone();  // 在原图上画会留下痕迹，在临时的复制品上画框
    rectangle(copyImage, roi, Scalar(255, 255, 255));
    imshow("vedio", copyImage);
}
```

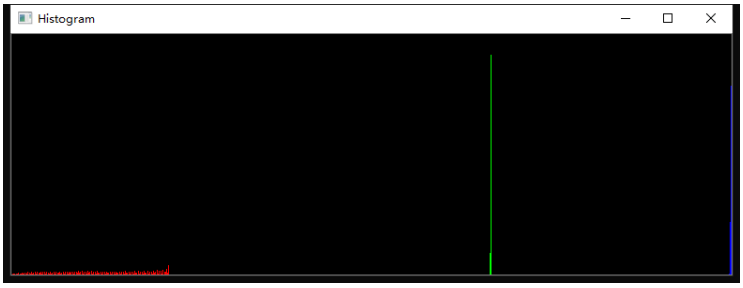
第 2 步：计算目标区域的直方图并显示。由于是基于直方图的目标跟踪，因此在之后的帧中要根据直方图的相似程度判断某区域是否是目标区域。用之前实现的图像通道分离分别获取 RGB 通道，对每个通道计算其直方图，最后汇总在一个数组里，此时数组里存的仅是像素数目，并没有归一化。画直方图时先创建宽 256*3 高 256 的画板，先将数组元素归一化至 0 到 1 的小数，然后缩放至 0 到 256 并画到画板上。

```
// 计算单通道的直方图
void calc_hist(uchar *data, int width, int height, int step, int H[256])
{
    memset(H, 0, sizeof(H[0]) * 256);
    uchar *row = data;
    for (int y = 0; y < height; ++y, row += step)
    {
        for (int x = 0; x < width; ++x)
        {
            H[row[x]]++;
        }
    }
}
```

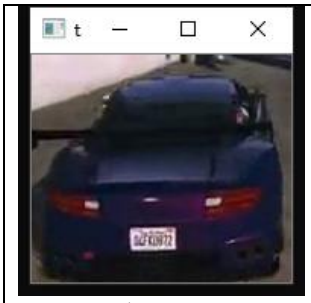
```
// 计算图像的直方图
void calc_hist(const Mat& img, int hist[256 * 3])
{
    for (int ch = 0; ch < 3; ++ch)  // B G R
    {
        Mat channelImage(img.size(), CV_8UC1);
        getChannel(img.data, img.cols, img.rows, (int)img.step, 3, channelImage.data, (int)channelImage.step, ch);
        calc_hist(channelImage.data, channelImage.cols, channelImage.rows, (int)channelImage.step, hist + 256 * (2 - ch));
    }
}
```

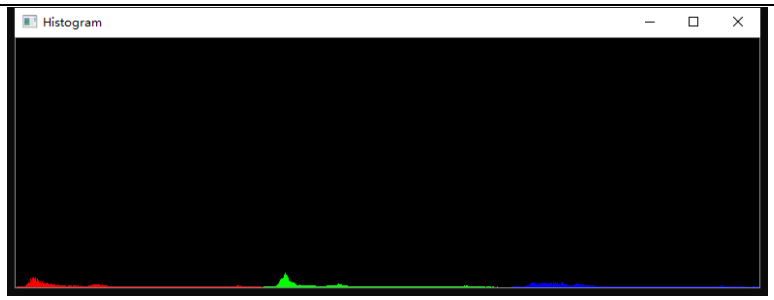
直方图测试：





目标区域的直方图：

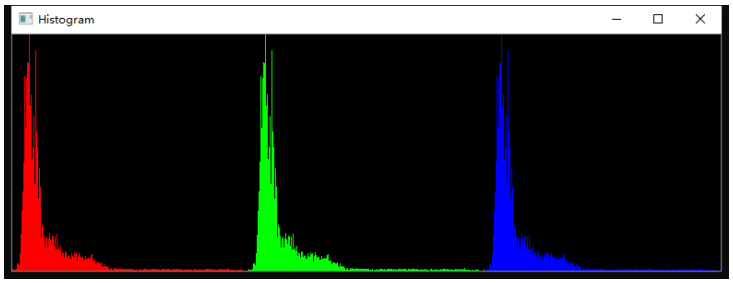




固定的坐标比例足以展示直方图，但是有时会不太美观，可以通过缩放比例使直方图更清楚一些，不再进行归一化除以总像素数，而是除以直方图数组中的最大值。

```
// 画直方图
void draw_hist(const Mat& img, int hist[256 * 3])
{
    Mat hist_picture(256, 256 * 3, CV_8UC3, Scalar(0, 0, 0));
    int n = img.rows * img.cols;
    int max_hist = 0;
    for (int i = 0; i < 256 * 3; ++i) max_hist = max(max_hist, hist[i]);
    Scalar color[] = { Scalar(0, 0, 255) , Scalar(0, 255, 0) , Scalar(255, 0, 0) };
    for (int i = 0; i < 256; ++i)
    {
        for (int j = 0; j < 3; ++j)
        {
            double value = 1.0 * hist[i] / max_hist;//double value = 1.0 * hist[i] / n;
            line(hist_picture, Point(i + 256 * j, 256), Point(i + 256 * j, (1 - value) * 256), color[j]);
        }
    }
    imshow("Histogram", hist_picture);
}
```

缩放后的直方图：



第 3 步：在目标区域的附近遍历，寻找与目标区域的直方图最接近的区域并移动到该区域实现追踪。由于已经获得了目标区域的直方图，因此在之后的每帧中只需要找一个区域，要求其直方图与目标区域直方图最相似。为了减少计算量，只在这一帧的目标区域的附近遍历寻找，将寻找范围设置为宽高均 2 倍于目标区域，同时设置遍历时的间隔为 10 像素，遍历并计算直方图再作比较，这里简单地使用了归一化后的 l2-distance（为了减少计算量就不开方了，反正也是找最小的）。遍历结束后，如果结果的偏差不大，就更新目标区域为新的准目标区域。

```
// 计算直方图间的差异
double compare_hist(const int hist1[256 * 3], const int hist2[256 * 3])
{
    int n = targetImage.rows * targetImage.cols;
    double diff = 0.0;
    for (int i = 0; i < 256 * 3; ++i)
    {
        diff += (hist1[i] - hist2[i]) * (hist1[i] - hist2[i]) * 1.0;
    }
    return diff / (n * n); //sqrt(diff);
}
```

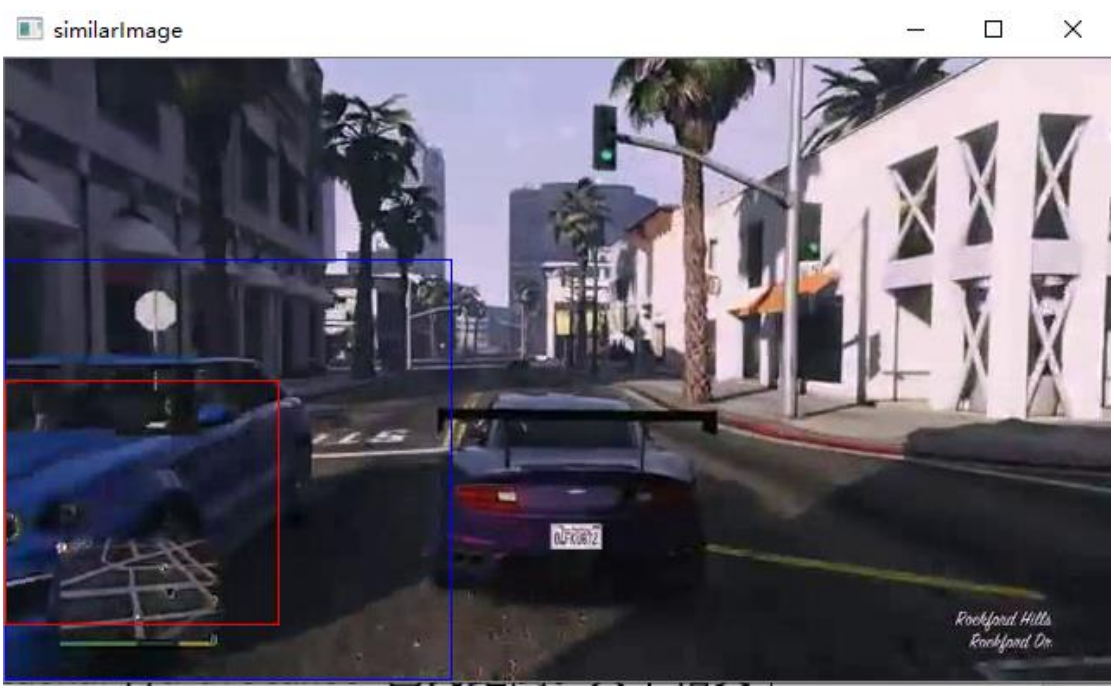
实验时发现视频中的阴影对跟踪的影响很大，在晴朗的时候选定目标区域，在之后的阴影地带就很可能跟丢，而在阴影时选好区域，大部分时间里跟踪效果还可以。蓝色边框是搜查范围，红色框为跟踪区域。

RGB 与 HSV 颜色空间不同，对于某些数值的度量方式的特殊对待也可能造成结果的差异，另外 OpenCV 的 API 中还提供了其他的选项来计算两个直方图间的相似度。尝试使用 HSV 颜色空间，并用 OpenCV 的计算直方图相似度函数的不同方法。

```
// 尝试转换为HSV后的效果
void calc_hist(const Mat& img, Mat& HSVHist)
{
    Mat HSVImage;
    int channels[] = { 0, 1 };
    int histSize[] = { 30, 32 };  // 将色调量化为30级，将饱和度量化为32级
    float HRanges[] = { 0, 180 };  // hue varies from 0 to 179
    float SRanges[] = { 0, 256 };  // 饱和度从0（黑白）到255（纯光谱色）不等
    const float *ranges[] = { HRanges, SRanges };
    cvtColor(img, HSVImage, COLOR_BGR2HSV);  // 转换到HSV
    calcHist(&HSVImage, 1, channels, Mat(), HSVHist, 2, histSize, ranges, true, false);
    normalize(HSVHist, HSVHist, 0, 1, NORM_MINMAX);  // 归一化
}
```

```
// 使用OpenCV的函数，尝试不同的度量方法
double compare_hist(const Mat hist1, const Mat hist2)
{
    return compareHist(hist1, hist2, 3);
}
```

Opencv 提供的比较方法有四种：Correlation 相关性比较，Chi-Square 卡方比较，Intersection 交集法，Bhattacharyya distance 巴氏距离。其中相关性方法范围为 0 到 1，1 为最好匹配，卡方法和 Bhattacharyya 距离法是值为 0 最好，而交集法为值越大越好。实验中发现使用巴氏距离来度量相似度或相关性时，光照情况对其影响很大，如果初始的目标图像没有选好，后面可能跟踪到别的地方（实验中有几次跟到别的汽车上了）。



结果分析与体会：

根据目标图像的直方图可以近似的实现目标追踪

第 1 步：设置鼠标事件，打开视频播放，手动划出目标区域

第 2 步：计算目标区域的直方图并显示

第 3 步：在目标区域的附近遍历，寻找与目标区域的直方图最接近的区域并移动到该区域实现追踪

视频中的阴影对跟踪的影响很大，在晴朗的时候选定目标区域，在之后的阴影地带就很可能跟丢，而在阴影时选好区域，大部分时间里跟踪效果还可以。