

# 10.1 图像拼接

- 基于OpenCV实现图像拼接，可以对两张或更多 的输入图像，将图像对齐后拼接成一张全景图。（ 相关函数： `findHomography`, `warpPerspective` ）
- 效果图：



# 10.1 实验步骤过程

- 1、对每幅图进行特征点提取
- 2、对特征点进行匹配，并进行筛选
- 3、进行图像配准
- 4、把图像拷贝到另一幅图像的特定位置，裁剪尺寸处理
- 5、对重叠边界处理优化，图像融合（去裂缝处理）

## 测试拼接所用的原图片：



Pic01.jpg



Pic02.jpg



Pic03.jpg



Pic04.jpg



Pic05.jpg

# 一、特征点提取

- 提取特征点有sift、surf、harris角点、ORB等常用方法
- 考虑到精确度、稳定性和速度，本实验中采用了SIFT和SURF的方法
- SURF精确度和稳定性不及SIFT，但是其综合能力还是优越一些，在速度方面有了明显的提高（速度是SIFT的3倍）
- 注：opencv stitch选择的特征检测方式第一选择是SURF, 第二选择才是ORB

## 二、获取匹配点与筛选

- 1、获取：

使用opencv的 `FlannBasedMatcher`

（更快找到最近邻近似匹配，不尝试所有可能的匹配，不一定最佳，保证相对好的匹配和速度，`BFMatcher`）

然后调用 `KNNMatch`，可设置  $K = 2$ ，即对每个匹配返回两个最近邻描述符，仅当第一个匹配与第二个匹配之间的距离足够小时，才认为这是一个匹配。得到所有匹配点。

- 2、筛选：（自定义`goodMatch`+二次匹配）

（1）只保留匹配点中的两点距离较小的：

`matchePoints[i][0].distance < 0.4 * matchePoints[i][1].distance`

（2）进行二次匹配：`img1->img2 → goodmatch → img2->img1`

得到更多好的匹配点（参考opencv自带的拼接算法`stitch`）

## • 进行特征点的检测和匹配点筛选后的结果：

左图为将Pic02.jpg和Pic03.jpg合并得到dst01.jpg时打印的信息，有图将Pic01.jpg和dst01.jpg合并拼接的信息

```
C:\Users\wangbinKF\Desktop\vsproject\cv10\x64\Debug\...
全部匹配点数目：342
1->2 good 匹配点数目：26
1->2 & 2->1 good 匹配点数目：33
变换矩阵为：
[0.5160805325845348, 0.03243005491217496, 160.8980848377823;
-0.194186386036559, 0.8393518004047872, 29.34015263209948;
-0.001146012366782238, 2.027951028609315e-05, 1]

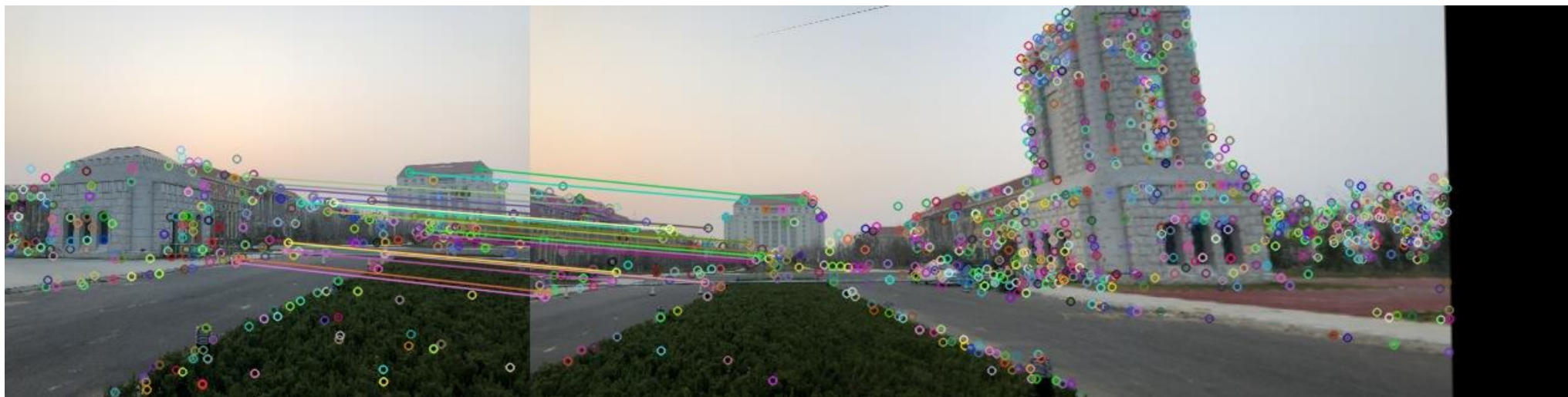
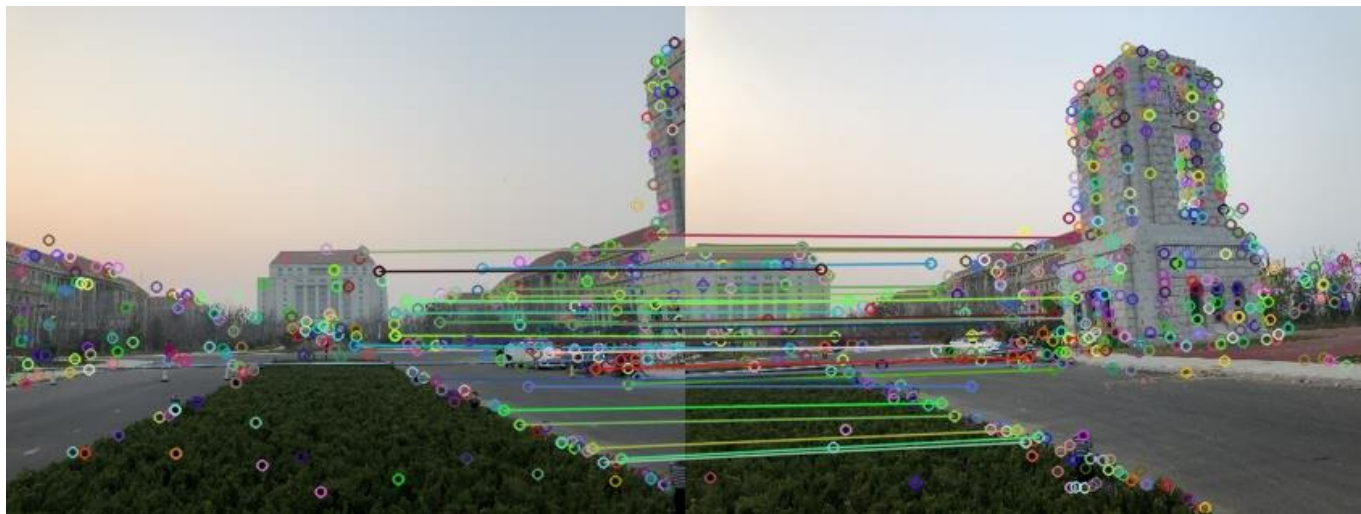
列向量 V2: [0;
0;
1]
列向量 V1: [160.8980848377823;
29.34015263209948;
1]
左上:[160.898, 29.3402]
左下:[170.056, 292.7]
右上:[733.34, -101.875]
右下:[743.953, 406.718]
请按任意键继续. . .
```

```
C:\Users\wangbinKF\Desktop\vsproject\cv10\x64\Debug\cv10.exe
全部匹配点数目：319
1->2 good 匹配点数目：18
1->2 & 2->1 good 匹配点数目：22
变换矩阵为：
[0.5362152912474305, 0.1182380495320776, 160.1027844658315;
-0.2045715906517961, 0.9256557621679746, 1.625782123402213;
-0.001204561045583402, 0.0002690585756571133, 0.9999999999999999]

列向量 V2: [0;
0;
1]
列向量 V1: [160.1027844658315;
1.625782123402213;
0.9999999999999999]
左上:[160.103, 1.62578]
左下:[181.993, 271.085]
右上:[-39633.1, 11060.4]
右下:[9334.84, 1748.83]
请按任意键继续. . .
```



- 进行特征点的检测和匹配点筛选后的结果:

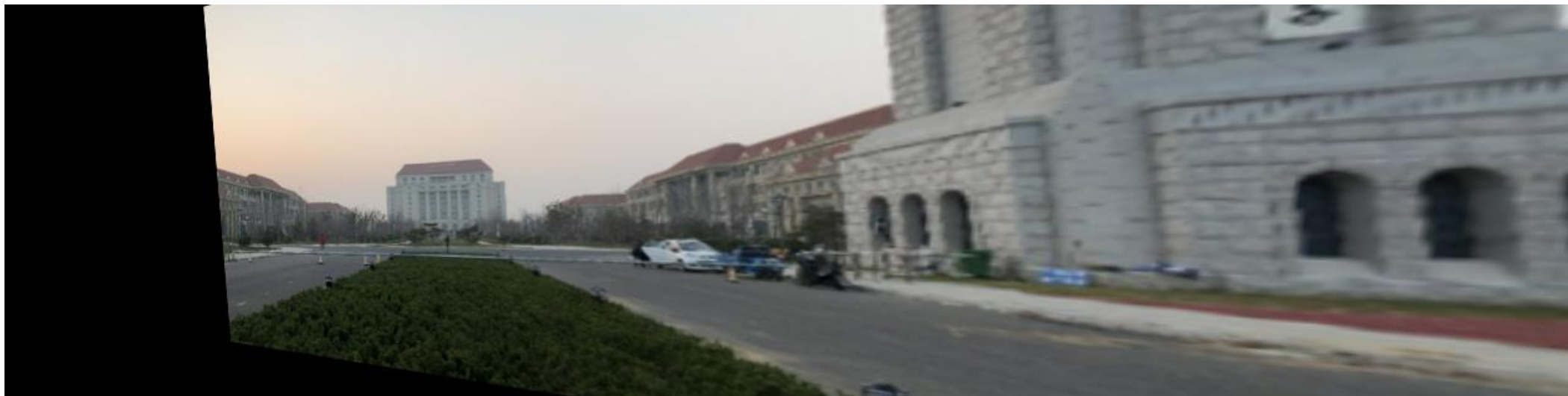
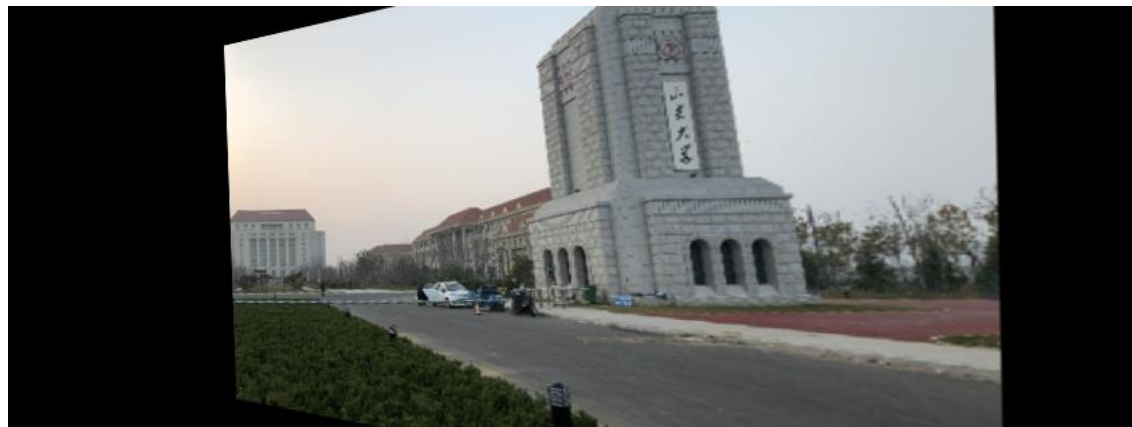


### 三、进行图像配准与拷贝

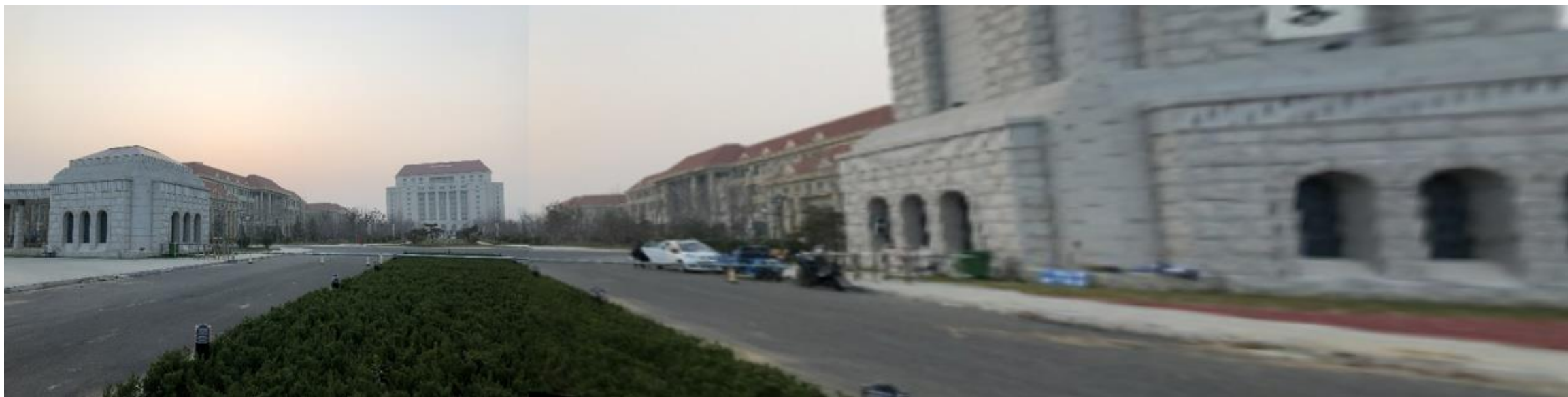
- 先利用 `findHomography` 函数得到投影映射矩阵  
(也可以使用`getPerspectiveTransform`方法获得透视变换矩阵, 不过要求只能有4个点, 效果稍差)
- 计算配准图的四个顶点坐标
- 调用 `warpPerspective` 变换另一幅图像, 最后设置好尺寸, 使用`copyTo`完成拼接



# 1、转换后tran.jpg



## 2、初步拼接后图片



## 四、边界处理优化

- 优化两图的连接处，使得拼接自然：

- 策略：（修改alpha通道值）

>>如果遇到图像trans中无像素的黑点，则完全拷贝img1中的数据

>>否则将img1中像素的权重，设置为与当前处理点距重叠区域左边界的距离成正比的大小



## 1、最终效果图一：



## opencv stitch效果图：



## 2、最终效果图二：



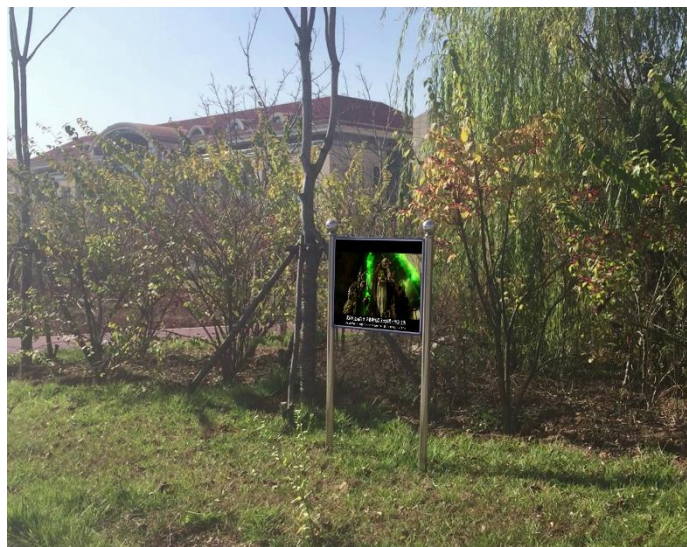
## opencv stitch效果图：





## 10.2 基于平面跟踪的AR

- 对包含一个平面目标的视频，跟踪由用户指定的平面上的一个四边形区域，并替换区域内的视频内容。
  - 被跟踪的四边形区域由用户在第1帧指定，或者基于给定的模板进行识别；
  - 请优化跟踪的稳定性和速度





## 10.2 实验内容步骤

- 打开视频对第一帧进行处理，交互确定四边形的初始点
- 框定进行特征检测与匹配的局部区域，加速匹配计算变换矩阵
- 对视频流每帧进行处理计算出矩阵后得到新的四个顶点坐标（`perspectiveTransform`）再次利用`findHomography`和`warpPerspective`将视频进行嵌入

# 一、优化及一些问题

- 1、框定局部区域可以加速计算并派出一些干扰，来求解 homograph，区域太小会导致计算出的匹配点太少，误差较大，造成晃动

- 优化策略：框定具有明显特征的区域，进一步优化筛选匹配点，自定义的 good match 可以调高阈值，适当增加匹配点的数量

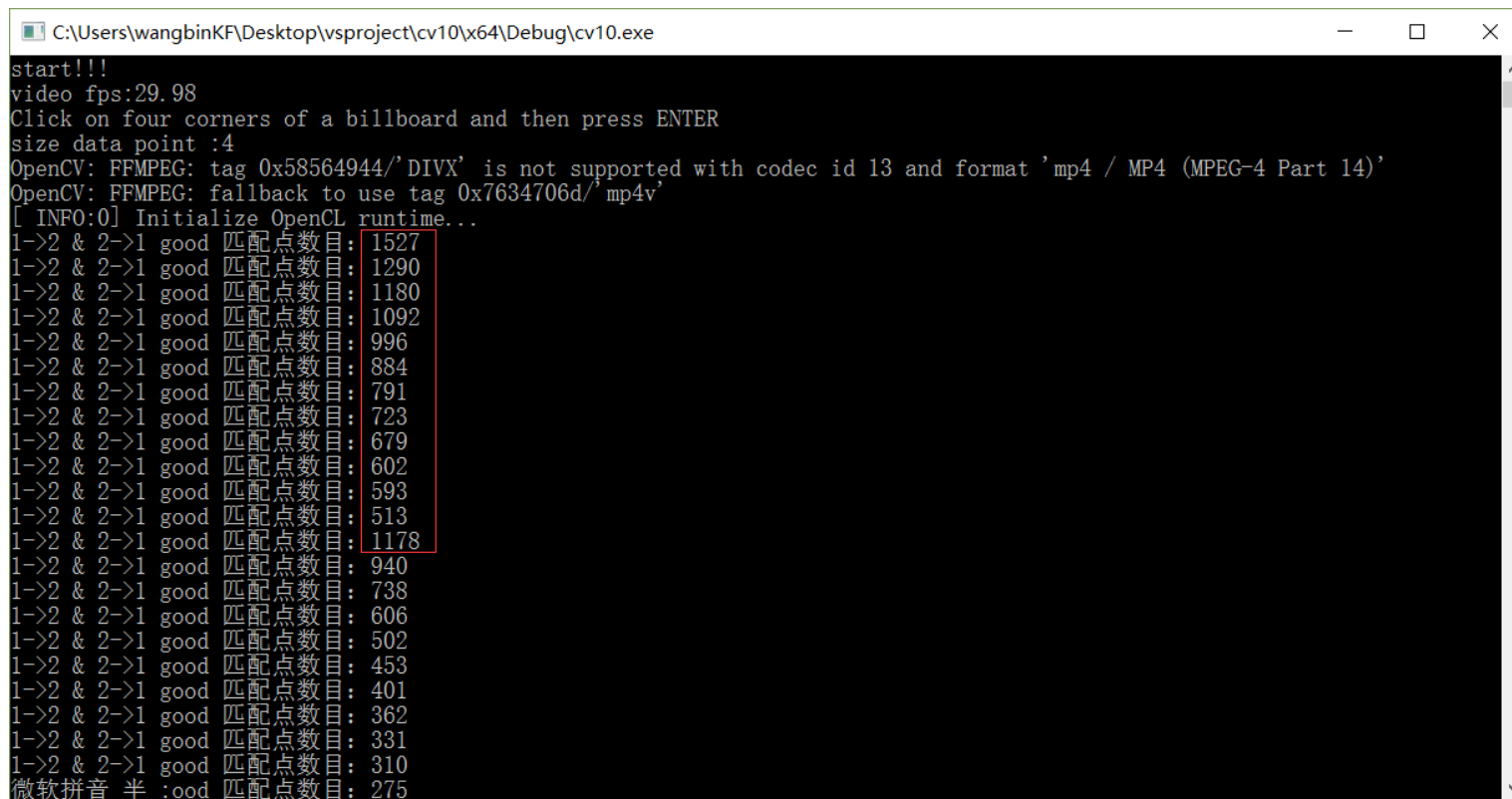


# 一、优化及一些问题

2、计算变换矩阵时，并不将当前帧与前一帧作为输入，而是将之前存储的base帧与当前帧输入，并且每隔一定帧数（10帧左右），更新base帧，同时更新基准顶点坐标

用前后帧或更新过快的话可能会导致意外的错误大量累积，每次更新的顶点都在上一次误差上加大，同时更新过慢的话，会由于图片场景变化过大，得到的匹配点数目越来越少，导致计算出的误差较大，不精准。

解决策略：调参！实验发现每隔10-12帧更新一次，结果较好，比较稳定



```
C:\Users\wangbinKF\Desktop\vsproject\cv10\x64\Debug\cv10.exe
start!!!
video fps:29.98
Click on four corners of a billboard and then press ENTER
size data point :4
OpenCV: FFMPEG: tag 0x58564944/'DIVX' is not supported with codec id 13 and format 'mp4 / MP4 (MPEG-4 Part 14)'
OpenCV: FFMPEG: fallback to use tag 0x7634706d/'mp4v'
[ INFO:0] Initialize OpenCL runtime...
1->2 & 2->1 good 匹配点数目: 1527
1->2 & 2->1 good 匹配点数目: 1290
1->2 & 2->1 good 匹配点数目: 1180
1->2 & 2->1 good 匹配点数目: 1092
1->2 & 2->1 good 匹配点数目: 996
1->2 & 2->1 good 匹配点数目: 884
1->2 & 2->1 good 匹配点数目: 791
1->2 & 2->1 good 匹配点数目: 723
1->2 & 2->1 good 匹配点数目: 679
1->2 & 2->1 good 匹配点数目: 602
1->2 & 2->1 good 匹配点数目: 593
1->2 & 2->1 good 匹配点数目: 513
1->2 & 2->1 good 匹配点数目: 1178
1->2 & 2->1 good 匹配点数目: 940
1->2 & 2->1 good 匹配点数目: 738
1->2 & 2->1 good 匹配点数目: 606
1->2 & 2->1 good 匹配点数目: 502
1->2 & 2->1 good 匹配点数目: 453
1->2 & 2->1 good 匹配点数目: 401
1->2 & 2->1 good 匹配点数目: 362
1->2 & 2->1 good 匹配点数目: 331
1->2 & 2->1 good 匹配点数目: 310
微软拼音 半 :ood 匹配点数目: 275
```

# 一、优化及一些问题

- 优化匹配点问题:

knnMatch + goodmatch  
+ RANSAC(封装在  
findHomography中)

- 核心代码示例:

```
while (true) {
    video >> curImg; myvideo >> inImg;
    if (!inImg.data || !curImg.data || waitKey(pauseTime) == 27) //图像为空或Esc键按下退出播放
    {
        break;
    }
    im_temp = curImg.clone();
    vimg = curImg.clone();
    //Mat curPart = curImg(Rect(originalPoint, processPoint));
    //h = getMyHomographMat(lastPart, curPart);
    h = getMyHomographMat(FstImg, curImg);
    //cout << "变换矩阵为: \n" << h << endl << endl; //输出映射矩阵
    perspectiveTransform(data.points, CURponits, h);
    //cout << "size data point : " << data.points.size() << endl;
    Mat HH = findHomography(pts_src, CURponits);
    warpPerspective(inImg, inImg, HH, curImg.size());
    for (int i = 0; i < 4; i++)
    {
        pts_dst[i] = CURponits[i];
    }
    fillConvexPoly(vimg, pts_dst, 4, Scalar(0), CV_AA);
    shImg = vimg + inImg;
    //不去更新对比图片, 如果每次在上一次的点基础上变换, 会导致误差不断累积
    //imshow("result", shImg);
    count_frame++;
    if (count_frame % 10 == 0) {
        FstImg = im_temp;
        //lastPart = im_temp(Rect(originalPoint, processPoint)); //提升速度
        data.points = CURponits;
    }
    writerr.write(shImg);
    //shImg.release();
}
```



## 二、视频成果

