### 滚动字幕的检测与拼接

刘哲泽 大三 南开大学

1. **实验要求**
2. 在一段视频中，对于一个给定的字幕区域，判断其是否为滚动字幕。
3. 对于滚动字幕，将其拼接为完整的字幕条。
4. 对于固定字幕，直接将其输出。

我借助了opencv完成了实验（opencv 4.9.0，Visual Studio 2022，windows 11）。

1. **实验内容**
2. 判断是否为滚动字幕
3. 确定字幕区域

根据图片，人工确定[x,y,width,height]，以确定字幕区域。

1. 滚动字幕判断

我使用了一个启发式的判断方法，来区分滚动字幕和固定字幕。该步骤在第（二）节第3小节得到good\_matches之后。

首先定义一个map，它的key值为两点间的L1距离，value值为出现次数。

|  |
| --- |
| 1. map<double, int> count;*//L1距离 出现次数* |

然后，遍历good\_matches，计算match的两个点间的L1距离，并加入map中：

|  |
| --- |
| 1. for (int i = 0; i < good\_matches.size(); i++) { 2. ...... 3. int x1 = kps1[good\_matches[i].queryIdx].pt.x; 4. int y1 = kps1[good\_matches[i].queryIdx].pt.y; 5. int x2 = kps2[good\_matches[i].trainIdx].pt.x; 6. int y2 = kps2[good\_matches[i].trainIdx].pt.y; 7. int L1 = abs(x1 - x2) + abs(y1 - y2); 8. if (count.find(L1) == count.end()) { 9. *//key不存在* 10. count.insert(pair<double, int>(L1, 1)); 11. } 12. else { 13. map<double, int>::iterator it = count.find(L1); 14. it->second = it->second + 1; 15. } 16. } |

接着，根据value值，对map从大到小排序：

|  |
| --- |
| 1. vector< pair<double, int> > vec; 2. for (map<double, int>::iterator it = count.begin(); it != count.end(); it++) { 3. vec.push\_back(pair<double, int>(it->first, it->second)); 4. } 5. sort(vec.begin(), vec.end(), [](pair<double, int>a, pair<double, int>b) {return a.second > b.second; }); |

最后，判断出现次数最大的L1距离，是否小于60（经验值），如果小于，判断为固定字幕，退出；否则，认定为滚动字幕，继续接下来的步骤。vec.size() == 0是为了避免vec为空而出现异常。

|  |
| --- |
| */\*==========判断是否为滚动字幕===========\*/*   1. if (vec.size() == 0 || vec[0].second < 60) { 2. *//根据相似点匹配数量进行判断* 3. cout << endl; 4. cout << "=================================" << endl; 5. cout << "           不是滚动字幕" << endl; 6. cout << "=================================" << endl; 7. Mat res; 8. vconcat(image2, image1, res); 9. imwrite(output\_fix, res); 10. return; 11. } 12. else { 13. cout << endl; 14. cout << "=================================" << endl; 15. cout << "           是滚动字幕" << endl; 16. cout << "=================================" << endl; 17. } |

我选择这种启发式判断方法的依据是，滚动字幕匹配到的特征点，具有平移关系，所以具有相同L1距离的滚动字幕的匹配特征点的数量应该比固定字幕的多。

1. 滚动字幕的拼接

我自定义一个函数用于滚动字幕的拼接，其中有四个参数：

* + string left: 要拼接的左边图的地址
  + string right: 要拼接的右边图的地址
  + Rect roi: 字幕区域
  + bool processed: 右图是否已经根据roi裁剪过（这是为了能够连续拼接多张图）

|  |
| --- |
| 1. void mystitch(string left, string right, Rect roi, bool processed = false) |

接下来对该函数的实现进行介绍：

1. 读取图片

设置保存地址，如果为滚动字幕，存储在output\_roll中；反正存储在output\_fix中；接着进行对图片的预处理操作，首先使用imread读取图片，然后对右图进行裁剪得到字幕区域；根据传入的processed，判断左图是否需要裁剪，并根据判断结果进行不同操作。注意，最终image1存储的是右图的字幕区域。image2存储的是左图的字幕区域。

|  |
| --- |
| 1. string output\_roll = "result/roll.jpg"; 2. string output\_fix = "result/fix.jpg"; 3. *// [预处理]* 4. Mat src1 = imread(left); 5. Mat src2 = imread(right); 6. */\** 7. \* 提取 ROI 区域 8. \* x y width height 9. \*/ 10. *//获取ROI区域，这是要拼接的滚动字幕区域* 11. Mat image1 = src2(roi);*//右图* 12. Mat image2;*//左图* 13. if (processed) { 14. image2 = src1; 15. } 16. else { 17. *//如果传入的是未处理的原始图像，直接赋值* 18. *//（为了能够多次拼接）* 19. image2 = src1(roi); 20. } 21. imwrite("result/right.jpg", image1); 22. imwrite("result/left.jpg", image2); |

1. 计算特征点和描述子[[1]](#footnote-0)

我使用了SIFT算法进行图片特征点的检测，得到图片image1和image2的特征点集合kps1和kps2。SIFT检测器得到的KeyPoint的内容如下：

* 关键点的坐标位置 （x, y）
* 关键点的尺度大小
* 关键点的方向角度（为每个特征点指定方向，以实现图像旋转的不变性）
* 关键点的响应值 （表示关键点的显著性），SIFT默认以0.03作为阈值
* 其他可能的属性或描述符

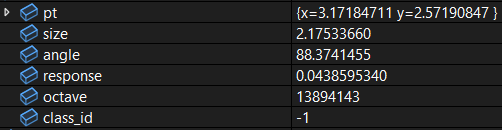


图 1 KeyPoint的值

得到特征点后，就可以进行特征描述子的计算。在特征点（KeyPoint）周围选取 16x16 的邻域。将它为 16 个 4x4 大小的子块。对于每个子块，创建包含 8 个 bin [[2]](#footnote-1)的方向直方图。因此总共有 128 个 bin 值可用（如下图中的cols = 128）。由这 128 个形成的向量构成了特征点描述子。如下所示，特征描述子记录了特征点的一些信息，比如“rows = 475”表示共有475个特征点。

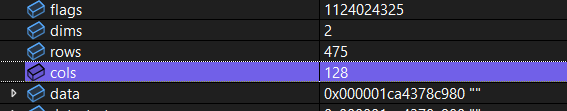


图 2 特征描述子部分内容

|  |
| --- |
| 1. *// [SIFT]* 2. *//检测特征点* 3. Ptr<SIFT> detector = SIFT::create(); 4. vector<KeyPoint> kps1, kps2; 5. detector->detect(image1, kps1); 6. detector->detect(image2, kps2); 7. *//计算特征描述符* 8. Mat descriptors1; 9. Mat descriptors2; 10. detector->compute(image1, kps1, descriptors1); 11. detector->compute(image2, kps2, descriptors2); 12. *//绘制特征点* 13. *//draw\_feature(image1, kps1, descriptors1);* 14. *//draw\_feature(image2, kps2, descriptors2);* 15. *// [SIFT]* |

1. 匹配特征

使用opencv的BruteForce匹配器进行特征匹配。其中的knnMatch函数，对于每个查询特征描述子，它会计算与其最接近的k个训练特征描述子，并返回这k个最近邻的距离。这样可以得到更多的匹配结果，但也可能包含一些不太准确的匹配。（使用knnMatch函数时，设置k=2，然后可以计算最近距离与第二最近距离的比率。如果比率大于 0.8，则忽略它们。根据脚注1，这样做消除了大约 90％的错误匹配，而同时只去除了 5％的正确匹配）

|  |
| --- |
| 1. *// [BruteForce Matcher]* 2. *//创建特征点匹配器* 3. Ptr<DescriptorMatcher> matcher = DescriptorMatcher::create("BruteForce"); 4. *//进行特征匹配* 5. */\** 6. class DMatch{ 7. ... 8. int queryIdx; // 特征点在第一幅图像中的索引 9. int trainIdx; // 特征点在第二幅图像中的索引 10. float distance; // 两个特征点之间的距离 11. }; 12. \*/ 13. vector<vector<DMatch>> matches; 14. vector<DMatch> good\_matches; 15. *//matcher->match(descriptors1, descriptors2, matches);//vector<DMatch> matches* 16. matcher->knnMatch(descriptors1, descriptors2, matches, 2);*//k=2* 17. *// Lowe's algorithm,获取优秀匹配点* 18. for (int i = 0; i < matches.size(); i++) 19. { 20. if (matches[i][0].distance > 0.8 \* matches[i][1].distance) 21. { 22. *//最近距离与第二最近距离的比率，大于 0.8，则忽略它们* 23. *//消除了大约 90％的错误匹配，而同时只去除了 5％的正确匹配* 24. } 25. else { 26. good\_matches.push\_back(matches[i][0]); 27. } 28. } |

1. 根据匹配到的特征，计算单应性矩阵

首先，得到源点和目的点：

|  |
| --- |
| 1. vector<Point2f> srcPoints, dstPoints; 2. for (int i = 0; i < good\_matches.size(); i++) { 3. srcPoints.push\_back(kps1[good\_matches[i].queryIdx].pt); 4. dstPoints.push\_back(kps2[good\_matches[i].trainIdx].pt); 5. } |

然后使用opencv的RANSAC 算法计算单应性矩阵，用于透视校正或图像对齐：

|  |
| --- |
| 1. Mat H = findHomography(srcPoints, dstPoints, RANSAC); |

1. 对图片进行透视变换

首先，使用opencv的warpPerspective方法对右侧图像（image1）进行透视变换，设置变换后的图片宽度为原始图像的两倍，高度相同：

|  |
| --- |
| 1. *//对图像进行透视变换* 2. Mat imageTransform1, imageTransform2; 3. *//warpPerspective(image1, imageTransform1, H, Size(image1.cols + image2.cols, image2.rows));* 4. warpPerspective(image1, imageTransform1, H, Size(image1.cols + image2.cols, image2.rows)); |

得到的某个右图输出如下所示：



图 3 透视变换1

然后，将左侧图像拷贝到透视变换的图像中：

|  |
| --- |
| 1. *//图像拷贝* 2. *//创建拼接后的图,需提前计算图的大小* 3. int dst\_width = imageTransform1.cols;  *//取最右点的长度为拼接图的长度* 4. int dst\_height = image2.rows; 5. Mat dst(dst\_height, dst\_width, CV\_8UC3); 6. dst.setTo(0); 7. imageTransform1.copyTo(dst(Rect(0, 0, imageTransform1.cols, imageTransform1.rows))); 8. image2.copyTo(dst(Rect(0, 0, image2.cols, image2.rows)));*//复制到左边* |

得到的图像如下所示：



图 4 透视变换2

注意到右侧有黑边，为了美观也为了能够拼接多个滚动字幕，我进行了消除黑色区域的操作。

1. 其他操作
   1. 去除右侧黑色区域

定义了一个函数delete\_black，用于消除右侧的黑色区域，如下所示，先将RGB图像转为灰度图像，然后再转为二值图像，接着我利用opencv自带的函数boundingRect进行分割，得到一个矩形框，该矩形块包含的是非黑色区域内容，如图 5所示。接着只保留矩形框区域即可，最终得到的图如图 6所示。

（Sample1/016660.jpg Sample1/016717.jpg 165, 520, 550, 50）

|  |
| --- |
| 1. void delete\_black(Mat& image) { 2. *//去除右侧黑边* 3. int rs = image.rows; 4. int cs = image.cols; 5. try { 6. *// 转换为灰度图像* 7. Mat grayImage; 8. cvtColor(image, grayImage, cv::COLOR\_BGR2GRAY); 9. *// 阈值处理* 10. Mat binaryImage; 11. threshold(grayImage, binaryImage, 1, 255, cv::THRESH\_BINARY); 12. *// 使用boundingRect确定黑边范围* 13. Rect boundingRect = cv::boundingRect(binaryImage); 14. *// // 在原始图像上绘制边界矩形* 15. *//rectangle(image, boundingRect, cv::Scalar(0, 255, 0), 2);* 16. image = image(boundingRect); 17. *//show(image);* 18. *//cout << rs << " " << cs << endl;* 19. *//cout << image.rows << " " << image.cols << endl;* 20. } 21. catch (Exception e) { 22. cout << e.err; 23. } 24. } |

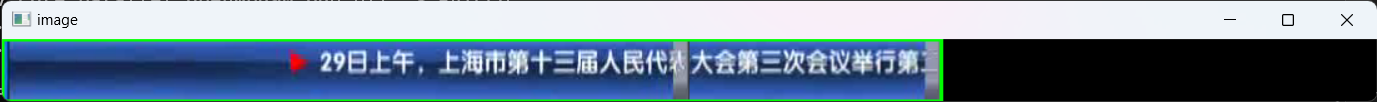


图 5 矩形框



图 6 拼接结果

去除了黑框以后，就能进行以拼接结果作为左图，新的图作为右图，进行连续拼接了。（如图 7所示）

（图 6 Sample1/016816.jpg 165, 520, 550, 50）

|  |
| --- |
| 1. *//连续拼接* 2. string left = "result/roll.jpg"; 3. string right = "Sample1/016816.jpg"; 4. mystitch(left, right, roi[0], true); |

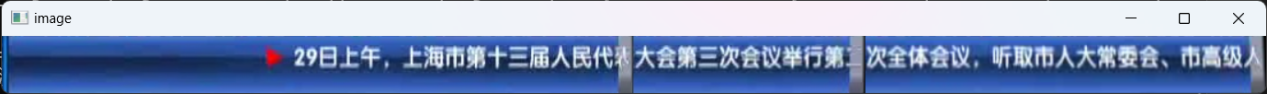


图 7 多次拼接结果

1. **测试**
2. 测试1：滚动字幕

left

图 8 left

right

图 9 right

roll

图 10 合并

1. 测试2：固定字幕

这两个数据是玩自己创建[[3]](#footnote-2)。

left

图 11 left

right

图 12 right

判定为固定字幕，我将两个图片上下拼接，然后保存拼接后的图片。



图 13 固定

1. 测试3：多个滚动字幕拼接

left

图 14 left

right

图 15 right1

right

图 16 right2

roll

图 17 合并

1. 测试4：距离很近的滚动字幕，判断为 是滚动字幕

left

图 18 left

right

图 19 right

roll

图 20 合并

1. **存在问题**
2. 测试5：距离很远的滚动字幕，判断为 不是滚动字幕

我的算法，会将此种情况的字幕，判断为固定字幕。

left

图 21 left

right

图 22 right

判断为固定字幕：



图 23 固定

1. 测试6：两个不同帧但相同的固定字幕

我的算法会将此种情况，判断为滚动字幕。

left

图 24 left

right

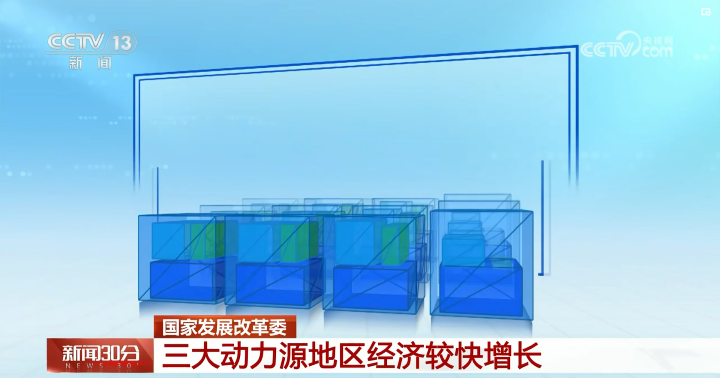
图 25 right

roll

图 26 合并

1. 思考
2. 给定的样本不含有下面情况的固定字幕，即两帧的固定字幕不同。





1. 我的算法会将距离很长的两帧的滚动字幕判定为固定字幕，以及会将不同帧但内容相同的固定字幕判定为滚动字幕；我觉得这两种情况应当不予讨论，我认为滚动字幕总有一部分文字相同，而固定字幕应当具有不同文字。另一方面，我暂时找不出这个问题的解决办法。
2. 我遇到的主要问题，就是如何区分滚动字幕和固定字幕，我放到了最后才解决这个问题。我是假设滚动字幕的字体是平移过去的，因此滚动字幕的匹配特征点应当具有大量相等的距离，我也是基于此来进行二者的区分的。除此之外，我还有另外一个想法，滚动字幕应当有部分连续的字体相同，因此可以识别出字体，然后进行判断，但此种方法未进行实践。
3. 我目前未能很好的理解单应性矩阵以及透视变换的原理，对它们的理解仅仅停留在单应性矩阵是确定四个点位置，并在不同的视图角度中均能确定这四个位置；透视变换是将图片将图片投影到一个新的视平面，即投影映射。

1. 该小节和第3小节内容的原理来自D.Lowe 的论文Distinctive Image Features from Scale-Invariant Keypoints，本人未阅读，在此仅作为说明。 [↑](#footnote-ref-0)
2. bin值指的是方向直方图中每个柱子的值，用来表示该子块内的梯度方向信息。 [↑](#footnote-ref-1)
3. https://tv.cctv.com/2024/05/01/VIDEyGVj8XoEEcIFwer6XKy5240501.shtml?spm=C45404.P9Z56wAyBcZ2.E8JlE0NUktSy.2中的2:51和3:11的截图。 [↑](#footnote-ref-2)