

计算摄影学作业报告

Lab 6 —— 全景图拼接

Jessie Peng

2019/05/21

1 实验内容

本实验我们实现全景图拼接算法。

全景图拼接是利用同一场景的多张图像通过重叠部分寻找匹配关系，从而生成整个场景图像的技术。

本次实验需要实现柱面坐标转换、特征匹配和全景图拼接三个部分，接口类定义如下：

```
class CylindricalPanorama
{
public:
virtual bool makePanorama(std::vector<cv::Mat>& img_vec, cv::Mat&
img_out, double f) = 0;
};
```

其中 `img_vec` 为输入的平面图像，`img_out` 为求出的全景图，`f` 为相机焦距。

本次实验的测试用例有两组，分别为从左至右按顺序排列的 12 张和 20 张图像，相机的焦距均已在 `K.txt` 文件中给出。



DSC01538.JPG



DSC01539.JPG



DSC01540.JPG



DSC01541.JPG



DSC01542.JPG



DSC01543.JPG



DSC01544.JPG



DSC01545.JPG



DSC01546.JPG



DSC01547.JPG



DSC01548.JPG



DSC01549.JPG



K.txt



DSC01599.JPG



DSC01600.JPG



DSC01601.JPG



DSC01602.JPG



DSC01603.JPG



DSC01604.JPG



DSC01605.JPG



DSC01606.JPG



DSC01607.JPG



DSC01608.JPG



DSC01609.JPG



DSC01610.JPG



DSC01611.JPG



DSC01612.JPG



DSC01613.JPG



DSC01614.JPG



DSC01615.JPG



DSC01616.JPG



DSC01617.JPG



DSC01618.JPG



K.txt

2 实验环境

编程语言: C++

开发环境: CLion 2018.3

操作系统: macOS 10.14.1 (18B75)

3 实验原理

3.1 柱面坐标转换

当相机位置固定，拍摄方向只在水平方向旋转时，可以使用柱面坐标映射的方式来求得全景图。

假设坐标原点在图像中心，则将平面坐标转换到柱面坐标有如下公式：

$$x' = r \tan^{-1} \left(\frac{x}{f} \right)$$

$$y' = \frac{ry}{\sqrt{x^2 + f^2}}$$

其中 r 为柱面半径， f 为焦距。

反过来，将柱面坐标转换为平面坐标有如下公式：

$$x = f \tan \left(\frac{x'}{r} \right)$$

$$y = \frac{y'}{r} \sqrt{x^2 + f^2}$$

3.2 SIFT 算法

SIFT (Scale Invariant Feature Transform) 算法在空间尺度中寻找极值点，并提取出其位置、尺度、旋转不变量。

Lowe 将 SIFT 算法分解为如下四步：

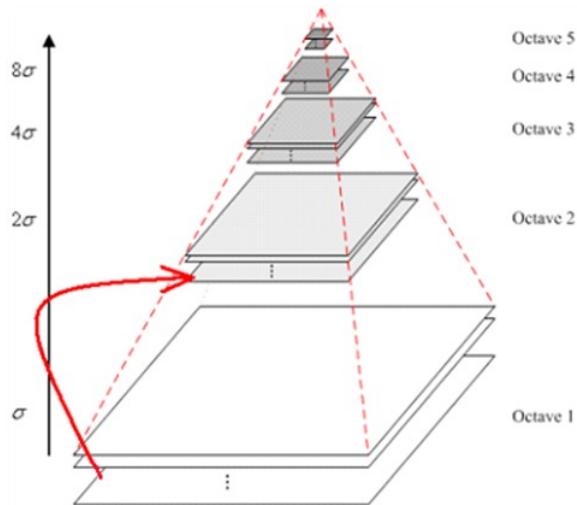
- 尺度空间极值检测：搜索所有尺度上的图像位置，通过高斯微分函数来识别潜在的对与尺度和旋转不变的兴趣点。
- 关键点定位：在每个候选的位置上，通过一个拟合精细的模型来确定位置和尺度。关键点的选择依据于它们的稳定程度。
- 方向确定：基于图像局部的梯度方向，分配给每个关键点位置一个或多个方向。所有后面的对图像数据的操作都相对于关键点的方向、尺度和位置进行变换，从而提供对于这些变换的不变性。
- 关键点描述：在每个关键点周围的邻域内，在选定的尺度上测量图像局部的梯度。这些梯度被转换成一种表示，这种表示允许比较大的局部形状的变形和光照变化。

3.2.1 尺度空间极值检测

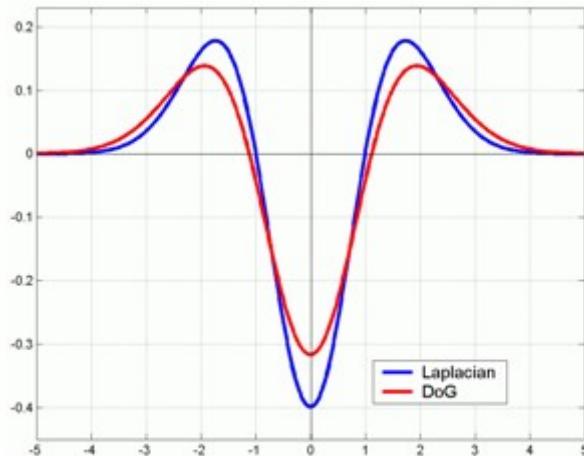
尺度空间中各尺度图像的模糊程度逐渐变大，能够模拟人在距离目标由近到远时目标在视网膜上的形成过程。大尺度对应于概貌特征，小尺度对应于细节特征。一个图像的尺度空间定义为一个变化尺度的高斯函数与原图像的卷积：

$$L(x, y, \sigma) = G(x, y, \sigma) * I(x, y)$$

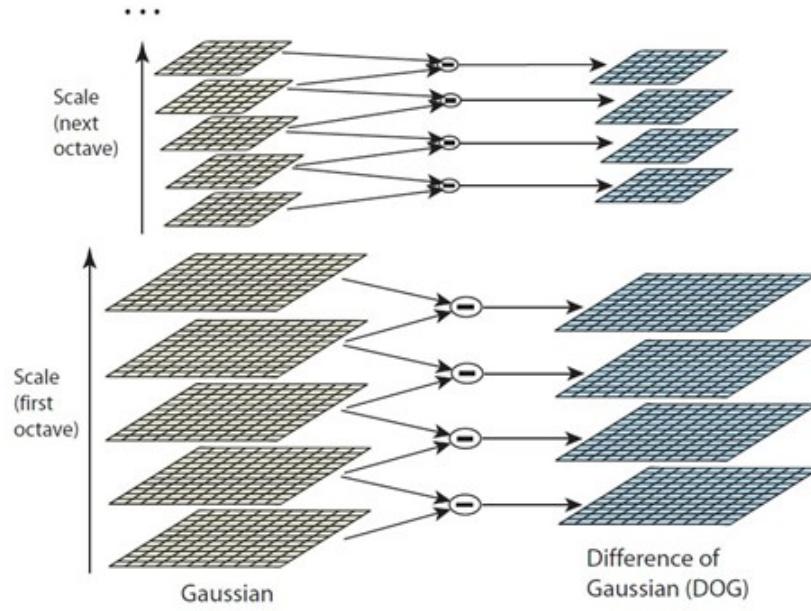
尺度空间可以用高斯金字塔表示：首先对图像做不同尺度的高斯模糊（因此每层有多张图像），然后由前一层的倒数第三张图像降采样形成金字塔的新的一层。



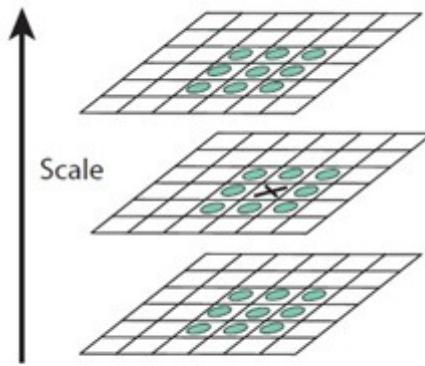
尺度规范化的 LoG (Laplacian of Gaussian) 算子具有真正的尺度不变性，而高斯差分函数 (Difference of Gaussian, DoG 算子) 与其非常近似：



Lowe 使用高斯差分金字塔（高斯金字塔中每个 Octave 中的上下两个相邻图像相减）近似 LoG 算子。



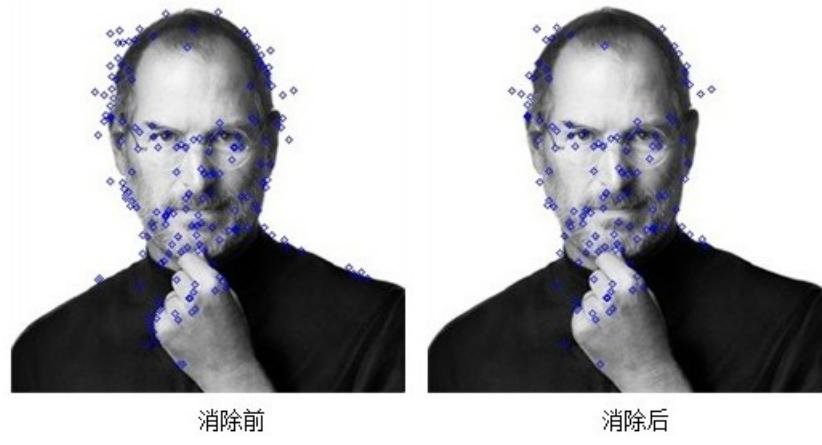
检测每个尺度空间的关键点，即找 DoG 函数的极值点，要将每个像素与周围的 26 个像素进行比较：



3.2.2 关键点定位

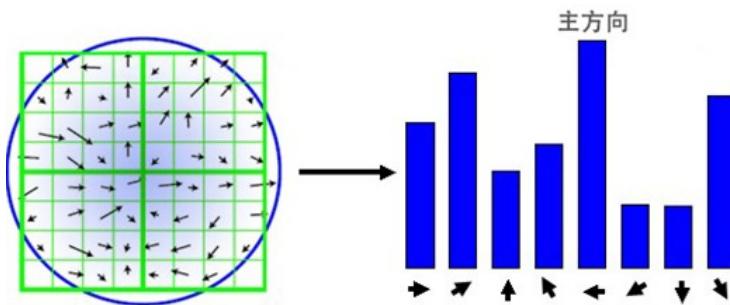
上述求得的极值点是离散空间中的，因此通过子像素插值（Sub-pixel Interpolation）得到连续空间的极值点。

另外，DoG 算子会产生较强的边缘效应，边缘上的点在某一个方向梯度很大，另一个方向梯度很小，即该点处 Hessian 矩阵的两个特征值相差很大。为了剔除边缘响应点，只需要设定一个阈值。



3.2.3 方向确定

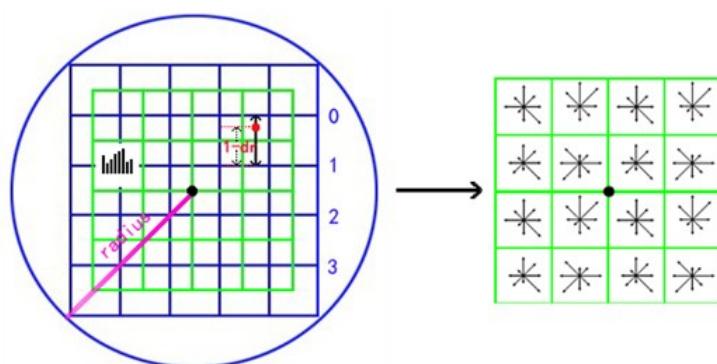
为了使描述符具有旋转不变性，需要给每个关键点确定基准方向。采集关键点在高斯金字塔图像 3σ 邻域窗口内像素的梯度和方向，并用直方图统计找到主方向。



保留峰值大于主方向峰值的 80% 的方向作为辅方向。实际编程实现中，就是把该关键点复制成多份关键点，并将方向值分别赋给这些复制后的关键点。并且，离散的梯度方向直方图要进行插值拟合处理，来求得更精确的方向角度值。

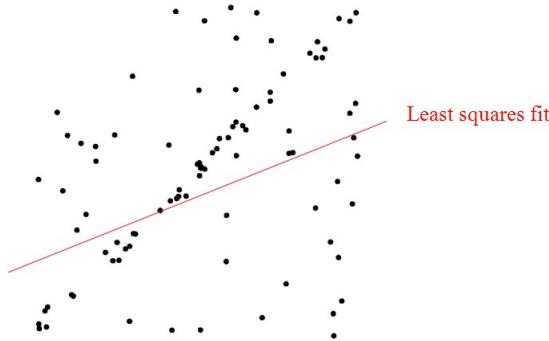
3.2.4 关键点描述

描述子使用在关键点尺度空间内 4×4 的窗口中计算的 8 个方向的梯度信息，共 $4 \times 4 \times 8 = 128$ 维向量表征：



3.3 RANSAC 算法

想从一组输入数据中找到一个模式，比如用直线拟合点云，当输入数据噪声较大时一般的回归方法比如最小二乘法不能奏效，如下图所示：

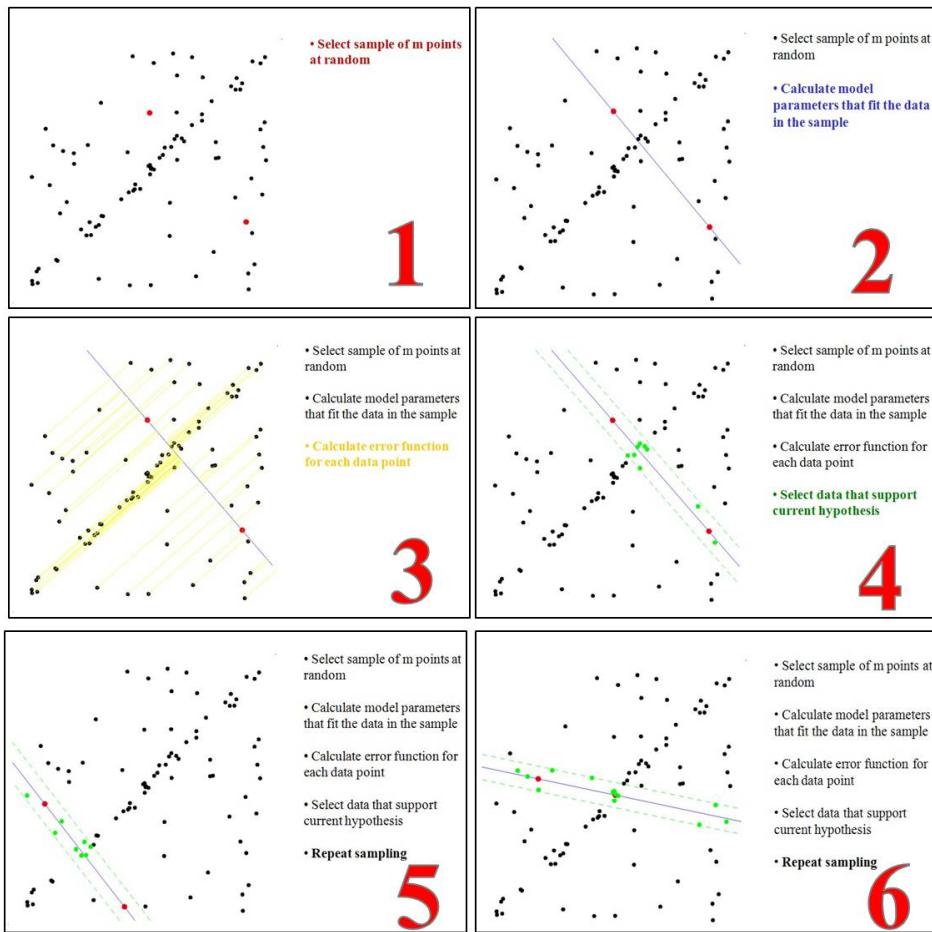


而 RANSAC (RANdom SAmple Consensus) 算法则能解决这类问题。

它的输入是一组可能含有较大的噪声或无效点的观测数据，一个用于解释观测数据的参数化模型以及一些可信的参数。样本中的正确数据称为内点(inliers)，错误数据称为外点(outliers)。

RANSAC 算法过程如下：

- 选择数据中的一组随机子集（子集大小正好可以计算所需参数）计算出一个模型，并假设该子集为内点
- 用该模型去测试所有的其它数据，则符合该模型的点也是内点
- 如果内点数量足够多，则该模型是合理的，然后用所有内点去重新估计模型（采用最小二乘等方法）。然后重新选取子集重复上述过程。
- 上述过程被重复执行固定的次数后，每次产生的模型要么因为内点太少而被舍弃，要么因为比现有的模型更好而被选用



更加数学化的语言描述如下：

- 考虑一个最小抽样集的势为 n 的模型 (n 为初始化模型参数所需的小样本数) 和一个样本集 P , 集合 P 的样本数 $\#(P) > n$, 从 P 中随机抽取包含 n 个样本的子集 S 初始化模型 M 。
- 余集 $SC = P \setminus S$ 中与模型 M 的误差小于某一设定阈值 t 的样本集以及 S 构成 S^* 。
 S^* 认为是内点集, 它们构成 S 的一致集 (Consensus Set)。
- 若 $\#(S^*) \geq N$, 认为得到正确的模型参数, 并利用集 S^* (内点 inliers) 采用最小二乘等方法重新计算新的模型 M^* 。重新随机抽取新的 S , 重复以上过程。
- 在完成一定的抽样次数后, 若未找到一致集则算法失败, 否则选取抽样后得到的最大一致集判断内外点, 算法结束。

4 实现过程与代码分析

本次实验中我分别处理两组测试图像。

对于每组测试图像, 首先进行柱面坐标投影, 然后进行图像配准, 最后将配准后的图像融合到一幅图中去产生全景图。

```
/** Part 1
 * Warp the images into cylindrical coordinates
 */
```

```

for (auto img : img_vec)
{
    img_cylinder.push_back(warpCylinder(img, f, f)); // set r = f
}

```

图像配准部分，我用输入图片序列的中间一幅作为参考，将其它图像都转换到它的坐标系里。具体来说，采用 SIFT 算法提取特征点和特征向量，并对每两幅相邻的图像：做特征点匹配，然后筛选出好的特征点，并用 RANSAC 算法剔除掉外点，最后用剩下的内点求出单应性矩阵，根据单应性矩阵把图像变换到目标坐标系中。

```

/** Part 2
 * Image registration
 */
for (int i = 0; i < n; i++)
{
    img_rectified.push_back(Mat::zeros(img_out.size(), CV_8UC3));
    mask_rectified.push_back(Mat::zeros(img_out.size(), CV_8UC3));
}

/** take the middle image as reference */
int ref = n / 2;
Mat img_ref = img_cylinder[ref];
img_ref.copyTo(img_rectified[ref](Rect(
    (img_out.cols - img_ref.cols) / 2, (img_out.rows - img_ref.rows) / 2,
    img_ref.cols, img_ref.rows
))); // put the reference image in the center

mask_cylinder.copyTo(mask_rectified[ref](Rect(
    (img_out.cols - img_ref.cols) / 2, (img_out.rows - img_ref.rows) / 2,
    img_ref.cols, img_ref.rows
)));

/** register images on the right side */
for (int i = ref; i < n - 1; i++)
{
    registerImage(img_cylinder[i+1], img_rectified[i], img_rectified[i+1],
    mask_cylinder, mask_rectified[i+1]);
}

/** register images on the left side */
for (int i = ref; i >= 1; i--)
{
    registerImage(img_cylinder[i-1], img_rectified[i], img_rectified[i-1],
    mask_cylinder, mask_rectified[i-1]);
}

```

图像融合部分，对于每个目标像素点，找到输入图像中所有对应的点。注意到输入图像中有一两幅出现了行人，造成了图像序列中的噪声，需要在融合前剔除噪声像素，然后使用剩下的像素平均值作为目标像素。

```

/** Part 3
 * Image blending
 */
blendImage(img_rectified, img_out, mask_rectified);

```

4.1 柱面投影

如下图所示，柱面投影后图像的尺寸会发生变化：



因此先利用转换后的坐标范围求出柱面投影图像的宽高：

```
int x_range = (img.cols - 1) / 2;
int y_range = (img.rows - 1) / 2;
int ret_x_range = static_cast<int>(plane_to_cylinder_X(x_range, r, f));
int ret_y_range = static_cast<int>(plane_to_cylinder_Y(0, y_range, r, f)); // when
x=0, y is the maximum
int ret_cols = ret_x_range * 2 + 1;
int ret_rows = ret_y_range * 2 + 1;
```

然后对于目标图像的每个像素点，根据逆变换公式求其在原图像中的坐标，并拷贝相对应的像素：

```
Mat img_ret = Mat::zeros(ret_rows, ret_cols, CV_8UC3);
for (int yy = 0; yy < ret_rows; yy++)
{
    for (int xx = 0; xx < ret_cols; xx++)
    {
        int x = cvRound(cylinder_to_plane_X(xx - ret_x_range, r, f));
        int y = cvRound(cylinder_to_plane_Y(x, yy - ret_y_range, r, f));
        x += x_range;
        y += y_range;

        if (x >= 0 && x <= img.cols && y >= 0 && y <= img.rows)
        {
            img_ret.at<Vec3b>(yy, xx) = img.at<Vec3b>(y, x);
        }
    }
}
```

4.2 特征点和特征向量的提取

使用 OpenCV 提供的接口，用 SIFT 算法对相邻的两幅图分别提取出特征点和特征向量：

```
/** Step 1
 * Detect features
 */
// note the OpenCV 3.X standard
Ptr<xfeatures2d::SiftFeatureDetector> detector =
xfeatures2d::SiftFeatureDetector::create();
Ptr<xfeatures2d::SiftDescriptorExtractor> extractor =
xfeatures2d::SiftDescriptorExtractor::create();

vector<KeyPoint> keypoint_src, keypoint_dst;
Mat descriptor_src, descriptor_dst;
```

```

/** 1.1 Detect keypoints with SIFT */
detector->detect(src, keypoint_src);
detector->detect(dst, keypoint_dst);

/** 1.2 Extract descriptors */
extractor->compute(src, keypoint_src, descriptor_src);
extractor->compute(dst, keypoint_dst, descriptor_dst);

```

每幅图提取出了一千个左右的特征点：



4.3 特征点筛选和匹配

用 OpenCV 提供的接口进行特征点匹配，由于特征点太多，所以我只筛选出了特征最好的 500 个点，筛选的标准就是每对 match 之间的距离越近越好。

```

/** Step 2
 * Match features
 */
Ptr<DescriptorMatcher> matcher = DescriptorMatcher::create("BruteForce");

vector<DMatch> matches;
vector<DMatch> good_matches;
vector<DMatch> inlier_matches;

Mat img_match;
int good_match_size = 500;

/** 2.1 Compute all correspondences with brute force */
matcher->match(descriptor_dst, descriptor_src, matches);

/** 2.2 Keep the top 500 matches */

```

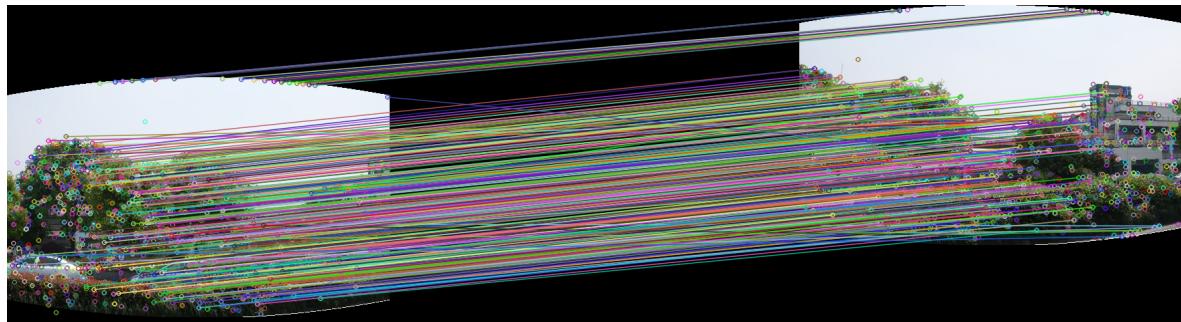
```

Mat dist = Mat::zeros(matches.size(), 1, CV_32F);
Mat sorted_idx;
for (int i = 0; i < matches.size(); i++)
{
    dist.at<float>(i, 0) = matches[i].distance;
}
sortIdx(dist, sorted_idx, SORT_EVERY_COLUMN + SORT_ASCENDING);

for (int i = 0; i < good_match_size; i++)
{
    good_matches.push_back(matches[sorted_idx.at<int>(i, 0)]);
}

```

这样做出来的匹配结果示例如下：



可见还是有一些误匹配的点，并且很多图片边缘的点也被误当作了特征点。于是采用 RANSAC 算法再剔除外点：

```

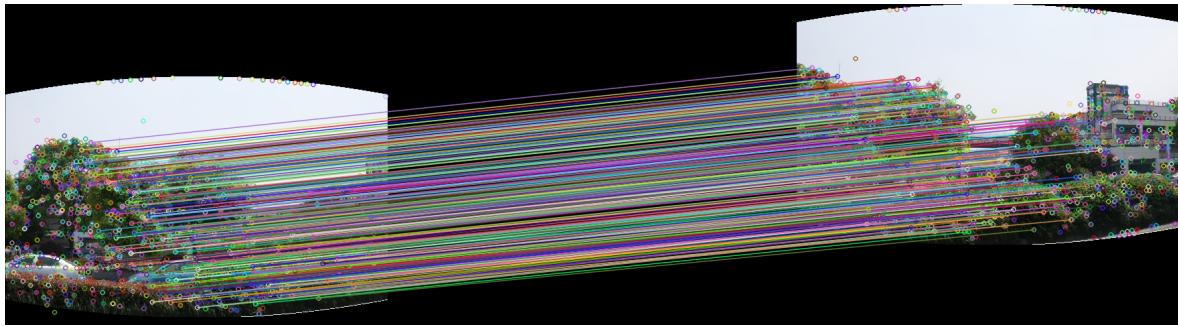
/** 2.3 Eliminate outliers with RANSAC */
vector<Point2f> src_float, dst_float;
for (auto it = good_matches.begin(); it != good_matches.end(); it++)
{
    src_float.push_back(keypoint_src[it->trainIdx].pt);
    dst_float.push_back(keypoint_dst[it->queryIdx].pt);
}

vector<uchar> is_inliers(good_match_size);
Mat homography = findHomography(src_float, dst_float, CV_RANSAC, 3, is_inliers);

for (int i = 0; i < is_inliers.size(); i++)
{
    if (is_inliers[i])
    {
        inlier_matches.push_back(good_matches[i]);
    }
}

```

剩下内点的匹配如下，可见刚才的很多误匹配都被剔除了，现在的结果很好：



4.4 单应性矩阵和透视变换

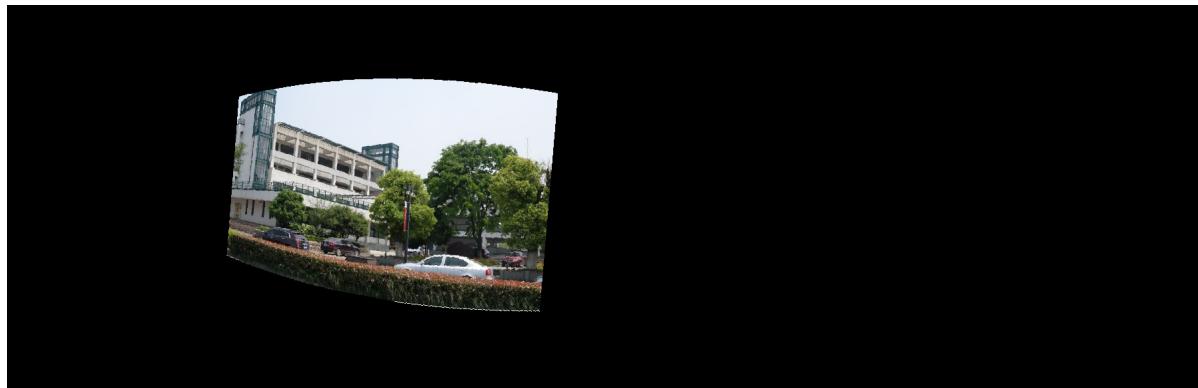
用上述筛选后的内点通过 OpenCV 提供的接口求得单应性矩阵，并对源图像做透视变换：

```
src_float.clear();
dst_float.clear();
for (auto it = inlier_matches.begin(); it != inlier_matches.end(); it++)
{
    src_float.push_back(keypoint_src[it->trainIdx].pt);
    dst_float.push_back(keypoint_dst[it->queryIdx].pt);
}

homography = findHomography(src_float, dst_float, CV_RANSAC);

/** Step 3
 * Register images
 */
warpPerspective(src, out, homography, out.size(), INTER_NEAREST);
warpPerspective(src_mask, out_mask, homography, out.size(), INTER_NEAREST);
```

变换到目标坐标系中的图像示例：



这里我最开始碰到一些问题，当时我做出来的图像在拼接处（即柱面投影图像的边缘处）有明显的缝隙：



这些缝隙应该是深色而不是纯黑色的，因为纯黑色的背景在图像融合时不会参与对像素的贡献。这种深色缝隙很难通过像素值的筛选直接消除，因为图片里也有正常的深色像素，那么它是怎么形成的呢？

经过思考，我后来想到了对图像做透视变换的这个环节，图像发生形变必然涉及到像素点的插值，因此形变后的边界附近的点就是由图像边界上的点和黑色背景点做插值形成的，因此是深色的。

之前我在 `warpPerspective(src, out, homography, out.size());` 这一句中当时没有指定第四个参数 flags，它的默认值是 `INTER_LINEAR`，就是线性插值。后来我将其改成使用参数 `INTER_NEAREST`，表示取原图中最近的像素点，这样就解决了深色缝隙的问题。



4.5 图像融合及噪声剔除

注意到第二组测试图片中有两幅照片拍到了行人：



因为行人是移动的，而其它照片中都没有行人，因此这两幅图中的行人就成为了图像融合前必须剔除的噪声像素，否则将出现若隐若现的融合效果：



每个位置的噪声只存在于一幅照片中，因此可以将该位置在原图片中的所有对应像素拿来比较，如果发现有一个像素跟其它差别较大就可以判断是噪声。

对目标图片的每个位置，首先找到所有备选像素，即一范数不为 0 的像素：

```
for (auto img : img_vec)
{
    Vec3b tmp = img.at<Vec3b>(i, j);
    if (norm(tmp, NORM_L1) > 0)
    {
        candidates.push_back(tmp);
        sum += tmp;
    }
}
```

如果有两个及以上备选像素，则求一个平均像素，然后求所有备选像素到平均像素的距离（二范数）。如果距离很大（这里界限设为 20），并且比其它像素的平均距离明显要大（这里界限设为 1.5 倍），就判断是噪声，将其从备选像素中删除，然后更新平均像并拷贝给目标图片。

```

int cand_num = candidates.size();
if (cand_num > 1)
{
    average = sum / cand_num;

    /// get rid of noise
    if (cand_num > 2)
    {
        double dist[cand_num];
        for (int k = 0; k < cand_num; k++)
        {
            dist[k] = norm(candidates[k] - average, NORM_L2);
        }

        double max_dist = dist[0];
        int max_idx = 0;
        double sum_dist = dist[0];
        for (int k = 1; k < cand_num; k++)
        {
            if (dist[k] > max_dist)
            {
                max_dist = dist[k];
                max_idx = k;
            }
            sum_dist += dist[k];
        }

        if (max_dist > 20 && max_dist > sum_dist / cand_num * 1.5) // filter out
noise
        {
            sum -= candidates[max_idx];
            average = sum / (cand_num - 1);
        }
    }

    img_out.at<Vec3b>(i, j) = average;
}

```

对于只有一个备选像素的情况，直接拷贝即可：

```

else if (cand_num == 1)
{
    img_out.at<Vec3b>(i, j) = candidates[0];
}

```

这样融合后的图片成功把行人去除了：





5 结果分析与实验总结

5.1 全景图结果评价

两组测试图片拼接成的全景图整体效果都很好，场景完整清晰，拼接过渡自然，行人、接缝等噪声像素都被剔除了，总体来说是成功的。



但是仍然有需要改进的地方，比如原始图片总体亮度（或曝光程度）不一致，可能需要拼接前先预处理对图片亮度进行调整，才能做出各部分亮度更加统一的全景图。也可以考虑采用更高级的融合算法，使连接处过渡更加自然。

5.2 问题与解决

本次实验主要遇到如下的问题：

- 连接处深色缝隙的问题，上文已经讨论过。
- 场景中剔除行人的问题，上文也已讨论过。
- 图片融合的顺序问题：我最开始是采用的一张一张图依次叠加到输出图像上的方式进行融合，但是这样每次只能混合两个像素，不能很好区分出哪些像素是需要剔除的。后来改为一次性考虑所有图片，通过统计就能找到最不同的噪声像素（本实验中的行人），达到了比较好的效果。
- 最后是关于 OpenCV 的接口问题，在使用关于 SIFT 的接口时一开始误用了 OpenCV 2.X 的标准，写成如下形式：

```
cv::xfeature2d::SiftFeatureDetector detector;  
detector.create();  
detector.detect( img, keypoints );
```

我用的是 OpenCV 4.0，因此编译错误。后来查到了 OpenCV 3.X 的标准写法：

```
cv::Ptr<cv::xfeature2d::SiftFeatureDetector> detector = cv::xfeature2d::SiftFeatureDetector::create();  
detector->detect( img, keypoints );
```

6 参考文献

课程网站：<http://www.cad.zju.edu.cn/home/gfzhang/course/computational-photography/lab6-panorama/panorama.html>

Image Alignment and Stitching: A Tutorial: <https://www.microsoft.com/en-us/research/wp-content/uploads/2004/10/tr-2004-92.pdf>

OpenCV 3 "The function/feature is not implemented" error:

<https://stackoverflow.com/questions/30622304/opencv-3-blobdetection-the-function-feature-is-not-implemented-in-detectand>

Distinctive Image Features from Scale-Invariant Keypoints-SIFT 算法译文：

<http://www.cnblogs.com/cuteshongshong/archive/2012/05/25/2506374.html>

SIFT 算法详解：<https://blog.csdn.net/zddblog/article/details/7521424>

RANSAC 算法详解：<https://www.cnblogs.com/weizc/p/5257496.html>

RANSAC：<https://blog.csdn.net/u012704941/article/details/84325883>

图像拼接和图像融合技术：<https://www.cnblogs.com/skyfsm/p/7411961.html>