

- **写在前面**

这份报告包括了作业要求中5.1（思路分析+7张图片）与5.2的部分（分析+优化改进的bonus部分）

文件夹outputs是输出效果图，除了规定的7张图片外，另存有“panorama\_id\_old”命名的四张全景图，它们是对stitch\_blend 优化改进前运行的效果图（运行stitch\_blend注释掉的部分即可得到这四张图）；以及一张测试角点匹配的test图片

同时在这里，简单说明提交代码homework2.py中的一些相关事情：

1. 代码最后段多个自定义MakePanorama\_grail()、MakePanorama\_parrington()等四个函数，调用分别得到各自的全景拼接图
2. 四张全景图拼接时采用了不同的参数，分别如下：(四个函数内部注释中也可见，报告中的图片均为在这些参数组合下的运行结果)
  1. MakePanorama\_grail() --> panorama\_1：  
threshold = 0.75 (在def feature\_matching函数中); max\_iter = 1000, threshold = 4 (在def align\_pair函数中)
  2. MakePanorama\_library() --> panorama\_2：  
threshold = 0.8 (在def feature\_matching函数中); max\_iter = 1200, threshold = 4 (在def align\_pair函数中)
  3. MakePanorama\_parrington() --> panorama\_3：  
threshold = 0.85 (在def feature\_matching函数中); max\_iter = 1000, threshold = 4 (在def align\_pair函数中)
  4. MakePanorama\_XueMountainEntrance() --> panorama\_4：  
threshold = 0.8 (在def feature\_matching函数中); max\_iter = 1000, threshold = 4 (在def align\_pair函数中)
3. 对于stitch\_blending函数优化前后的对比效果图，均为在上述相同的参数组合下运行所得
4. 运行自定义的PairStitching()函数（用来保存1\_1/1\_2 2\_1/2\_2 3\_1/3\_2的pair-image stitching三张融合图）时，选用的参数是：threshold = 0.75 (在def feature\_matching函数中)
5. utils.py中一共有两个函数，分别用来转换为齐次坐标和生成高斯核
6. (不知道是否是可能是未修改好的小bug，可能会出现开机后第一次运行时无法拼接报错rate<0.03的情况，不用改动再运行一至两次即可)

## 5.1 Report

### 实现思路

#### Harris Corner Detection

图像拼接中，利用Harris角点检测获取所有角点，随后计算角点周围区域的梯度方向直方图，生成一个描述该角点及其局部区域的特征向量，然后对两张图进行角点的特征匹配。在这个过程中我完善了代码中的函数

harris\_response, corner\_selection, histogram\_of\_gradients，实现如下：

## 1) harris\_response

这个函数需要计算并返回图像的 Harris 响应 R，响应越大的地方越可能是角点。

根据课堂ppt，R的计算公式是：

$$R = \det(M) - \alpha \times (\text{trace}(M))^2$$

其中，矩阵 M 是由图像梯度的二阶矩阵组成，可以通过以下方式表示：

$$M = \begin{pmatrix} A & B \\ B & C \end{pmatrix}$$

$$A = I_x^2 \quad (\text{图像在 } x \text{ 方向梯度的平方})$$

$$B = I_x I_y \quad (\text{图像在 } x \text{ 和 } y \text{ 方向的梯度乘积})$$

$$C = I_y^2 \quad (\text{图像在 } y \text{ 方向梯度的平方})$$

$\text{trace}(M) = A+C$ ,  $\det(M) = AC-B^2$ ;

alpha是一个常量，通常取值在 0.04 到 0.06 之间，用于控制矩阵迹的影响。

同时，矩阵M的元素直接来源于图像的梯度信息，如果有噪声梯度可能会受到干扰，检测结果不准确。因此，为了减少噪声的影响使用一个高斯窗口进行平滑，代码如下：

```
def harris_response(img, alpha, win_size):
    if len(img.shape) == 3: # 将图像转换为灰度图
        img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    w = gaussian_blur_kernel_2d(win_size, 1.0)
    I_x = gradient_x(img)
    I_y = gradient_y(img)
    A = ndimage.convolve(I_x * I_x, w, mode='reflect')
    B = ndimage.convolve(I_x * I_y, w, mode='reflect')
    C = ndimage.convolve(I_y * I_y, w, mode='reflect')
    det = A * C - B ** 2
    trace = A + C
    R = det - alpha * (trace ** 2)

    return R
```

## 2) corner\_selection

根据上一个函数得到的Harris响应R，通过非极大值抑制来选择角点，并根据阈值筛选出有效的角点。

只有当一个像素的响应值大于其邻域内的所有像素才被认为是一个角点，利用 `ndimage.maximum_filter` 函数直接进行这局部最大值比较；

考虑到参数中 `min_dist` 对于两个角点之间的最小距离的要求，直接令这个filter的 `size=min_dist`，保证每个角点的 `min_dist` 范围内没有其他角点：

```
def corner_selection(R, thresh, min_dist):
    local_max = ndimage.maximum_filter(R, size=min_dist)
```

再根据阈值去除Harris响应值较低的点即可获得所有合格的角点：

```
R_selection = (R == local_max) & (R > thresh)
```

通过 `np.where` 找到这些角点的坐标，注意这个返回的顺序是（行，列），而根据框架代码中按照列主序进行处理，返回的pix列表中需要扭转一下行列顺序：

```
y_coords, x_coords = np.where(R_selection)
pix = [(x, y) for x, y in zip(x_coords, y_coords)]

return pix
```

### 3) histogram\_of\_gradients

前两步在找到、筛选角点后，需要计算角点周围区域的梯度方向直方图生成一个描述该角点及其局部区域的特征向量。

①准备工作：转换为灰度图像+获取图像宽高

```
if len(img.shape) == 3:
    img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
H, W = img.shape
```

②将图像划分为小块cell，每个 cell 计算一个梯度方向的直方图，而多个 cell 组成一个大的块block，经过测试cell = 4 (一个cell含4 \* 4个像素) 可以捕捉到更多的局部梯度变化，同时block = 16 (一个block由16 \* 16个cell组成) 增强稳定性，呈现效果较好

```
cell = 4
block = 16
```

④计算梯度方向：利用 `np.arctan2` 和 `np.degrees`，统一转化成0~360的角度制处理

```
grad_x = gradient_x(img)
grad_y = gradient_y(img)
grad_mag = np.sqrt(grad_x**2 + grad_y**2)
grad_dir = np.degrees(np.arctan2(grad_y, grad_x))
grad_dir[grad_dir < 0] += 360
```

⑤计算每个角点的 HOG 特征：以每个角点为中心，唯一对应block所反应大小的周围区域，行、列范围分别是 `y_start ~ y_end` 和 `x_start ~ x_end` (都用x表示列坐标，y表示行坐标)

```
tmp = [] # 用来存储每个角点的特征
L = block * block * 8 # 特征向量纬度，这里=8*16*16=1024
delta = int(cell * block // 2) # 用来控制特征提取区域的范围
for pixel in pix:
    cx, cy = pixel[0], pixel[1]
    feature = np.empty((L)) # 用来存储当前角点的特征向量
    feature_idx = 0 # 特征向量的索引
    y_start = cy - delta
    y_end = cy + delta
    x_start = cx - delta
    x_end = cx + delta
```

- 越界判断：如果角点的block区域超出了图像的边界，跳过这个交点，越界的区域被赋予全零特征向量：

```

if y_start < 0 or x_start < 0 or y_end >= H or x_end >= W:
    feature = np.zeros(block * block * 8)
    tmp.append(feature)
    features = np.array(tmp)
    continue

```

⑥ 从block遍历每个小块cell

```

for col_start in range(x_start, x_end - cell, cell): # 遍历cell的每个像素
    for row_start in range(y_start, y_end - cell, cell):
        col_end = col_start + cell
        row_end = row_start + cell

```

第一次遍历遍历cell的每个像素点，36个分区（更细致的方向划分）找主方向计算：

```

hist = np.zeros(36)
for col in range(col_start, col_end + 1):
    for row in range(row_start, row_end + 1):
        hist[int(grad_dir[row, col] // 10)] += grad_mag[row, col]

```

得到主方向后，第二次遍历 cell 的每个像素，先将梯度方向对齐到主方向提高特征的不变性，再按照8个分区得到特征向量：

```

main_degree = np.argmax(hist) * 10
_hist = np.zeros(8)
for col in range(col_start, col_end + 1):
    for row in range(row_start, row_end + 1):
        rotate_dir = (grad_dir[row, col] - main_degree) % 360
        _hist[int(rotate_dir // 45)] += grad_mag[row, col]

```

⑦ 特征归一化：每个角点的特征向量最终由多个 cell 的梯度方向直方图组成，我们将这些直方图合并成一个  $L$  维的特征向量，然后对特征向量进行归一化，避免由于图像亮度变化或对比度变化导致的特征向量差异，增强特征描述符的稳定性。最后函数返回：

```

feature[feature_idx:feature_idx + 8] = _hist
feature_idx += 8 # 更新特征向量索引
feature = np.array(feature)
feature /= (np.linalg.norm(feature) + 1e-10) # 每个block，做一次归一化
tmp.append(features)
features = np.array(tmp)
return features

```

## 2.RANSAC for homography

图像拼接单应性矩阵描述两个图像之间的变换关系，用于将一个图像中的点映射到另一个图像中的对应点。

`compute_homography` 函数已经实现计算任意给定的两对点列间的单应性矩阵。

而为了减少点列中 outliers 的影响精准对齐，需要借助 `align_pair` 函数中的 RANSAC 算法，每次随机选取+多次迭代，最终返回最佳的单应性矩阵 `homo_matrix`

## 1) compute\_homography

首先在 `utils.py` 补充一个函数，得到二维点列转化到齐次坐标的结果：后续计算需要基于齐次坐标进行。

```
def listpoint_homo_coordinate(list_of_points):
    list_of_points = np.asarray(list_of_points)
    N=np.shape(list_of_points)[0]
    if N == 0:
        return np.zeros((0, 3))
    homo_listpoint=np.ones((N,3))
    homo_listpoint[:,0:2]=list_of_points
    return homo_listpoint
```

为了计算单应性矩阵，需要建立一个矩阵  $A$ ，它由匹配点对提供的线性方程组成：注意到每一对匹配点提供 2 个方程，需要至少 4 对点来构建 8 个方程解出单应性矩阵的 8 个自由度。

对于每一对匹配点  $(x_1, y_1)(x_2, y_2)$ ，满足以下关系：

$$\begin{pmatrix} x_2 \\ y_2 \\ 1 \end{pmatrix} = H \cdot \begin{pmatrix} x_1 \\ y_1 \\ 1 \end{pmatrix}$$

这两对方程可以展开为：

$$\begin{aligned} x_2 &= h_{11}x_1 + h_{12}y_1 + h_{13} \\ y_2 &= h_{21}x_1 + h_{22}y_1 + h_{23} \end{aligned}$$

合并所有的方程组，并写成  $Ah=0$  的矩阵方程形式，上面两对方程的点对应  $A$  中的这两行：

$$\begin{pmatrix} x_1 & y_1 & 1 & 0 & 0 & 0 & -x_2x_1 & -x_2y_1 & -x_2 \\ 0 & 0 & 0 & x_1 & y_1 & 1 & -y_2x_1 & -y_2y_1 & -y_2 \end{pmatrix}$$

每对匹配点对提供 2 行，最终的矩阵  $A$  是  $2N*9$  的矩阵（ $N$ : 匹配点对的数量），根据上述公式填充  $A$  矩阵：

```
A = np.zeros((2 * N, 9))
for i in range(N):
    x1, y1, _ = pixels_1_homo[i]
    x2, y2, _ = pixels_2_homo[i]
    A[2 * i] = np.array([-x1, -y1, -1, 0, 0, 0, x2 * x1, x2 * y1, x2])
    A[2 * i + 1] = np.array([0, 0, 0, -x1, -y1, -1, y2 * x1, y2 * y1, y2])
```

然后使用奇异值分解来解线性方程，在最小二乘解中找到最佳拟合的单应性矩阵：

```
_, S, V = np.linalg.svd(np.transpose(A).dot(A))
```

最小奇异值对应的解是最接近零的解，它表示对所有点的最小误差，选择这个解作为返回值：

```
homo_matrix = np.reshape(V[np.argmin(S)], (3, 3))
return homo_matrix
```

## 2) align\_pair

根据课堂ppt，在solve homography过程中每一对匹配点会产生两个方程，而homography的自由度为8，意味着至少需要4对不共线的匹配点对。所以在RANSAC算法中，每一次迭代过程中，在长度为 `N` 的点列中随机抽取4个正整数作为点对的索引：

```
def align_pair(pixels_1, pixels_2):
    N = len(pixels_1)
    pixels_1_homo = listpoint_homo_coordinate(pixels_1)
    pixels_2_homo = listpoint_homo_coordinate(pixels_2)
    est_homo = np.zeros((3, 3))
    max_iterate = 1000
    threshold = 4.0
    max_inliers = 0
    for _ in range(max_iterate):
        rand = np.random.choice(N, 4, replace=False)
        rand_pixels_1 = [pixels_1[j] for j in rand]
        rand_pixels_2 = [pixels_2[j] for j in rand]
```

进过齐次坐标化，直接根据上一个 `compute_homography` 函数计算处当前随机取4对匹配点确定的 `homo_matrix`

```
homo_matrix = compute_homography(rand_pixels_1, rand_pixels_2)
```

用得到的单应性矩阵 `homo_matrix` 对图像1中的点进行变换，映射到图像2的空间：

```
trans_pixels_1 = homo_matrix.dot(np.transpose(pixels_1_homo))
trans_pixels_1 = trans_pixels_1 / trans_pixels_1[2, :]
trans_pixels_1 = np.transpose(trans_pixels_1)
```

计算变换后点与真实点的欧氏距离 `dis`，根据设置的 `threshold` 阈值内点数量，并更新最佳的单应性矩阵：

```
dis = spatial.distance.cdist(trans_pixels_1, pixels_2_homo, metric='euclidean')
dis = np.diagonal(dis)
num_inliers = np.count_nonzero(dis < threshold)
if num_inliers > max_inliers:
    max_inliers = num_inliers
    est_homo = homo_matrix
return est_homo
```

## 可视化效果

**blend\_1.jpg**



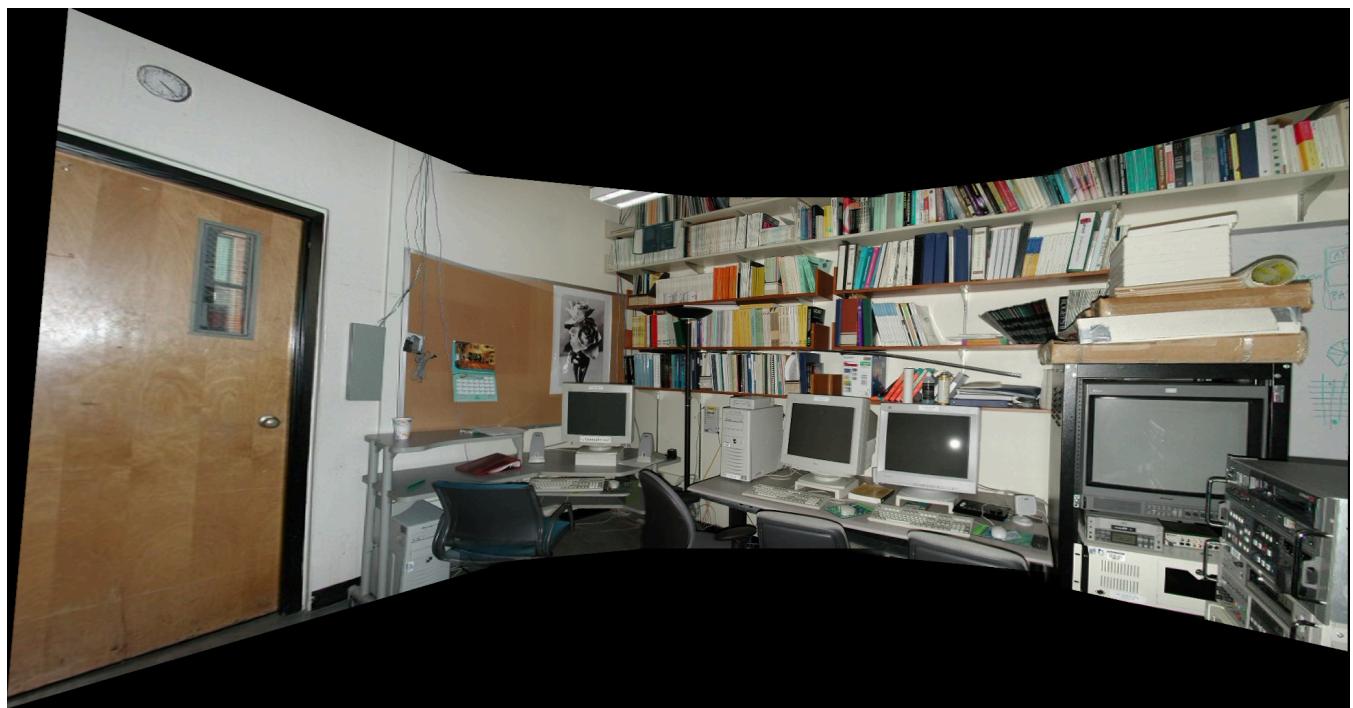
**blend\_2.jpg**



**blend\_3.jpg**



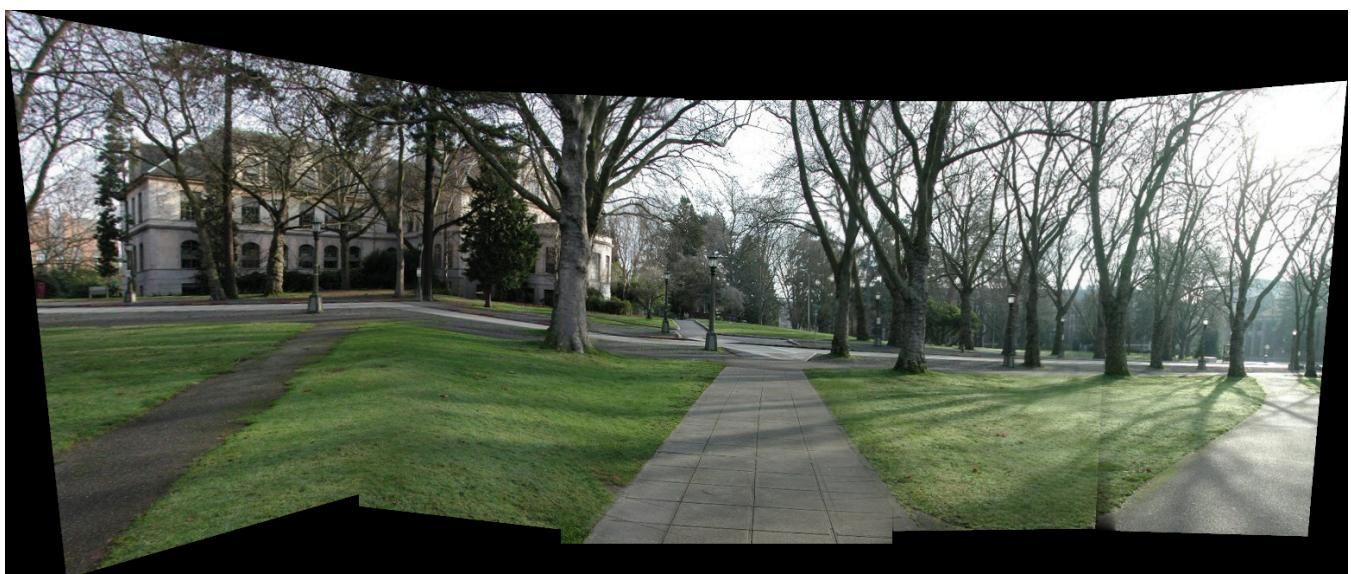
**panorama\_1**



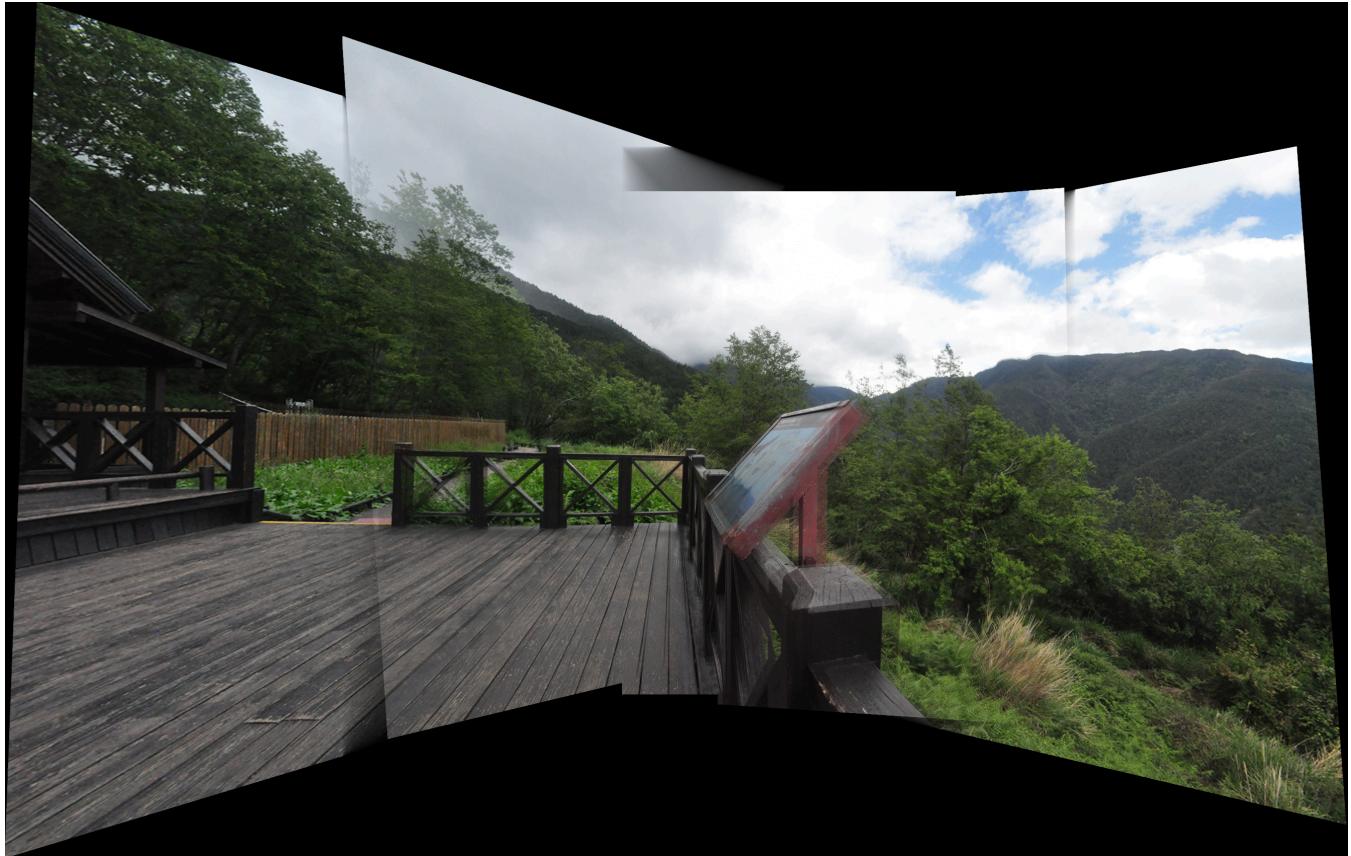
**panorama\_2**



**panorama\_3**



## panorama\_4



## 5.2 Analyze

### 1) 首先，以下是修改、调参过程中的简单试验总结：

- 第一个问题：`feature_matching` 连续两次运行或多次的结果极不稳定，比如下图的终端显示

```
命令提示符 E:\CV\Problem_Set_2>python homework2.py
----Start test_matching----
final rate: 0.0
Traceback (most recent call last):
  File "E:\CV\Problem_Set_2\homework2.py", line 423, in <module>
    test_matching()
  File "E:\CV\Problem_Set_2\homework2.py", line 193, in test_matching
    pixels_1, pixels_2 = feature_matching(img_1, img_2)
  File "E:\CV\Problem_Set_2\homework2.py", line 181, in feature_matching
    assert rate >= 0.03, "Fail to Match!"
AssertionError: Fail to Match!

E:\CV\Problem_Set_2>python homework2.py
----Start test_matching----
E:\CV\Problem_Set_2\homework2.py:173: RuntimeWarning: invalid value encountered in scalar divide
  if dis_min/np.min(dis[:, p]) <= threshold:
final rate: 0.46616541353383456
  test ending.

Make panorama for Xue-Mountain-Entrance begin.....
final rate: 0.0
Traceback (most recent call last):
  File "E:\CV\Problem_Set_2\homework2.py", line 430, in <module>
    MakePanorama_XueMountainEntrance()
  File "E:\CV\Problem_Set_2\homework2.py", line 415, in MakePanorama_XueMountainEntrance
    pano = generate_panorama(img_list)
  File "E:\CV\Problem_Set_2\homework2.py", line 317, in generate_panorama
    pixels1, pixels2 = feature_matching(ordered_img_seq[j], principle_img)
  File "E:\CV\Problem_Set_2\homework2.py", line 181, in feature_matching
    assert rate >= 0.03, "Fail to Match!"
AssertionError: Fail to Match!
```

-->根据报错显示，在`if dis_min/np.min(dis[p])<=threshold`可能出现除以0的情况，导致数值不稳定，对于`feature_matching`求出`dis`后，添加一个微小的量`1e-7`防止分母为0

```
R1 = harris_response(img_1, 0.04, 9)
R2 = harris_response(img_2, 0.04, 9)
cor1 = corner_selection(R1, 0.01*np.max(R1), 5)
cor2 = corner_selection(R2, 0.01*np.max(R1), 5)
fea1 = histogram_of_gradients(img_1, cor1)
fea2 = histogram_of_gradients(img_2, cor2)
dis = spatial.distance.cdist(fea1, fea2, metric='euclidean')
dis+=1e-7 #add
```

-->初始时`histogram_of_gradients`函数中的block size选择过小（开始时设为`block = 2~4`），特征提取不稳定，增加至16后效果明显提升。

- RANSAC迭代

初始，拼接grail的时候，呈现结果很不稳定，提高`align_pair`中的`max_iterate`到1000后效果明显改善（初始时仅取400左右）-->可能是迭代次数不足未收敛导致的不稳定

- 连续五张照片的选取

在全景图拼接时，有些照片的特征点可能不那么明显，匹配点不多、匹配出来rate很低，效果一般。通过选取图集中较好的连续几张图可以部分解决这个问题，比如在library的拼接中发现10~14比1~5的编号容易拼接许多。

- 超参数调整

本次作业中核心的参数调整围绕`feature_matching`的`threshold`展开，它的含义是：只有当图像1中某个角点与图像2中某个角点之间的最小距离和该角点与所有其他角点的距离的比例小于`threshold`时，才认为这两个角点是有效匹配的。初始`threshold=0.6`

以下是对1\_1 1\_2做`test_matching`时`threshold`依次取0.5/0.6/0.7/0.8/0.9的效果图，可以看出来，在`threshold`取不是很高的值的时候，随着`threshold`增加有效匹配点的对数也增加（连线数量更多），同时保持良好的匹配没有错误匹配角点；而这个阈值过高的时候（>=0.75时尤其明显），会对“最小距离与其他点距离的比值”要求放宽，放过较多的特征点，可能错误地配对了某些不应该匹配的点对（如最后两张图所示效果）

全景拼接时，尤其在library的拼接中，常在“`assert rate >= 0.03, "Fail to Match!"`”报错，调试输出显示后发现是`feature_matching`中留下的匹配点数量较少(library拼接中频繁有个位数)，选择适当提高`threshold`，可以在“留下更多匹配点提高`rate`”和“保证特征点匹配正确”中达到平衡。



