计算机视觉实践——练习 1_图像拼接实验报告

目录

1 实验目的

- 理解尺度变化不变特征 SIFT。
- 采集一系列局部图像，自行设计拼接算法。
- 使用 Python 实现图像拼接算法。

2 实验原理

主要包括特征点匹配、图像拼接 2 个部分。

2.1 特征点匹配

2.1.1 原理

特征点是图像中的显著、易于识别的位置，比如角点、边缘点等。在图像中提取特征点，是一种常见的图像处理方法，也是计算机视觉领域中的重要内容之一。本次实验使用的特征点提取算法是 SIFT (Scale-Invariant Feature Transform) 算法，该算法可以检测图像中的关键点，并描述它们的特征。通过特征点匹配可以找到两幅图像中对应的特征点，从而实现图像对齐和拼接。

2.1.2 执行步骤

1. 使用 SIFT 算法提取两幅图像的特征点。
2. 使用 Brute-Force 匹配器，对两幅图像中的特征点进行匹配。
3. 通过 KNN 算法筛选出最佳的特征点，并生成最佳的匹配。

2.2 图像拼接

图像拼接是将多幅图像拼接成一幅大图像的过程，是一种常用的图像处理方法。本次实验使用的图像拼接算法是基于单应性矩阵的方法。

2.2.1 原理

单应性矩阵是指对于一对图像中对应的点集，存在一个 3×3 的矩阵 H，使得两个图像中的对应点在同一平面上，其中一个图像上的点通过 H 的变换可以映射到另一个图像上的对应点，如图 1 所示。通过计算这个矩阵，可以实现图像的对齐和拼接。

$$\begin{bmatrix} x_2 \\ y_2 \\ z_2 \end{bmatrix} = \begin{bmatrix} H_{11} & H_{12} & H_{13} \\ H_{21} & H_{22} & H_{23} \\ H_{31} & H_{32} & H_{33} \end{bmatrix} \begin{bmatrix} x_1 \\ y_1 \\ z_1 \end{bmatrix} \Leftrightarrow \mathbf{x}_2 = H\mathbf{x}_1$$

图 1 单应性矩阵计算公式

其中，单应性矩阵的计算可以使用 RANSAC 算法来进行鲁棒性优化，避免

由于特征点匹配误差带来的不良影响。

2.2.2 执行步骤

1. 计算之前筛选出的最佳匹配点之间的单应性矩阵。
2. 使用单应性矩阵将第二幅图像进行仿射变换，使得第二幅图像与第一幅图像在同一平面上。
3. 生成图像掩膜，用于实现边缘的过渡。
4. 两幅图像分别乘以对应的掩膜并相加得到最终的拼接融合图像。

3 实验

3.1 实验细节

本次实验使用的语言为 Python，使用的库有 opencv、numpy、matplotlib。大部分算法是基于 opencv 已有的方法实现的，如 SIFT、单应性矩阵的生成等。

3.2 实验结果

输入图如下：

图 2 输入图

匹配图如下：

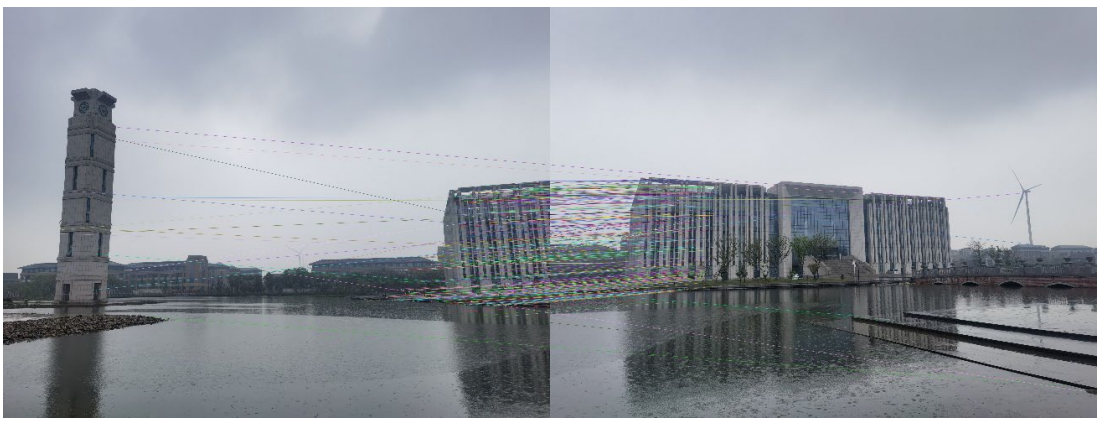

图 3 匹配图

拼接图如下：

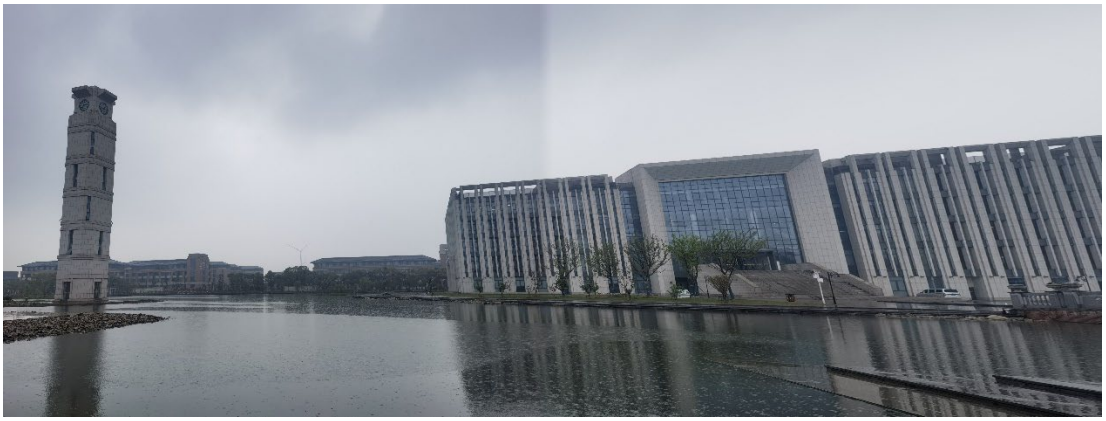

图 4 拼接图

3.3 实验分析

本次实验的拼接图结果基本达到实验目的要求，但还是可以看出部分的拼接线。

本次实验中，仅实现了两幅图片的拼接，且拼接图片的顺序得按照拍摄角度从左到右开始拼接，同时拼接存在部分拼接线，故还需继续完善才能达到多图无缝拼接的效果。

附录——代码展示

仅展示主要代码。

```python
class ImageStitching:
    def __init__(self, ratio=0.65, min_match=5, smoothing_window_size=100):
        self.ratio = ratio  # 特征点匹配的阈值
        self.min_match = min_match  # 最小匹配点数量
        self.smoothing_window_size = smoothing_window_size  # 融合部分的平滑窗大小
        self.sift = cv2.SIFT_create()  # 创建SIFT算子
```

```python
# 计算两张图片之间的单应性矩阵
def registration(self, img1, img2, output_path):
    # 使用SIFT获得特征点
    key_point1, des1 = self.sift.detectAndCompute(img1, None)
    key_point2, des2 = self.sift.detectAndCompute(img2, None)
    raw_matches = cv2.BFMatcher().knnMatch(des1, des2, k=2)  # 使用Brute-Force匹配器，对两幅图像中的特征点进行匹配
    best_features = []
    best_matches = []

    # 通过KNN（K=2）获得最佳特征点
    for m1, m2 in raw_matches:
        if m1.distance < self.ratio * m2.distance:
            best_features.append((m1.trainIdx, m1.queryIdx))
            best_matches.append([m1])
    img3 = cv2.drawMatchesKnn(img1, key_point1, img2, key_point2, best_matches, None, flags=2)

    cv2.imwrite(output_path + os.sep + 'matching.jpg', img3)  # 输出特征点匹配图

    # 如果匹配的特征点数量超过了指定的最小匹配数，该方法将返回单应性矩阵
    if len(best_features) > self.min_match:
        image1_kp = np.float32(
            [key_point1[i].pt for (_, i) in best_features])
        image2_kp = np.float32(
            [key_point2[i].pt for (i, _) in best_features])
        # 计算单应性矩阵，加入了Ransac
        h, status = cv2.findHomography(image2_kp, image1_kp, cv2.RANSAC, 5.0)
        return h
    else:
        print(f'两张图的特征点小于{self.min_match}，请检查两张图是否有关联。')
        exit(1)
```

```python
# 创建融合图像时的掩膜（mask）
def create_mask(self, img1, img2, side):
    height_blended = img1.shape[0]
    width_blended = img1.shape[1] + img2.shape[1]
    offset = int(self.smoothing_window_size / 2)
    barrier = img1.shape[1] - int(self.smoothing_window_size / 2)
    mask = np.zeros((height_blended, width_blended))

    # 左右图使用不同的masK
    if side == 'left':
        mask[:, barrier - offset:barrier + offset] = np.tile(np.linspace(1, 0, 2 * offset).T, (height_blended, 1))
        mask[:, :barrier - offset] = 1
    else:
        mask[:, barrier - offset:barrier + offset] = np.tile(np.linspace(0, 1, 2 * offset).T, (height_blended, 1))
        mask[:, barrier + offset:] = 1
    return cv2.merge([mask, mask, mask])
```

```python
# 图像融合拼接
def blending(self, img1, img2, output_path):
    # 获得单应性矩阵
    h = self.registration(img1, img2, output_path)
    height_blended = img1.shape[0]
    width_blended = img1.shape[1] + img2.shape[1]

    # 进行图片仿射变换与融合
    blend1 = np.zeros((height_blended, width_blended, 3))
    blend1[0:img1.shape[0], 0:img1.shape[1], :] = img1
    blend1 *= self.create_mask(img1, img2, 'left')
    blend2 = cv2.warpPerspective(img2, h, (width_blended, height_blended)) * self.create_mask(img1, img2, 'right')
    result = blend1 + blend2

    rows, cols = np.where(result[:, :, 0] != 0)
    result_image = result[min(rows):max(rows) + 1, min(cols):max(cols) + 1, :]
    return result_image
```

```python
# 拼接结果展示
def stitching(self, images, output_path):
    img_stitched = self.blending(images[0], images[1], output_path)
    cv2.imwrite(output_path + os.sep + 'output.jpg', img_stitched)
    plt.imshow(img_stitched[:,:,::-1].astype('uint8'))
```