CVPR 第一次作业　图像对齐 & 图像拼接

吴天阳　*4124136039*　人工智能学院 *B2480*

1　实验目的

1. 掌握图像的 SIFT 特征检测原理；
2. 掌握图像特征描述子的匹配度量（距离比），RANSAC 方法；
3. 完成图像的 SIFT 特征提取与匹配；
4. 完成基于单应矩阵 H（2D 射影变换）的图像视点变换与拼接；讨论图像融合方法与鲁棒匹配/估计方法，以及多单应矩阵的图像拼接.

2　实验原理

2.1　SIFT 特征检测的主要步骤

2.1.1　尺度空间极值检测

构建尺度空间是为了在多尺度下检测关键点. 尺度空间通过高斯模糊函数生成，定义如下：

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

其中，$G(x, y, \sigma)$ 是高斯核函数：

$$G(x, y, \sigma) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}},$$

$I(x, y)$ 是输入图像，$*$ 表示卷积操作.

为了检测尺度空间中的关键点，计算高斯差分（DoG）：

$$D(x, y, \sigma) = L(x, y, k\sigma) - L(x, y, \sigma),$$

其中 k 为尺度系数.

然后，通过在空间和尺度上的邻域内寻找局部极值点.

2.1.2　关键点定位与过滤

对检测到的极值点进行精确定位，通过泰勒展开近似求解亚像素级关键点位置. 目标函数 $D(x, y, \sigma)$ 在关键点位置进行二阶导数测试以评估关键点稳定性，剔除低对比度点和边缘响应点.

2.1.3　方向分配

为每个关键点分配一个或多个方向，用以实现旋转不变性. 通过计算关键点邻域的梯度幅值和方向，构造方向直方图：

$$m(x, y) = \sqrt{(L_x)^2 + (L_y)^2}, \quad \theta(x, y) = \tan^{-1}\left(\frac{L_y}{L_x}\right),$$

其中，L_x 和 L_y 为图像梯度.

方向直方图的主方向对应关键点的主方向.

2.1.4 关键点描述符生成

在关键点邻域内生成描述符. 将邻域划分为 4×4 的网格，每个子网格计算 8 个方向的梯度直方图，形成长度为 128 的特征向量.

2.2 匹配度量：距离比

在图像特征匹配中，特征点的描述子通常通过欧几里得距离衡量相似性. 假设两幅图像中的特征描述子集合分别为 $\mathbf{D}_1 = \{\mathbf{d}_1^i\}$ 和 $\mathbf{D}_2 = \{\mathbf{d}_2^j\}$，描述子 $\mathbf{d}_1^i$ 和 $\mathbf{d}_2^j$ 的匹配度量可以表示为

$$\text{dist}(\mathbf{d}_1^i, \mathbf{d}_2^j) = \|\mathbf{d}_1^i - \mathbf{d}_2^j\|_2.$$

为了提高匹配的准确性，通常采用距离比（Ratio Test）方法. 定义最邻近和次邻近描述子的距离分别为 d_1 和 d_2，则当满足

$$\frac{d_1}{d_2} < \tau,$$

其中 τ 为经验阈值（通常取 0.7），即可认为该匹配是有效的.

2.3 RANSAC 方法

在匹配点对中，通常存在一定数量的错误匹配（outliers）. 为了鲁棒地估计图像之间的变换关系（如单应矩阵或基础矩阵），可以采用随机抽样一致性（RANSAC）算法. RANSAC 的基本步骤如下：

1. 从匹配点集中随机抽取最小子集，计算模型参数（例如，单应矩阵 **H** 或基础矩阵 **F**）.
2. 根据模型参数，计算其他匹配点对是否符合模型（即是否为内点，通常通过投影误差 error $< \epsilon$ 判断）.
3. 重复上述步骤，直到达到预定的迭代次数或找到最佳模型.
4. 返回内点最多的模型作为最终结果.

RANSAC 的鲁棒性来自于内点的比例. 如果内点比例为 p，需要采样的最小次数 N 满足

$$N = \frac{\log(1-P)}{\log(1-p^s)},$$

其中 P 是期望的模型可靠性，s 是每次采样的点数.

2.4 单应矩阵 H 的计算与应用

单应矩阵 H 描述了图像之间的二维射影变换，其形式为：

$$\begin{bmatrix} x' \\ y' \\ w' \end{bmatrix} = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix},$$

其中 $\mathbf{p}' = (x', y')$ 为目标图像中的像素点，$\mathbf{p} = (x, y)$ 为源图像中的对应点.

2.4.1 视点变换

通过 H，可以将图像从一个视角投影到另一个视角，完成视点变换. 假设有源图像 I_s 和目标图像 I_t，变换后的图像 I'_s 可通过：$\mathbf{p}' = H\mathbf{p}$ 得到.

2.4.2 图像拼接

拼接图像时，需将多幅图像变换到同一全局坐标系下. 假设 H_i 是第 i 幅图像的变换矩阵，则拼接结果 I_f 表示为：

$$I_f = \sum_i W(I_i, H_i),$$

其中 $W(I_i, H_i)$ 为将 I_i 按 H_i 变换后的结果.

2.5 图像融合方法

图像拼接过程中可能产生接缝，为此可采用以下融合方法：

- **直接叠加法**：简单加权平均相邻区域像素值，公式为：

$$I_f(x, y) = \alpha I_1(x, y) + (1 - \alpha) I_2(x, y), \quad \alpha \in [0, 1].$$

- **渐变融合**：在重叠区域使用线性渐变权重，避免边缘突兀.
- **多频段融合**：将图像分解为不同频率成分（如高斯金字塔），分别融合再重构.

2.6 鲁棒匹配与估计方法

- **匹配方法**：利用描述子（如 SIFT、ORB）进行特征匹配，通过距离比筛选或交叉验证提高匹配质量.
- **鲁棒估计方法**：在单应矩阵估计中使用 RANSAC 算法，以处理匹配点中的外点，估计最优 H.

2.7 多单应矩阵的拼接

当图像场景具有复杂的几何结构（如全景场景），单个单应矩阵无法准确建模. 此时需对图像分块，并分别估计各块的单应矩阵. 假设图像被分为 n 个区域，每个区域的变换矩阵为 H_i，全局拼接公式为：

$$I_f = \sum_{i=1}^{n} W(I_i, H_i).$$

3 实验步骤与结果分析

3.1 SIFT 特征检测

使用 Python 中的 `cv2.SIFT_create()` 高效完成 SIFT 关键点检测，代码如下：

```
# 1. 初始化 SIFT 检测器
sift = cv2.SIFT_create()

# 2. 检测关键点并计算特征描述符
keypoints, descriptors = sift.detectAndCompute(image, None)

```

```
# Output the number of keypoints and the shape of the descriptors
print(f"Number of keypoints detected: {len(keypoints)}")
print(f"Shape of descriptors: {descriptors.shape}")

# 3. 绘制关键点
image_with_keypoints = cv2.drawKeypoints(
    image,
    keypoints,
    None,
    flags=cv2.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS
)
```

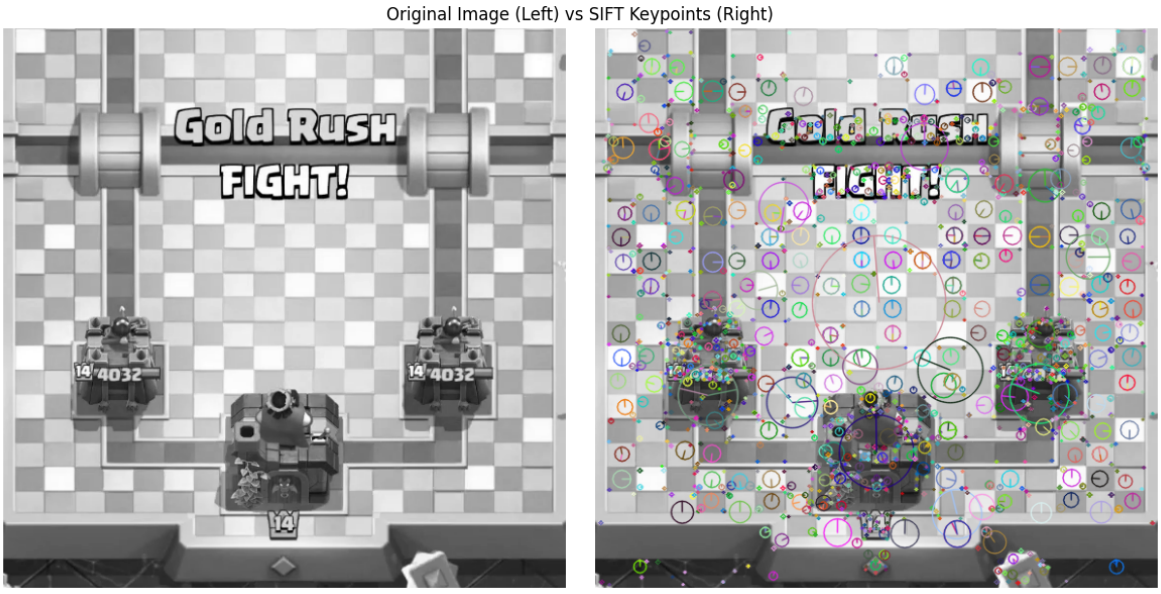

图 1: 左图灰度原图，右图 SIFT 关键点结果

3.2 特征描述子匹配和 RANSAC 方法

```
# 1. 初始化 SIFT 检测器
sift = cv2.SIFT_create()

# 2. 检测关键点并计算描述子
keypoints1, descriptors1 = sift.detectAndCompute(image1, None)
keypoints2, descriptors2 = sift.detectAndCompute(image2, None)

# 3. 使用 FLANN(快速近似最近邻) 匹配描述子，使用 KD 树算法快速匹配描述子
flann_index_kdtree = 1
index_params = dict(algorithm=flann_index_kdtree, trees=5)
search_params = dict(checks=50)

flann = cv2.FlannBasedMatcher(index_params, search_params)
matches = flann.knnMatch(descriptors1, descriptors2, k=2)

# 4. 距离比筛选 (Lowe's Ratio Test)
good_matches = []
```

```
for m, n in matches:
    if m.distance < 0.7 * n.distance:  # 距离比阈值
        good_matches.append(m)

# 5. 提取匹配点
src_pts = np.float32([keypoints1[m.queryIdx].pt for m in good_matches]).reshape(-1, 1, 2)
dst_pts = np.float32([keypoints2[m.trainIdx].pt for m in good_matches]).reshape(-1, 1, 2)

# 6. 使用 RANSAC 方法估计单应性矩阵
H, mask = cv2.findHomography(src_pts, dst_pts, cv2.RANSAC, 5.0)
matches_mask = mask.ravel().tolist()

# 7. 可视化匹配结果
draw_params = dict(matchColor=(0, 255, 0),  # 内点为绿色
                   singlePointColor=(255, 0, 0),  # 关键点为蓝色
                   matchesMask=matches_mask,  # 仅显示内点
                   flags=cv2.DrawMatchesFlags_DEFAULT)

result_image = cv2.drawMatches(image1, keypoints1, image2, keypoints2, good_matches, None, **draw_params)
```

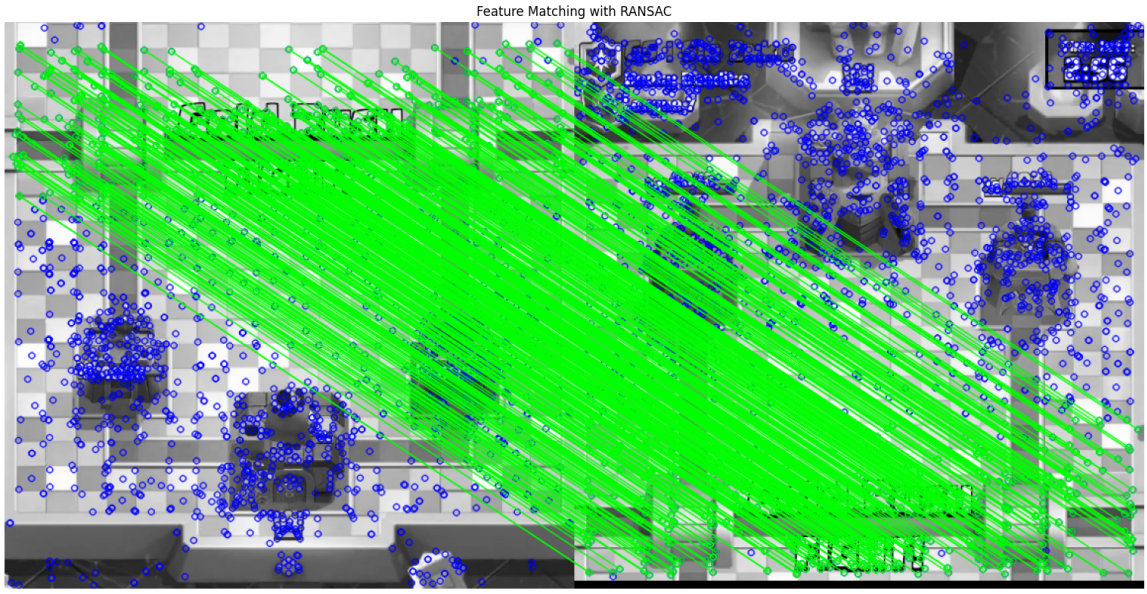

图 2: 特征描述子匹配效果（两幅类似背景的图片，在不同时刻下不同位置处的截图）

3.3 图像融合

```
# 基于 RANSAC 得到的单应性矩阵 H
# 将第一张图像进行单应性变换，配准到第二张图像，计算输出图像的大小（可以容纳两张图像）
result = cv2.warpPerspective(image1, H, (width + image1.shape[1], height))

# 将第二张图像放入结果图像中
```

```
result[0:height, 0:width] = image2

# 在第二张图像（image2）周围画一个绿色的矩形框
cv2.rectangle(result, (0, 0), (width, height), (0, 255, 0), 5)

# 对第一张图像的四个角点应用单应性变换，得到变换后的矩形框
pts = np.float32([[0, 0], [image1.shape[1], 0], [image1.shape[1], image1.shape[0]], [0, image1.shape[0]]]).reshape(-1, 1, 2)
pts_transformed = cv2.perspectiveTransform(pts, H)

# 在变换后的图像中画出第一张图像的矩形框（蓝色）
pts_transformed = np.int32(pts_transformed)
cv2.polylines(result, [pts_transformed], isClosed=True, color=(255, 0, 0), thickness=5)
```

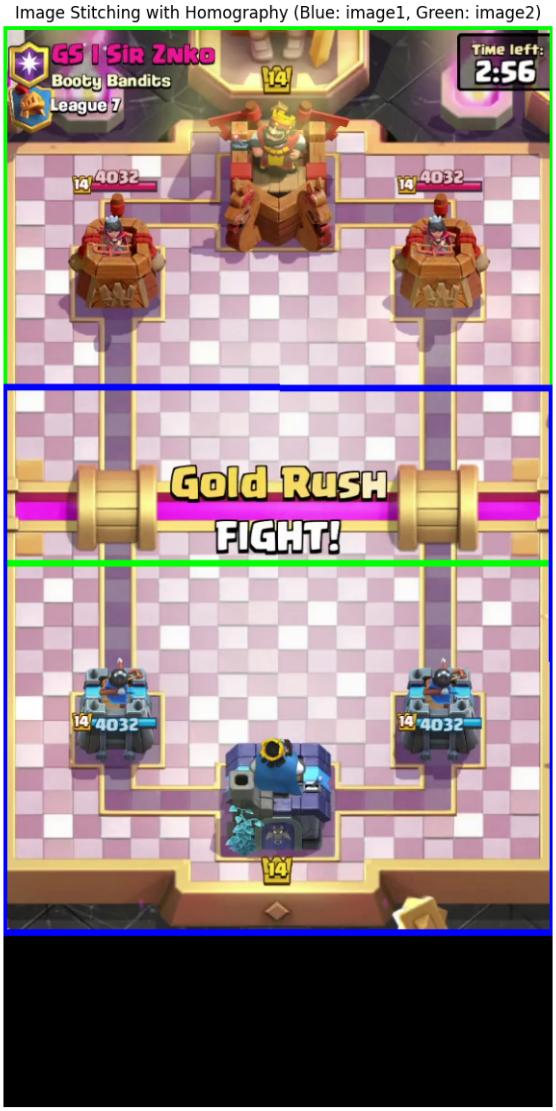

图 3: 图像融合结果，蓝色为图2中的左图，绿色为图2中的右图