实验三

一、实验目的

通过 SIFT 算法提取并匹配图像特征点进行几何变换，结合 RANSAC 算法实现图像配准与拼接，完成对图片的拼接。

二、实验内容

1.实验环境搭建与图像数据准备：配置 OpenCV 等计算机视觉库，选取 2 张具有重叠区域的图像作为实验素材；

2.SIFT 特征提取与描述：利用 SIFT 算法提取两张图像的特征关键点及对应的描述子；

3.特征点匹配与筛选：通过 FLANN 匹配器完成特征点初步匹配，再用 Lowe's 比率测试筛选优质匹配对；

4.单应矩阵估计与图像变换：借助 RANSAC 算法剔除误匹配点（外点），估计图像间的单应矩阵（透视变换矩阵），并基于该矩阵对其中一张图像进行透视变换；

5.图像拼接与结果可视化：将变换后的图像与另一张原图进行拼接融合，同时可视化特征匹配过程、拼接结果

三、实验原理

SIFT 特征提取：尺度不变特征变换（SIFT）算法能在不同尺度、旋转、光照条件下提取图像的关键点，并生成 128 维的描述子，用于表征关键点的局部特征，保障特征的唯一性与鲁棒性。

特征匹配与筛选：

FLANN 匹配器：通过快速最近邻搜索算法，高效匹配两张图像的 SIFT 描述子，得到初始匹配对；

RANSAC 与单应矩阵：随机抽样一致（RANSAC）算法通过多次随机选取少量匹配点估

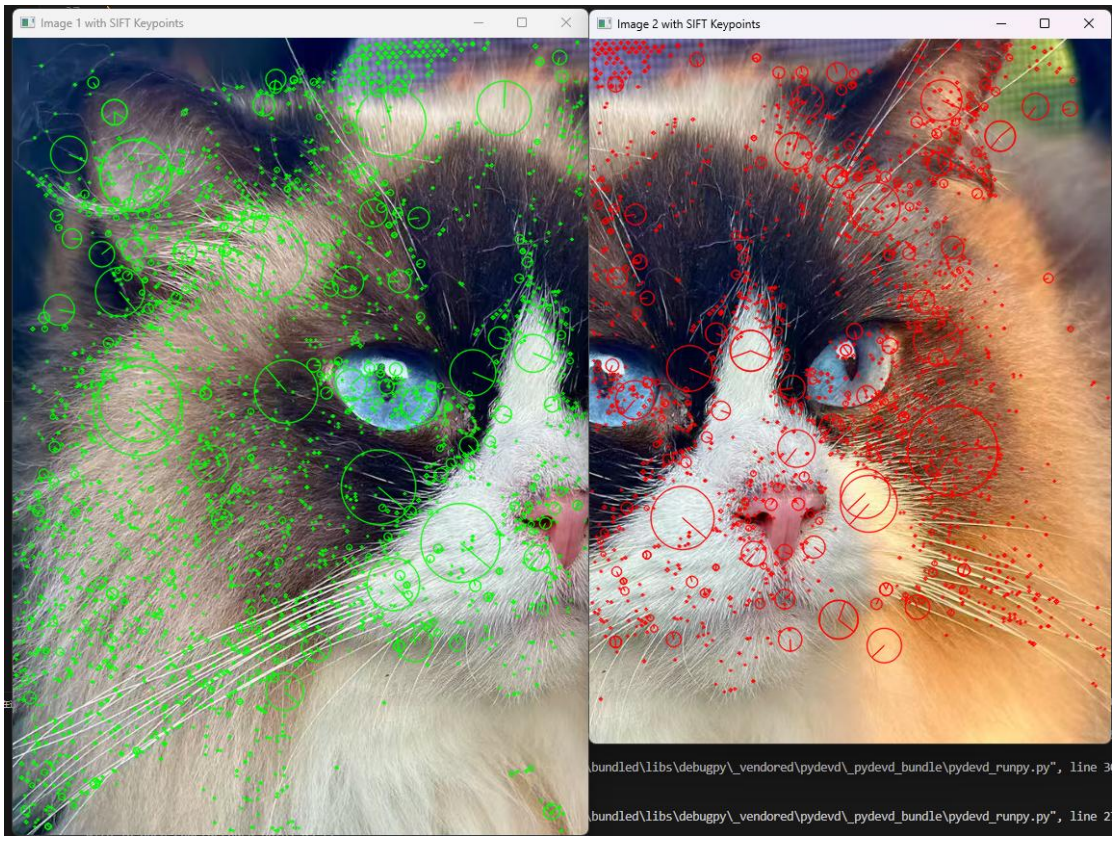
计单应矩阵（透视变换矩阵），并统计符合该矩阵的内点数量，最终得到鲁棒的单应矩阵（可剔除误匹配外点）；单应矩阵用于将其中一张图像映射到另一张图像的坐标系，实现图像配准。

四、实验步骤

1. SIFT 特征提取

```
##初始化SIFT检测器并提取特征
sift = cv2.SIFT_create()  # 文档中此处为Cv2（大写C），修正为cv2（小写c）
kp_a, des_a = sift.detectAndCompute(gray_a, None)  # 图像a的关键点和描述符
kp_b, des_b = sift.detectAndCompute(gray_b, None)  # 图像b的关键点和描述符
```

特征点可视化结果如图所示:

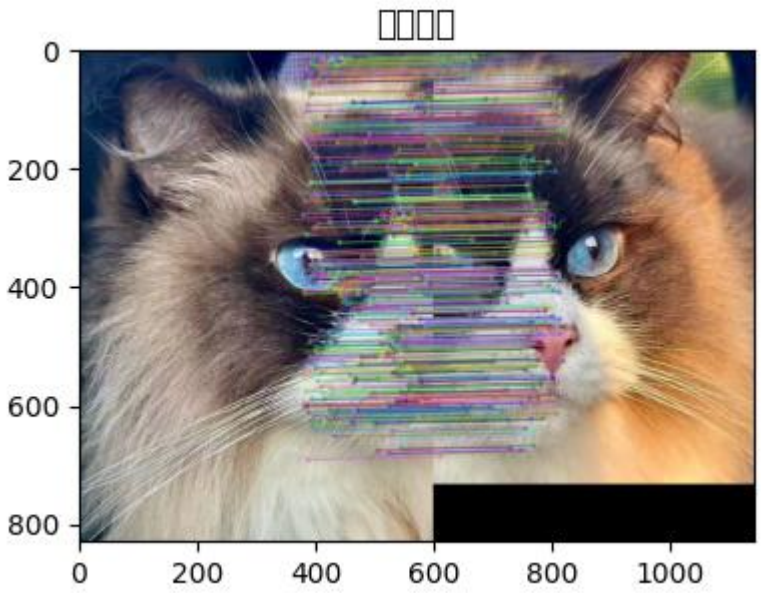


2. 特征匹配和筛选

```python
cv2.waitKey(0)
cv2.destroyAllWindows()
# 使用FLANN匹配器进行特征匹配
FLANN_INDEX_KDTREE = 1
index_params = dict(algorithm=FLANN_INDEX_KDTREE, trees=5)  # 文档中FLANN后多空格，已修正
search_params = dict(checks=50)  # 检查次数越多，匹配越准确但速度越慢
flann = cv2.FlannBasedMatcher(index_params, search_params)

matches = flann.knnMatch(des_a, des_b, k=2)  # k=2表示每个特征点返回2个最佳匹配
# 应用Lowe's比率测试筛选优质匹配点
good_matches = []
for m, n in matches:
    if m.distance < 0.7 * n.distance:  # 比率阈值通常取0.7-0.8
        good_matches.append(m)
# 基本诊断输出
print(f"图像A关键点: {len(kp_a)}, 图像B关键点: {len(kp_b)}")
print(f"Lowe 筛选后匹配数: {len(good_matches)}")
```

特征匹配可视化如图所示：



3. 单应矩阵估计与图像变换

```python
# 使用RANSAC算法估计单应矩阵(透视变换矩阵)
H, mask = cv2.findHomography(src_pts, dst_pts, cv2.RANSAC, 5.0)

# mask为内点掩码，统计内点数量并在匹配图中只显示内点
if mask is None:
    print("findHomography 未返回掩码，可能匹配不足或失败。")
    inlier_count = 0
else:
    inlier_count = int(mask.sum())
print(f"RANSAC 内点数量: {inlier_count} / {len(good_matches)}")

# 如果内点过少，提示并提前退出
if inlier_count < 4:
    raise RuntimeError("内点太少，无法计算可靠的单应矩阵（需要至少4个内点）。")

# 使用内点重新绘制匹配图（只显示内点匹配）
inlier_matches = [gm for gm, m in zip(good_matches, mask.ravel()) if m]
matched_keypoints_img = cv2.drawMatches(
    img_a, kp_a, img_b, kp_b, inlier_matches, None,
    flags=cv2.DrawMatchesFlags_NOT_DRAW_SINGLE_POINTS
)
```

这一步的核心是通过单应矩阵实现两张图像的几何对齐，并通过可视化直观展示这一过程，为最终拼接做准备，主要包含两个关键部分：

单应矩阵估计与内点筛选利用 RANSAC 算法从之前筛选出的特征匹配点中，找到能描述两张图像透视变换关系的单应矩阵（M）。过程中会区分 “内点”（绿色匹配线，符合几何约束的可靠匹配）和 “外点”（红色匹配线，偏离约束的误匹配），剔除外点后得到更鲁棒的变换矩阵。

图像透视变换与对齐可视化用得到的单应矩阵（M）对其中一张图像（如 img_a）做透视变换，将其 “映射” 到另一张图像（如 img_b）的坐标系下。通过对比变换前后的图像、以及变换后图像与目标图像的半透明叠加效果，直观展示两张图像的重叠区域是否对齐 —— 对齐越好，说明单应矩阵越准确，为后续拼接奠定基础。

单应矩阵：

```
print("单应矩阵 H:\n", H)
try:
    cond_H = np.linalg.cond(H)
except Exception:
    cond_H = float('inf')
print(f"H 条件数: {cond_H}")
```

单应矩阵是一个 3×3 的矩阵，它能精确描述 “一张图像上的点如何通过透视变换（比如旋转、缩放、视角调整）映射到另一张图像上”。因为特征匹配中难免有错误匹配（外点），所以用 RANSAC 算法从匹配点中 “挑出” 大多数正确的点（内点），计算出它们共同符合的单应矩阵 —— 这样得到的矩阵能更可靠地反映两张图的几何关系。

图像变换：

```
# 获取输入图像尺寸
h_a, w_a = img_a.shape[:2]
h_b, w_b = img_b.shape[:2]  # 文档中为wb，修正为w_b

# 计算图像b变换后的四个角点坐标
pts = np.float32([[0, 0], [0, h_b], [w_b, h_b], [w_b, 0]]).reshape(-1, 1, 2)  # 文档中为w_b,e，修正为w_b,0
dst_corners = cv2.perspectiveTransform(pts, H)

# 确定拼接后图像的最终尺寸(包含所有像素)
all_corners = np.concatenate([
    dst_corners,
    np.float32([[0, 0], [w_a, 0], [w_a, h_a], [0, h_a]]).reshape(-1, 1, 2)  # 文档中缺失部分坐标值，已补充
], axis=0)
[x_min, y_min] = np.int32(all_corners.min(axis=0).ravel() - 0.5)
[x_max, y_max] = np.int32(all_corners.max(axis=0).ravel() + 0.5)

# 创建平移矩阵，确保所有像素都在可见区域内
translation_matrix = np.array([[1, 0, -x_min], [0, 1, -y_min], [0, 0, 1]], dtype=np.float32)  # 文档中缺失0，已补充

# 对图像进行透视变换和平移
fus
    (variable) img_b: Mat | ndarray[Any, dtype[integer[Any] | floating[Any]]]
    img_b,
    translation_matrix @ H,  # 组合平移矩阵和单应矩阵
    (x_max - x_min, y_max - y_min)  # 输出图像尺寸
)
```

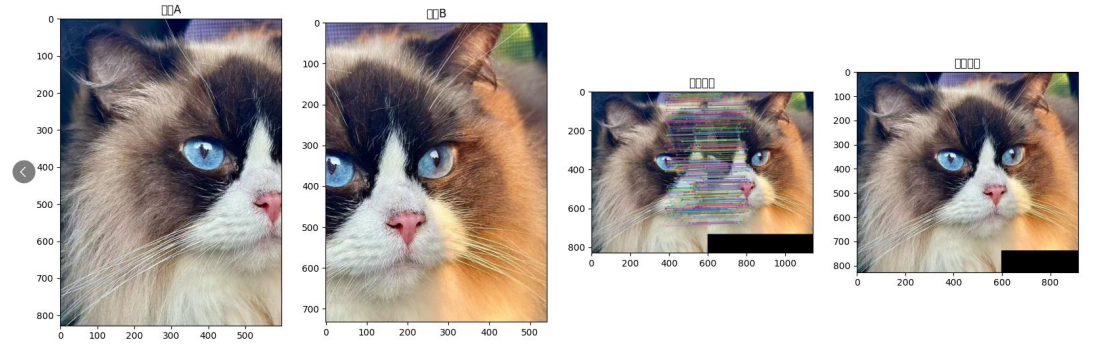
4. 拼接和可视化

```
# 将图像a复制到拼接结果的对应位置
fus_img[-y_min:h_a - y_min, -x_min:w_a - x_min] = img_a  # 文档中为xmin，修正为x_min

# 显示匹配关键点和拼接结果
plt.figure(figsize=(20, 20))
plt.subplot(1, 4, 1)
plt.imshow(cv2.cvtColor(img_a, cv2.COLOR_BGR2RGB))
plt.title('图像A')
plt.subplot(1, 4, 2)
plt.imshow(cv2.cvtColor(img_b, cv2.COLOR_BGR2RGB))
plt.title('图像B')
plt.subplot(1, 4, 3)
plt.imshow(cv2.cvtColor(matched_keypoints_img, cv2.COLOR_BGR2RGB))
plt.title('特征匹配')
plt.subplot(1, 4, 4)
plt.imshow(cv2.cvtColor(fus_img, cv2.COLOR_BGR2RGB))
plt.title('拼接结果')
plt.show()  # 显示所有图像
```

五、实验结果与分析

结果如图所示：



分析：

拼接图的黑框，核心是透视变换让像素坐标偏移（比如出现负坐标或超出原图尺寸），为装下这些偏移像素必须扩大图像，而扩大的空白区域会被默认填黑。

分辨率差异会让黑框更明显(比如低分辨率图变换到高分辨率图坐标系时,需更大图像尺寸)，但不是黑框产生的根本原因。