计算机视觉应用与实践（四）

目录

一、 实验目的

理解单应性矩阵的原理，并编程实现图像之间的单应性矩阵的计算。

二、 实验原理

单应性矩阵（Homography Matrix），也称为单应变换矩阵，是指在平面的任意两幅图像之间进行投影变换时所使用的矩阵。它可以将一张图像上的任意四个点对应到另一张图像上的对应四个点，从而实现图像的对齐和融合等操作。在计算机视觉领域，单应性矩阵常常用于图像拼接、目标跟踪、立体匹配等任务中。单应性矩阵包含了两个平面之间的几何关系，可以通过一定的算法从一组对应点中求解出来。

具体来说，单应性矩阵是一个 3*3 大小的矩阵，如下所示。

$$H = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix}$$

假设(u, v)和(x, y)分别是两幅图像中的匹配点，根据单应性矩阵的定义，我们可以得到：

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = H \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

将上式展开可得：

$$u = \frac{h_{11}x + h_{12}y + h_{13}}{h_{31}x + h_{32}y + h_{33}}$$
$$v = \frac{h_{21}x + h_{22}y + h_{23}}{h_{31}x + h_{32}y + h_{33}}$$

进一步展开可得：

$$h_{11}x + h_{12}y + h_{13} + h_{21} * 0 + h_{22} * 0 + h_{23} * 0 - h_{31}xu - h_{32}yu - h_{33}u = 0$$

$$h_{11} * 0 + h_{12} * 0 + h_{13} * 0 + h_{21}x + h_{22}y + h_{23} - h_{31}xv - h_{32}yv - h_{33}v = 0$$

转化为矩阵的形式：

$$\begin{bmatrix} x\ y\ 1\ 0\ 0\ 0\ -xu\ -yu\ -u \\ 0\ 0\ 0\ x\ y\ 1\ -xv\ -yv\ -v \end{bmatrix} * [h_{11}\ h_{12}\ h_{13}\ h_{21}\ h_{22}\ h_{23}\ h_{31}\ h_{32}\ h_{33}]^T = 0$$

上面的式子可以等价为$AH = 0$，显然一对匹配点可以提供两个约束条件，而单应性矩阵的自由度为 8，所以至少需要 4 对点才能求解出单应性矩阵。在实际计算中，我们可以通过选取 4 对点构成方程组并使用 SVD 求解单应性矩阵。

RANSAC（Random Sample Consensus）是一种迭代方法，用于从一组带有噪声和异常值的数据中估计出最佳模型的参数。它的基本思想是从数据中随机选择一组子集，这些子集被认为是符合模型的数据，然后使用这些子集拟合模型并计算出模型的误差。然后，将所有其他数据与模型进行比较，将其分为内点（符合模型的数据）和外点（不符合模型的数据）。如果内点的数量达到了一定的阈值，则认为当前模型是可接受的，并且使用所有内点来重新拟合模型。这个过程会重复执行多次，每次得到一个新的模型，并将最好的模型作为最终结果返回。

三、 程序代码

首先，我们先通过 SIFT 算法检测关键点和计算描述符，然后通过 knnMatch 匹配关键点，并去掉部分误匹配点。代码如下所示：

```
# 使用 opencv 创建 SIFT 对象
sift = cv2.xfeatures2d.SIFT_create()
# 检测关键点和计算描述符
kp1, des1 = sift.detectAndCompute(img1, None)
kp2, des2 = sift.detectAndCompute(img2, None)
bf = cv2.BFMatcher()
matches = bf.knnMatch(des1, des2, k=2)

good_matches = []
for m, n in matches:
    if m.distance < 0.75 * n.distance:
```

```
        good_matches.append(m)
# 绘制匹配结果
img_matches = cv2.drawMatches(img1, kp1, img2, kp2, good_matches, None,
 flags=2)
showImage(img_matches)
```

对于得到的匹配点，我们使用 RANSAC 方法迭代计算单应性矩阵，具体来说，每次从匹配点集合中随机抽取 4 对点，转化为方程组并使用 SVD 求解，对于得到单应性矩阵，计算变换后点与真实点之间的距离并统计内点个数，以得到最优的单应性矩阵。

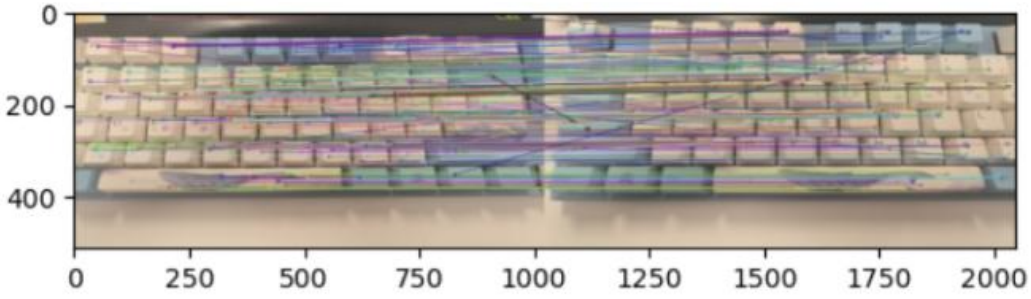
```
def find_homography(src_pts, dst_pts):
    # Reshape the input points
    src_pts = src_pts.reshape(-1, 2)
    dst_pts = dst_pts.reshape(-1, 2)

    # Construct the A matrix for the homography
    A = []
    for i in range(src_pts.shape[0]):
        x, y = src_pts[i]
        u, v = dst_pts[i]
        A.append([x, y, 1, 0, 0, 0, -u*x, -u*y, -u])
        A.append([0, 0, 0, x, y, 1, -v*x, -v*y, -v])
    A = np.asarray(A)

    # Compute the SVD of A
    U, s, Vt = np.linalg.svd(A)
    H = Vt[-1, :].reshape(3, 3)

    # Normalize the homography
    H = H / H[2, 2]

    return H
# 使用 RANSAC 算法计算单应性矩阵
def ransac_homography(src_pts, dst_pts, threshold=5.0, max_iterations=1
000):


    best_H = None  # 存储最好的单应性矩阵
    inliers = []   # 存储最好的内点

    for i in range(max_iterations):
        # 从源点和目标点中随机选择 4 个点
        indices = random.sample(range(src_pts.shape[0]), 4)
```

```
        src_sample = src_pts[indices]
        dst_sample = dst_pts[indices]

        # 计算这4个点对应的单应性矩阵
        H = find_homography(src_sample, dst_sample)

        # 使用当前的单应性矩阵计算所有的点
        projected_dst = cv2.perspectiveTransform(src_pts, H)

        # 计算所有点和其对应点之间的距离
        distances = np.sqrt(np.sum(np.square(projected_dst - dst_pts),
axis=2))

        # 统计内点个数
        current_inliers = np.where(distances < threshold)[0]

        # 如果当前的内点数大于历史最好内点数，则更新历史最好内点数和单应性矩阵
        if len(current_inliers) > len(inliers):
            best_H = H
            inliers = current_inliers

    return best_H, inliers
M,_ = ransac_homography(src_pts, dst_pts, 5.0)
```

四、 实验结果

原始图像

特征点匹配结果



使用 cv2.findHomography 的计算结果

```
cv2库计算单应性矩阵
[[ 1.09615903e+00  2.80916121e-02  2.84607011e+02]
 [ 1.00491211e-02  1.07220698e+00 -3.91761098e+01]
 [ 1.08803418e-05 -7.16214218e-05  1.00000000e+00]]
内点包含率43.57%
```

使用自己实现的函数的计算结果

```
自己实验计算单应性矩阵
[[ 1.09011513e+00  6.88578940e-02  2.82594595e+02]
 [ 2.19240030e-03  1.09515275e+00 -3.85838212e+01]
 [-9.36778885e-06 -6.05459054e-06  1.00000000e+00]]
内点包含率44.81%
```

可以看出，使用 cv2.findHomography 的函数内点包含率要低于使用自己实现函数的内点包含率，这表明后者计算得出的单应性矩阵更能拟合匹配点对。出现这种情况一方面可能是因为 RANSAC 算法的参数设置比 opencv 更加合理，另一

方面可能是因为 opencv 为了提高运行速度使用其他求解单应性矩阵的方法，从而丢失了一定的准确度。

五、 实验总结

通过这次实验，我学习了单应性矩阵的求解方法以及相关算法，并编程实现了使用 RANSAC 和 SVD 求解两张图像之间的单应性矩阵。这对于我今后的学习和研究都非常有帮助。