计算机视觉与应用实践实验报告（四）

目录

一. 实验内容

计算图片之间的单应性变换矩阵，并对结果进行分析。

二. 实验原理

完成图片之间的单应性矩阵计算，首先需要找到两张图像中的匹配点对（至少不共线的 4 对），所以下面介绍实验中使用的图像特征点检测算法 SIFT 和特征点匹配方法，以及使用匹配点对计算图像之间的单应性变换矩阵的计算方法及原理。

2.1 SIFT 算法

尺度不变特征转换检测算法 SIFT(Scale-Invariant Feature Transform)用于查找图像中的局部特征点，这些点不受图像大小、平移和旋转等变换因素的影响。可分解为四步完成：尺度空间极值点检测，关键点定位，方向确定和关键点描述。

2.1.1 尺度空间极值点检测

以构建尺度空间用于原始图像的表示为基础，在不同的尺度空间下查找极值点，通过查询到的极值点信息锁定像素点位置，并将其作为候选的图像特征点。

- **生成尺度空间：**高斯卷积核是进行尺度变换的唯一变换核，一个二维图

像的尺度空间定义为：

$$L(x,y,\sigma) = G(x,y,\sigma) * I(x,y) \tag{2.1}$$

式中$G(x,y,\sigma)$是高斯核函数：

$$G(x,y,\sigma) = \frac{\frac{1}{2\pi\sigma^2}e^{-(x^2+y^2)}}{2\sigma^2} \tag{2.2}$$

式中，*表示卷积操作，(x, y)表示图像中像素的位置，σ表示尺度空间的因子。

采样构造 N 组(Octave)不同尺寸的图层、通过高斯核函数平滑构造组内 S 层不同模糊度的图像，如图 2.1 所示，完成多尺度高斯金字塔的构造。

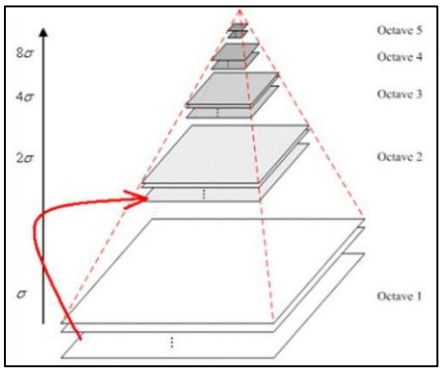

图 2. 1 高斯金字塔构建过程

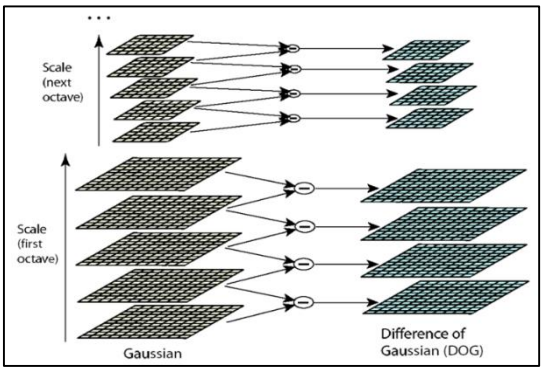

图 2. 2 高斯差分金字塔构建过程

为了在不同尺度空间检测到相应的稳定关键点，在高斯金字塔的基础上生成高斯差分金字塔（Difference-of-Gaussian, DoG)。高斯差分金字塔也存在组和层的概念，其中每一组对应高斯金字塔的一组，每一层是由它该组上一层与下一层的差得到，计算公式如式 2.3 所示，构造过程如图 2.2 所示。

$$D(x,y,\sigma) = \big(G(x,y,k\sigma) - G(x,y,\sigma)\big) * I(x,y) = L(x,y,k\sigma) - L(x,y,\sigma) \tag{2.3}$$

- **极值点初步检测**：如图 2.3 所示，将高斯差分图像中的每个像素与它的邻域、它对应上一层图像的邻域、对应下一层图像的邻域一共 26 个像素点比较，若其为最值，则记录此点的位置和它所在的尺度，作为候选特征点也就是关键点。

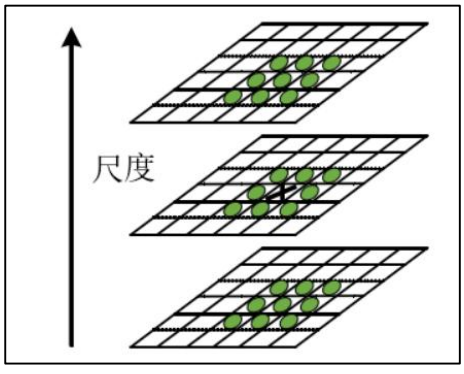

图 2. 3 DOG 尺度空间局部极值检测

2.1.2 关键点定位

上述方法检测到的是离散空间的极值点，并不是真正的连续空间特征点。所以接下来需要精确定位极值点并去除边缘响应点，来增强匹配的稳定性、提高抗噪声能力。通过 Taylor 展开式删除对比度较低的不稳定极值点和插值得到特征点的精确位置。使用 2 阶 Hessian 矩阵计算关键点在 DOG 金字塔中的主曲率以去除边缘点。

2.1.3 方向确定

在过滤掉对比度低的极值点和稳定性差的极值点后，对每一个保留下的极值点进行方向确定，保证任意方向都具有较强的稳定性。对于每一个关键点，通过其四邻域可以得到对应位置的梯度幅度 *m(x, y)*和梯度方向 $\theta(x, y)$：

$$m(x,y) = \sqrt{(L(x+1,y) - L(x-1,y)^2) + (L(x,y+1) - L(x,y-1)^2)} \quad (2.4)$$

$$\theta(x,y) = tan^{-1}\left(\frac{L(x,y+1) - L(x,y-1)}{L(x+1,y) - L(x-1,y)}\right) \quad (2.5)$$

式中，尺度 L 为每个关键点各自所在的尺度。

采样点的梯度经高斯加权后统计在直方图中，根据幅值的大小定义关键点所属的主方向和辅方向。（次峰值约为峰值数值的 80%）。这样，图像的关键点已经完成了检测，每个点包含三个信息：位置、尺度和方向。

2.1.4 关键点描述

在特征对应的高斯图像上，对关键点邻域图像进行区域分块，计算邻域内各点的梯度方向和幅值。由计算的方向信息进行坐标系旋转。计算子区域梯度方向直方图，将一周均分为 8 份，得到如图 2-3 所示的描述符构建过程。由高斯权重对梯度进行加权后，将 4×4 区域的 8 维梯度信息依次合并成 1 个描述符向量。最后进行归一化处理，即可获得尺度不变特征描述符。

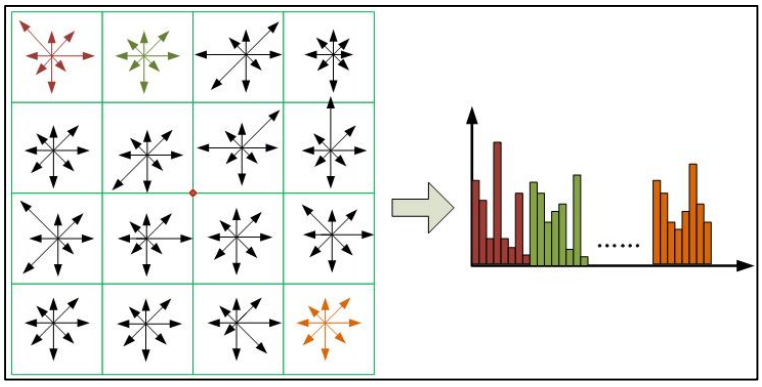

图 2. 4 SIFT 描述符构建过程

2.2 图像特征点匹配

使用 SIFT 算法分别在两张图像上提取特征后，需要根据特征点对向量对两幅图像的相同区域进行匹配。在进行特征点匹配时，首先应对相似性进行度量。匹配过程中，使用欧氏距离最小距离与次小距离的比值与阈值进行比较。如果该比值小于阈值，那么说明匹配正确，若比值大于阈值说明匹配失败，丢弃这对点。假设参考图像中某个特征点的特征向量设为 $T_i=(t_{i1},t_{i2},...,t_{i128})$，图像特征点的描述符向量表示为 $S_i=(s_{i1},s_{i2},...,s_{i128})$，欧氏距离的计算公式如 2.6 所示：

$$d(T_i,S_i)=\sqrt{\sum_{k=1}^{128}(t_{ik}-s_{ik})^2} \tag{2.6}$$

根据上述公式可以算出最小距离 d_1，和次小距离 d_2，设阈值为 r，当 $d_1/\ d_2 \leqslant r$ 时，(T_i,S_i)是一对匹配点。

在确定了两个匹配点的相似性度量算子后，实验中使用 k-d 树对每个特征点最临近的特征点进行优先查找。k-d 树，即 K-Dimensional Tree，是一种高维索引树型数据结构。常用于大规模高维数据空间的邻近或者 K 邻近查找。

2.2 单应性变换

3.2.1 定义

单应性变换（homography transformation）是指在平面上，一个点的位置经过某个变换后，仍然在同一平面上，且不改变它与其他点的位置关系。可以将一个平面或者空间中的点映射到另一个平面或者空间中的点，同时保持一定的几何特性不变。它是一个线性变换，可以用一个 3×3 的矩阵来表示。在图像处理和计算机视觉领域中，单应性变换通常用于将一幅图像中的物体映射到另一幅图像中。例如，当两幅图像视角不同但拍摄的是同一个场景时，可以使用单应性变换将其中一幅图像投影到另一幅图像上，从而实现图像对齐。

单应性变换保持直线的直线性质和平行线的平行性质，但不保持长度和角度等几何量的不变性。因此，单应性变换可以用于图像的仿射变换，如旋转、缩放、平移、错切等，但不能用于非刚性变换，如弯曲或扭曲等。

在计算机视觉中，单应性变换的计算通常使用四个或更多个点的匹配对来估计。这些点的匹配对可以通过手动标记或通过特征匹配算法自动获取。通过对这些点的坐标进行计算，可以获得一个 3×3 的矩阵，该矩阵即为单应性变换的矩阵表示。

3.2.2 求解

定义在齐次坐标系下的一组匹配点为$(x_i, y_i) \overset{match}{\Longleftrightarrow} (x_i', y_i')$，$H_{3\times3}$为图像之间的单应性矩阵，则有等式 2.7 成立：

$$\begin{pmatrix} x_i' \\ y_i' \\ 1 \end{pmatrix} = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix} \begin{pmatrix} x_i \\ y_i \\ 1 \end{pmatrix} = \begin{pmatrix} h_{11}x_i + h_{12}y_i + h_{13} \\ h_{21}x_i + h_{22}y_i + h_{23} \\ h_{31}x_i + h_{32}y_i + h_{33} \end{pmatrix} \tag{2.7}$$

可得方程组：

$$\begin{cases} x_i' = h_{11}x_i + h_{12}y_i + h_{13} \\ y_i' = h_{21}x_i + h_{22}y_i + h_{23} \\ 1 = h_{31}x_i + h_{32}y_i + h_{33} \end{cases} \tag{2.8}$$

进一步变换为：

$$\begin{cases} x_i' = \dfrac{h_{11}x_i + h_{12}y_i + h_{13}}{h_{31}x_i + h_{32}y_i + h_{33}} \\ y_i' = \dfrac{h_{21}x_i + h_{22}y_i + h_{23}}{h_{31}x_i + h_{32}y_i + h_{33}} \end{cases} \tag{2.9}$$

$$\begin{cases} (h_{31}x_i + h_{32}y_i + h_{33}) * x_i' = h_{11}x_i + h_{12}y_i + h_{13} \\ (h_{31}x_i + h_{32}y_i + h_{33}) * y_i' = h_{21}x_i + h_{22}y_i + h_{23} \end{cases} \tag{2.10}$$

写成$AX = 0$的形式：

$$\begin{bmatrix} x_i & y_i & 1 & 0 & 0 & 0 & -x_i'x_i & -x_i'y_i & -x_i' \\ 0 & 0 & 0 & x_i & y_i & 1 & -y_i'x_i & -y_i'y_i & -y_i' \end{bmatrix} \begin{bmatrix} h_{11} \\ h_{12} \\ h_{13} \\ h_{21} \\ h_{22} \\ h_{23} \\ h_{31} \\ h_{32} \\ h_{33} \end{bmatrix} = 0 \tag{2.11}$$

即一组匹配点$(x_i, y_i) \overset{match}{\Longleftrightarrow} (x_i', y_i')$可以获得 2 组方程。观察上述式子，可得在齐次坐标系下，矩阵H和$aH(a \neq 0)$得出的方程一致，作用完全一样。即点(x_i, y_i)无论经过H还是aH映射后，都变化为(x_i', y_i')。所以单应性矩阵$H_{3\times3}$虽然有 9 个参数，但只有 8 个自由度。那么最少只需要 4 组不共线的匹配点即可求解$H_{3\times3}$。下面给出求解的方法与原理：

首先介绍奇异值分解(SVD)：设$A \in \mathbb{C}_r^{m\times n}$，则存在正交矩阵$U \in \mathbb{C}^{m\times m}$与正交矩阵$V \in \mathbb{C}^{n\times n}$，使得

$$A = U \begin{bmatrix} \Sigma_r & 0 \\ 0 & 0 \end{bmatrix} V^T = UDV^T \tag{2.12}$$

其中$\Sigma_r = diag(\sigma_1, \dots, \sigma_r)$是矩阵 A 的奇异值且满足$\sigma_1 > \sigma_2 > \cdots > \sigma_r > 0$。

添加约束$\|H\| = 1$，求解齐次线性方程组$(AH = 0)$，问题等价于$\min_H \|AH\|^2$，

则：

$$\min_{H}\|AH\|^2 = \min_{H}\|UDV^TH\|^2 = \min_{H}\|DV^TH\|^2 \tag{2.13}$$

令$y = V^TH$，则上式变为$\min_{y}\|Dy\|^2 = \min_{y}(\sigma_1^2y_1^2 + \sigma_2^2y_2^2 + \cdots + \sigma_r^2y_r^2)$，因为有$\sigma_1 > \sigma_2 > \cdots > \sigma_r > 0$和$\|H\| = 1$作为约束条件，所以解得$y = [0,0,\ldots,1]^T$，此时$H = Vy$，即为$V$的最后一列。

根据上述分析所得，后续在代码实现中，直接对匹配矩阵进行奇异值分解，取V的最后一列即可获得单应性矩阵H的值。

在经过 SIFT 特征点提取并使用 K-D Tree 对特征点集进行第一次匹配处理后，由于光照、设备、角度的影响会使图像上像素点的灰度和几何结构产生极大的相似匹配，造成了误匹配现象。所以，在初步得到匹配点后，使用 RANSAC 算法剔除误匹配点，从而估计出最佳的单应性矩阵参数。

RANSAC (Random Sample Consensus)算法，即随机抽样一致算法，可用于基于特征点匹配的数据提纯。RANSAC 算法使用迭代的方式从一群离散型数据群中计算出最能描述该数据群的数学模型及参数。在样本数据群中包括正确数据，即那些可以被该同一参数模型描述的数据，也同样包括一些偏离正常范围，即不能被参数模型描述的数据。这些偏离正常范围的数据点可能是由于试验中错误的测量、计算方式造成，需剔除此类噪声点。使用 RANSAC 算法求解最佳单应性矩阵的过程如下：

1. 随机选择 4 个匹配点对，求解出单应性矩阵H；
2. 根据求解出的单应性矩阵H计算在该单应性变换下点集X的映射结果$\tilde{Y}$。
3. 计算Y和$\tilde{Y}$对应点之间的欧式距离，若距离小于预设的阈值，则将该点视为一个内点(inlier)，依次循环计算在该H模型下内点的数量。
4. 重复步骤 1-3，直至找到一个包含最多内点或内点个数满足一定条件的单应性矩阵H，输出此最佳单应性矩阵。

三．实验流程

实验流程可分为加载图像、SIFT 特征点的提取、特征点匹配和单应性矩阵计算四步。下面逐步给出每步的实现关键代码和解释以及结果。使用语言：python。

3.1 加载图像

如图 3.1 所示，使用 opencv2 库读取两张图像，后续计算这两者之间的单应性变换矩阵。图像是工位的不同角度成像。

```
path1 = 'images/img1.jpg'
path2 = 'images/img2.jpg'
img1 = cv2.imread(path1)
img2 = cv2.imread(path2)
plt.subplot(121), plt.imshow(cv2.cvtColor(img1, cv2.COLOR_BGR2RGB)), plt.title('src'), plt.axis('off')
plt.subplot(122), plt.imshow(cv2.cvtColor(img2, cv2.COLOR_BGR2RGB)), plt.title('dst'), plt.axis('off')
plt.show()
```

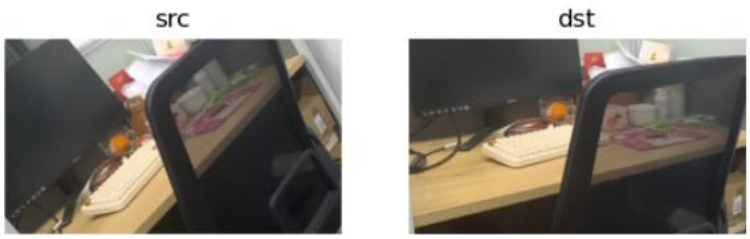

图 3.1 加载图像

3.2 SIFT 特征点检测

如图 3.2 代码所示，使用 opencv 库中自带的 SIFT 特征检测器进行图像的特征点检测，最后得到 128 维的特征点向量。将检测到的 SIFT 特征点在图像中用蓝色圈标出。这些点描述了图像的局部特征，其对旋转、尺度缩放、亮度变化保持不变性，对视角变化、仿射变换、噪声也保持一定程度的稳定性。

```
# 检测sift特征点
sift = cv2.SIFT_create() # 使用cv2库中自带的sift特征检测器
kp1, des1 = sift.detectAndCompute(img1, None)
kp2, des2 = sift.detectAndCompute(img2, None)
print(des1.shape, des2.shape) # 128的特征向量
```

```
(437, 128) (286, 128)
```

```
img_1 = cv2.drawKeypoints(img1, kp1, None, color= [255,0,0])
img_2 = cv2.drawKeypoints(img2, kp2, None, color= [255,0,0])
plt.subplot(121), plt.imshow(cv2.cvtColor(img_1, cv2.COLOR_BGR2RGB)), plt.title('src'), plt.axis('off')
plt.subplot(122), plt.imshow(cv2.cvtColor(img_2, cv2.COLOR_BGR2RGB)), plt.title('dst'), plt.axis('off')
plt.show()
```

图 3.2 SIFT 特征点检测

3.3 SIFT 特征点匹配

如图 5.6 代码所示，使用 KDTree 进行 SIFT 特征向量的欧式距离最近次近点对查询，对特征点进行筛选，保留最近邻距离小于 0.75 倍次近邻距离的点。用蓝色线条将匹配点对连出。

```
from scipy.spatial import KDTree
# 使用KDTree进行最近邻查询
tree = KDTree(des2)
dist, indices = tree.query(des1, k=2)
# 对特征点进行筛选，保留最近邻距离小于0.75倍次近邻距离的点
good_indices = np.where(dist[:, 0] < 0.75 * dist[:, 1])[0]
matches = []
for idx in good_indices:
    matches.append(cv2.DMatch(idx, indices[idx][0], dist[idx][0]))

# 绘制匹配结果
img_match = cv2.drawMatches(img1, kp1, img2, kp2, matches, None, matchColor= [255,0,0])
plt.axis('off')
plt.imshow(cv2.cvtColor(img_match, cv2.COLOR_BGR2RGB))
```

```
<matplotlib.image.AxesImage at 0x1efa810bb20>
```

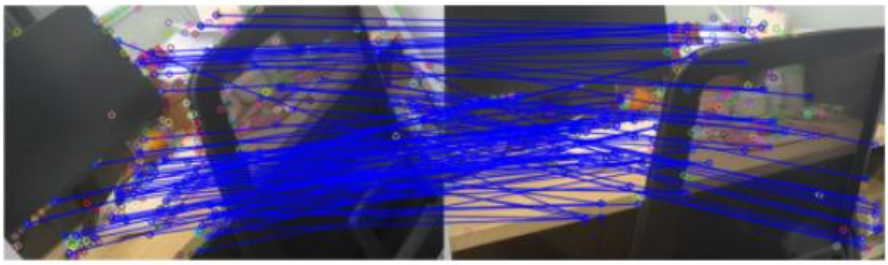

图 3.3 SIFT 特征点匹配

3.4 单应性矩阵计算

根据 2.2.2 小节介绍的单应性矩阵求解方法进行单应性矩阵计算。如图 3.4 所示，使用 4 对匹配对按式 2.11 构造方程组，转换成矩阵后对其使用奇异值分解，根据结论H为V的最后一列，将其 reshape 为 3×3。H和aH作用一致，所以对H_{33}归一化，返回计算的单应性矩阵。

```
# Computers a homography from 4-correspondences
def calculateHomography(correspondences):
    #loop through correspondences and create assemble matrix
    aList = []
    for corr in correspondences:
        p1 = np.matrix([corr.item(0), corr.item(1), 1])
        p2 = np.matrix([corr.item(2), corr.item(3), 1])

        a2 = [0, 0, 0, -p2.item(2) * p1.item(0), -p2.item(2) * p1.item(1), -p2.item(2) * p1.item(2),
              p2.item(1) * p1.item(0), p2.item(1) * p1.item(1), p2.item(1) * p1.item(2)]
        a1 = [-p2.item(2) * p1.item(0), -p2.item(2) * p1.item(1), -p2.item(2) * p1.item(2), 0, 0, 0,
              p2.item(0) * p1.item(0), p2.item(0) * p1.item(1), p2.item(0) * p1.item(2)]
        aList.append(a1)
        aList.append(a2)
    matrixA = np.matrix(aList)
    #svd composition
    u, s, v = np.linalg.svd(matrixA)
    #reshape the min singular value into a 3 by 3 matrix
    h = np.reshape(v[8], (3, 3))
    #normalize and now we have h
    h = (1/h.item(8)) * h
    return h
```

图 3.4 计算单应性矩阵

在经过 SIFT 特征点提取并使用 K-D Tree 对特征点集进行第一次匹配处理后，往往存在一部分坐标匹配是有误差的。此时，使用 2.2.2 小节介绍的 RANSAC 算法排除噪点数据，从而估计出最佳的单应性矩阵参数。1.首先随机选取 4 个匹配点对计算出单应性矩阵。2.根据此单应性矩阵计算点集X的映射结果$\tilde{Y}$。3.计算Y和$\tilde{Y}$对应点之间的欧式距离，若距离小于预设的阈值，则将该点视为一个内点(inlier)，依次循环计算在该H模型下内点的数量。4.重复步骤 1-3，直至找到一个包含最多内点或内点个数总匹配点数 60%的单应性矩阵H，输出此最佳单应性矩阵。

```python
#Runs through ransac algorithm, creating homographies from random correspondences
def ransac(corr, thresh):
    maxInliers = []
    finalH = None
    for i in range(1000):
        #find 4 random points to calculate a homography
        corr1 = corr[random.randrange(0, len(corr))]
        corr2 = corr[random.randrange(0, len(corr))]
        randomFour = np.vstack((corr1, corr2))
        corr3 = corr[random.randrange(0, len(corr))]
        randomFour = np.vstack((randomFour, corr3))
        corr4 = corr[random.randrange(0, len(corr))]
        randomFour = np.vstack((randomFour, corr4))

        #call the homography function on those points
        h = calculateHomography(randomFour)
        inliers = []
        for i in range(len(corr)):
            d = geometricDistance(corr[i], h)
            if d < 5:
                inliers.append(corr[i])
        # update H
        if len(inliers) > len(maxInliers):
            maxInliers = inliers
            finalH = h
        # meet the condition
        if len(maxInliers) > (len(corr)*thresh):
            break
    return finalH, maxInliers
```

图 3. 5 RANSAC 算法计算最佳单应性矩阵

计算Y和$\tilde{Y}$对应点之间的欧式距离的实现代码如图 3.6 所示。满足齐次坐标$w=1$即$(x_i, y_i, w_i) \Rightarrow (x_i/w_i, y_i/w_i, 1)$。

```python
#Calculate the geometric distance between estimated points and original points
def geometricDistance(correspondence, h):

    p1 = np.transpose(np.matrix([correspondence[0].item(0), correspondence[0].item(1), 1]))
    estimatep2 = np.dot(h, p1)
    estimatep2 = (1/estimatep2.item(2))*estimatep2

    p2 = np.transpose(np.matrix([correspondence[0].item(2), correspondence[0].item(3), 1]))
    error = p2 - estimatep2
    return np.linalg.norm(error)
```

图 3. 6 计算距离

实验的总体流程图如图 3.7 所示。

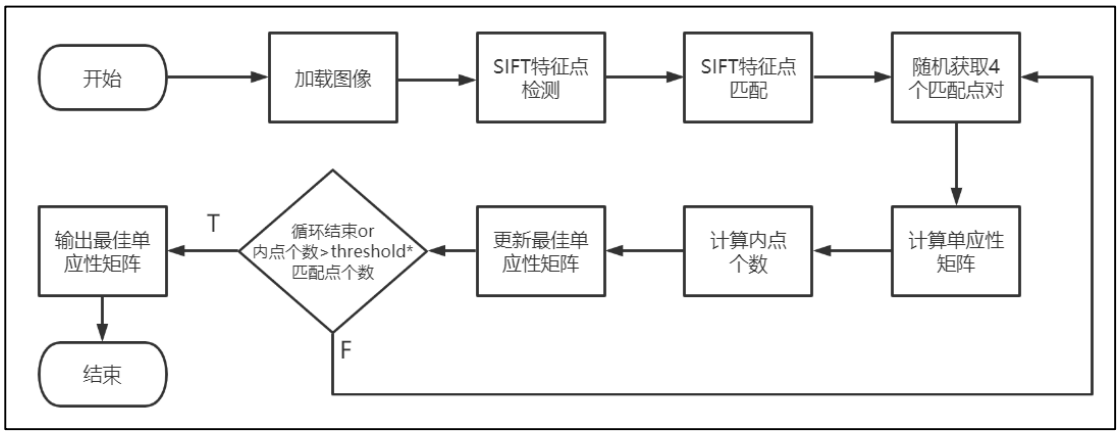

图 3.7 实验总体流程图

四．实验结果

计算出的两张图像之间最佳的单应性矩阵如图 4.1 所示。匹配点对有 87 对，满足内点条件的有 70 对。

```
matches count:  87
Final homography:
 [[ 9.35804588e-01  7.94581731e-01 -9.41677255e+01]
 [-5.56626327e-01  9.36912197e-01  1.72421795e+02]
 [ 1.72726919e-04  4.72348510e-04  1.00000000e+00]]
Final inliers count:  70
```

图 4.1 最佳单应性矩阵结果

如图4.2和图4.3所示，分别计算目标图像映射到原图像的最佳单应性矩阵，并使用矩阵对目标图像进行到原图的坐标映射，结果如第三列所示。

图 4.2 映射结果 1

图 4.3 映射结果 2

五．实验分析与总结

本次实验使用代码实现了图片之间的单应性变换矩阵的计算，完成了实验内容。通过实验了解和掌握了图像单应性的概念、原理和实现方法，了解了单应性矩阵的应用场景。实验结果验证了图像的单应性，理解并掌握了图片单应性矩阵的计算方法及原理。

在实验中，单应性变换矩阵可以通过图像的四个标志点计算得到，通过该矩阵可以将一张图片映射到另一张图片上，显示了变换前后的两张图片。通过对比显示可以发现，单应性变换矩阵的计算准确度较高，能够实现对图像的较好变换。同时，也通过实验调整所选取的标志点的位置，进一步了解单应性变换的特点和变换结果的变化情况。

综上，图像单应性实验通过实际操作，让我更深入地了解了单应性的概念、单应性变换矩阵的计算方法和使用 OpenCV 库进行图像处理的方法，是一次非常有价值的实验。