# 单应性变换

## 实验过程：

数据准备

本次实验使用的是两张图片：一张为源图片（img1.jpg），另一张为目标图片（img2.jpg）。这两张图片分别用于计算单应性矩阵和进行单应性变换。

特征点匹配

使用SIFT算法对源图片和目标图片进行特征点检测，并使用FLANN匹配器进行匹配。将第一幅图像的SIFT特征描述子des1和第二幅图像的SIFT特征描述子des2进行匹配。这里使用的是k-近邻匹配算法，即对于每个第一幅图像中的特征点，找到其在第二幅图像中最近的两个特征点，然后比较这两个特征点距离的大小，如果最近的特征点距离比次近的特征点距离小很多，则认为匹配成功，否则不匹配。这里的k值为2，表示对于每个特征点，最多只匹配2个特征点。匹配结果存储在matches变量中，是一个包含匹配点对的列表。

|  |
| --- |
| import cv2  import numpy as np  from matplotlib import pyplot as plt  # 读入待变换图片和参考图片  img1 = cv2.imread('img1.jpg')  img2 = cv2.imread('img2.jpg')  # 创建SIFT特征提取器  sift = cv2.SIFT\_create()  # 检测关键点并计算描述符  kp1, des1 = sift.detectAndCompute(img1, None)  kp2, des2 = sift.detectAndCompute(img2, None)  # 创建FLANN匹配器  FLANN\_INDEX\_KDTREE = 0  index\_params = dict(algorithm=FLANN\_INDEX\_KDTREE, trees=5)  search\_params = dict(checks=50)  flann = cv2.FlannBasedMatcher(index\_params, search\_params)  # 特征点匹配  matches = flann.knnMatch(des1, des2, k=2) |

筛选匹配点

使用比值测试法筛选匹配点，去除误匹配的点。

|  |
| --- |
| # 过滤匹配点，取优秀匹配  good = []  for m, n in matches:      if m.distance < 0.7 \* n.distance:          good.append(m)  # 绘制匹配结果  img\_match = cv2.drawMatches(img1, kp1, img2, kp2, good, None, flags=2)  # 显示匹配结果  plt.imshow( img\_match),plt.show() |

获取匹配点的坐标

将筛选后的匹配点的坐标分别保存在两个数组中。

|  |
| --- |
| # 获取匹配点坐标  src\_pts = np.float32([kp1[m.queryIdx].pt for m in good]).reshape(-1, 1, 2)  dst\_pts = np.float32([kp2[m.trainIdx].pt for m in good]).reshape(-1, 1, 2) |

计算单应性矩阵

我们使用了cv2.findHomography()函数来计算单应性矩阵。这个函数需要输入两组匹配点的坐标，并指定计算单应性矩阵所使用的算法和阈值。其中，我们使用了RANSAC算法，并将阈值设置为5.0，以进行误差筛选和单应性矩阵的计算。计算完成后，我们将单应性矩阵保存在变量H中，以便后续使用。

|  |
| --- |
| # 计算单应性矩阵  H, \_ = cv2.findHomography(src\_pts, dst\_pts, cv2.RANSAC, 5.0) |

进行单应性变换

使用cv2.warpPerspective()函数进行单应性变换，并将变换后结果进行展示和保存。

|  |
| --- |
| # 对待变换图片进行单应性变换  img\_transformed = cv2.warpPerspective(img1, H, (img2.shape[1], img2.shape[0]))  # 显示变换后的图片  plt.imshow( img\_transformed),plt.show()  # 保存变换后的图片  cv2.imwrite('transformed.jpg', img\_transformed) |

## 结果分析



1. （b）

如图所示，图a为原图像，图b为目标图像。在执行程序后，得到变换后图像如下：



在本次实验中，我们选择了两张图片进行单应性变换。首先，我们对两张图片进行SIFT特征点检测和匹配，筛选出能够用于单应性变换的特征点，最后使用cv2.findHomography()函数计算出单应性矩阵H，利用cv2.warpPerspective()函数将第一张图片进行单应性变换，使其与第二张图片的重合度最高。

实验结果显示，经过单应性变换后的图片与原始图片的重合度较高，表明单应性变换算法具有很好的效果。同时，在特征点匹配的过程中，由于SIFT算法对旋转、尺度和视角的不变性，因此可以有效地匹配出两张图片之间的相似特征点，从而提高单应性变换的准确性。