# 计算机视觉实践——练习 4_单应性变换

## 目录

# 1 实验目的

- 计算图片之间的单应性变换矩阵。

# 2 实验原理

主要包括特征点匹配、计算单应性矩阵 2 个部分。

## 2.1 特征点匹配

### 2.1.1 原理

特征点是图像中的显著、易于识别的位置，比如角点、边缘点等。在图像中提取特征点，是一种常见的图像处理方法，也是计算机视觉领域中的重要内容之一。本次实验使用的特征点提取算法是 SIFT (Scale-Invariant Feature Transform) 算法，该算法可以检测图像中的关键点，并描述它们的特征，通过特征点匹配可以找到两幅图像中对应的特征点。

### 2.1.2 执行步骤

1. 使用 SIFT 算法提取两幅图像的特征点。
2. 使用 Brute-Force 匹配器，对两幅图像中的特征点进行匹配。

## 2.2 计算单应性矩阵

### 2.2.1 原理

单应性矩阵是指对于一对图像中对应的点集，存在一个 3×3 的矩阵 H，使得两个图像中的对应点在同一平面上，其中一个图像上的点通过 H 的变换可以映射到另一个图像上的对应点，如图 1 所示。如果确定了两幅图片的四个匹配的对应点便可求出变换的单应性矩阵 H。

$$\begin{bmatrix} x_2 \\ y_2 \\ z_2 \end{bmatrix} = \begin{bmatrix} H_{11} & H_{12} & H_{13} \\ H_{21} & H_{22} & H_{23} \\ H_{31} & H_{32} & H_{33} \end{bmatrix} \begin{bmatrix} x_1 \\ y_1 \\ z_1 \end{bmatrix} \Leftrightarrow \mathbf{x}_2 = H\mathbf{x}_1$$

图 1 单应性矩阵计算公式

转换坐标$x'_2 = x_2/z_2$，$y'_2 = y_2/z_2$，则：

$$x'_2 = \frac{H_{11}x_1 + H_{12}y_1 + H_{13}z_1}{H_{31}x_1 + H_{32}y_1 + H_{33}z_1}$$

$$y'_2 = \frac{H_{21}x_1 + H_{22}y_1 + H_{23}z_1}{H_{31}x_1 + H_{32}y_1 + H_{33}z_1}$$

令$z_1 = 1$，则：

$$x_2'(H_{31}x_1 + H_{32}y_1 + H_{33}) = H_{11}x_1 + H_{12}y_1 + H_{13}$$
$$y_2'(H_{31}x_1 + H_{32}y_1 + H_{33}) = H_{21}x_1 + H_{22}y_1 + H_{23}$$

令：

$$\begin{aligned}\mathbf{h} &= (H_{11}, H_{12}, H_{13}, H_{21}, H_{22}, H_{23}, H_{31}, H_{32}, H_{33})^T \\ \mathbf{a}_x &= (-x_1, -y_1, -1, 0, 0, 0, x_2'x_1, x_2'y_1, x_2')^T \\ \mathbf{a}_y &= (0, 0, 0, -x_1, -y_1, -1, y_2'x_1, y_2'y_1, y_2')^T .\end{aligned}$$

则：

$$\mathbf{a}_x^T\mathbf{h} = \mathbf{0}$$
$$\mathbf{a}_y^T\mathbf{h} = \mathbf{0}$$

则：

$$A\mathbf{h} = \mathbf{0}$$

其中：

$$A = \begin{pmatrix} \mathbf{a}_{x1}^T \\ \mathbf{a}_{y1}^T \\ \vdots \\ \mathbf{a}_{xN}^T \\ \mathbf{a}_{yN}^T \end{pmatrix}$$

由于样本数大于特征数，为超定问题，故使用最小二乘法来拟合解。原方程最优解就是 V 的最小奇异值对应的列向量，将其 reshape 为 3×3，即求到单应性矩阵 H。

SVD 的求解实际是使用 scipy 库实现的。

# 3 实验

## 3.1 实验细节

本次实验使用的语言为 Python，使用的库有 opencv、numpy、matplotlib、scipy。

## 3.2 实验结果

输入图如下：

图 2 输入图

四点匹配图如下：

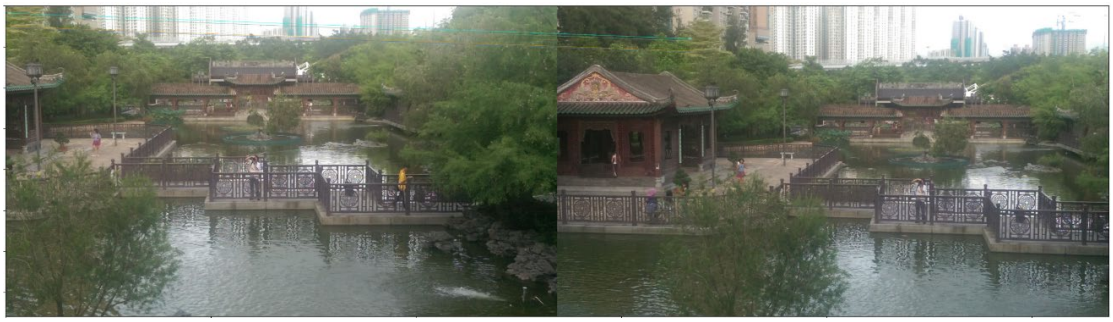

图 3 四点匹配图

单应性矩阵：

```
[[ 3.57565016e-01 -3.21903117e-02  6.55636132e-01]
 [ 1.19316556e-01 -1.73840160e-03 -6.53460681e-01]
 [ 5.38959168e-04 -4.96582106e-05  1.28970947e-03]]
```

## 3.3 实验分析

本次实验的单应性变换的算法基本都是导入已经实现的库的，主要以学习单应性变换是如何计算的。

# 附录——代码展示

仅展示主要代码。

```python
def homography(xy_src, xy_dst):
    src = np.asarray(xy_src, dtype=np.float32)
    dst = np.asarray(xy_dst, dtype=np.float32)

    if src.shape != dst.shape:
        raise Exception('Source and Destination dimensions must be same')
    if src.shape[0] < 4:
        raise Exception('At least 4 set of points is required to compute homography')
    if src.shape[1] != 2:
        raise Exception('Each row in Source and Destination matrices should contain [x, y] points')

    n_points = src.shape[0]

    # Form matrix A
    A = np.zeros((n_points*2, 9), dtype=np.float32)
    for i in range(n_points):
        # A[i] = [-x1, -y1, -1, 0, 0, 0, x2x1, x2y1, x2]
        A[2*i] = [-1.0*src[i][0], -1.0*src[i][1], -1, 0, 0, 0, dst[i][0]*src[i][0], dst[i][0]*src[i][1], dst[i][0]]
        # A[i+1] = [0, 0, 0, -x1, -y1, -1, y2x1, y2y1, y2]
        A[2*i+1] = [0, 0, 0, -1.0*src[i][0], -1.0*src[i][1], -1, dst[i][1]*src[i][0], dst[i][1]*src[i][1], dst[i][1]]

    U, Sigma, V_transpose = linalg.svd(A)

    ## Form homography matrix
    # homography matrix corresponds to the column of V
    # corresponding to the smallest value of Sigma.
    # linalg.svd() returns Sigma in decreasing order
    # hence homography matrix will can be chosesn as
    # the last column of V or last row of V_transpose
    H = np.reshape(V_transpose[-1], (3,3))

    return H
```

```python
def get_kp_match(img1, img2):
        # 使用SIFT获得特征点
        sift = cv2.SIFT_create()
        kp1, des1 = sift.detectAndCompute(img1, None)  # kp: key_points, des: description
        kp2, des2 = sift.detectAndCompute(img2, None)
        raw_matches = cv2.BFMatcher().knnMatch(des1, des2, k=2)  # 使用Brute-Force匹配器，对两幅图像中的特征点进行匹配
        best_features = []
        best_matches = []

        # 通过KNN（K=2）获得最佳特征点
        count = 0  ## 最佳特征点计数
        for m1, m2 in raw_matches:
            if m1.distance < 0.65 * m2.distance:
                best_features.append((m1.trainIdx, m1.queryIdx))
                best_matches.append([m1])
                count += 1
            if count == 4:  ## 获取4个点就可以进行单应性矩阵计算
                break

        img3 = cv2.drawMatchesKnn(img1, kp1, img2, kp2, best_matches, None, flags=2)
        plt.figure(figsize=(30,40))
        plt.imshow(img3[:,:,::-1])

        image1_kp = np.float32([kp1[i].pt for (_, i) in best_features])
        image2_kp = np.float32([kp2[i].pt for (i, _) in best_features])

        return image1_kp, image2_kp
```

```
img1, img2 = load_images(images_folder_path)
kp1, kp2 = get_kp_match(img1, img2)
print(homography(kp1, kp2))
```

在 2s 中执行，2 May at 10:50:45

```
 [[ 3.57565016e-01 -3.21903117e-02  6.55636132e-01]
  [ 1.19316556e-01 -1.73840160e-03 -6.53460681e-01]
  [ 5.38959168e-04 -4.96582106e-05  1.28970947e-03]]
```