单应性变换

姓名：陈浩　学号：123106222839　学院：计算机科学与工程学院

1　实验目标

实现单应性变换，计算图片之间的单应性变换，需要详细的实验过程和结果分析。

2　实现说明

2.1 方法原理

2.1.1 单应性变换

平面的单应性被定义为从一个平面到另一个平面的投影映射，简单说就是原图像（Original Image）上的点可以经过单应性变换到目标图像（Destination Image）上对应位置，其过程用公式表示为：

$$\begin{pmatrix} x_d \\ y_d \\ 1 \end{pmatrix} = H_{3\times3} \times \begin{pmatrix} x_o \\ y_o \\ 1 \end{pmatrix}$$

其中 $(x_d,\ y_d)$ 是目标图像上的点，而 $(x_o,\ y_o)$ 是原图像上的点，这两点称为一对特征匹配点。

进行单应性变换的关键在于计算单应性矩阵 $H_{3\times3}$，一般计算 8 自由度的单应性矩阵。由下面单应性变换可以发现单应性矩阵 H' 和 αH 是等价的，即点 $(x_{ii},\ y_i)$ 无论经过 H' 或 αH 映射，变换后都是 $(x_i',\ y_i')$ 。

$$\begin{pmatrix} x_i' \\ y_i' \\ 1 \end{pmatrix} = H' \times \begin{pmatrix} x_i \\ y_i \\ 1 \end{pmatrix} = \alpha H \times \begin{pmatrix} x_i \\ y_i \\ 1 \end{pmatrix}$$

若令 $\alpha = \frac{1}{h_{33}}$ ，那么有：

$$H' = \alpha H = \begin{bmatrix} \frac{h_{11}}{h_{33}} & \frac{h_{12}}{h_{33}} & \frac{h_{13}}{h_{33}} \\ \frac{h_{21}}{h_{33}} & \frac{h_{22}}{h_{33}} & \frac{h_{23}}{h_{33}} \\ \frac{h_{31}}{h_{33}} & \frac{h_{32}}{h_{33}} & 1 \end{bmatrix} = \begin{bmatrix} h_{11}' & h_{12}' & h_{13}' \\ h_{21}' & h_{22}' & h_{23}' \\ h_{31}' & h_{32}' & 1 \end{bmatrix}$$

通过添加约束，单应性矩阵 H' 虽然有 9 个未知数，但只有 8 个自由度。在求解单应性矩阵时一般添加约束 $h_{33} = 1$ ， 具体的计算方法将上述式子展开：

$$(h_{31}x_i + h_{32}y_i + 1) \cdot x_i' = h_{11}x_i + h_{12}y_i + h_{13}$$
$$(h_{31}x_i + h_{32}y_i + 1) \cdot y_i' = h_{21}x_i + h_{22}y_i + h_{23}$$

写成矩阵 $AX = b$ 形式：

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

由于一组匹配点对应 2 组方程，那么只需 4 组不共线的匹配点即可求解出矩阵。

2.1.2 尺度不变特征转换 SIFT

尺度变化不变特征 SIFT 通过在不同尺度空间中检测关键点，并利用关键点周围的局部图像信息生成描述子，实现了图像特征的尺度不变性和一定程度的旋转不变性。SIFT 算法的核心步骤如下。

①方向赋值，在实现关键点检测和定位后，对于每个关键点 SIFT 进行方向赋值，即确定关键点的主导方向。这一步是为了提高关键点的旋转不变性，使得关键点描述子在旋转变换下具有稳定性。通常采用图像局部梯度信息来确定关键点的主导方向。

②关键点描述，SIFT 利用关键点周围的局部图像区域来生成描述子，通常采用的方法是在关键点周围的图像区域中构建特征向量，该向量能够描述关键点周围的图像结构和纹理特征。这些描述子具有一定的局部不变性，能够在一定程度上抵抗图像的缩放、旋转和亮度变化等影响。

③匹配与验证，最后利用关键点的描述子进行特征匹配，通常采用的方法是计算描述子之间的相似性度量，如欧氏距离或余弦相似度。

2.2 实验流程

(a) 特征点匹配。本次实验实现了两种特征点匹配的方法。

首先是手动特征点匹配，利用鼠标事件标注获取四对匹配点坐标。

第二种方法是基于 SIFT 进行特征点匹配，具体流程如下：

①检测关键点和计算关键点描述，先将两图片转换为灰度图，并建立 SIFT 生成器，然后检测特征点并计算描述子。

②匹配两张图片的特征点，我采用 KNN 检测来自两图的 SIFT 特征匹配，过滤掉不合格匹配对，完成满足条件的匹配对点坐标的提取。

(b) 计算单应性矩阵

在计算单应性矩阵之前，要确保提供了至少四个源点和目标点以及源点和目标点数量相同。随后构建一个线性方程组，待定参数为单应性矩阵，并利用最小二乘法(Least Squares Method)求解这个方程组，得到单应性矩阵的近似解。

(c) 进行单应性变换

利用 cv2.warpPerspective 函数，传入原图像和单应性矩阵，即可实现从原图像到目标图像的单应性变换。

2.3 代码实现

2.3.1 鼠标事件获取特征点坐标

由函数 cv2.setMouseCallback('image', on_mouse)实现，其中 on_mouse 函数实现了点击图像打印该点坐标。由此可以获取原图和目标图匹配的四对特征点。

2.3.2 检测关键点和计算关键点描述

由函数 detectAndDescribe(image)实现，具体流程如下：

①使用 cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)将输入的彩色图像转换为灰度图像，以便进行后续的特征检测和描述子计算。

②使用 cv2.SIFT_create()创建一个 SIFT 特征检测器，方便使用 SIFT 算法。

③使用 descriptor.detectAndCompute(gray, None)对灰度图像进行特征点检测和描述子计算，得到关键点列表 kps 和对应的特征描述子 features。

④将关键点列表中的关键点坐标转换为 numpy 数组格式，以便后续的处理。

2.3.3 基于 SIFT 匹配两张图片的特征点

由函数 matchKeypoints(kpsA, kpsB, featureA, featureB, ratio, reprojThresh)实现，具体流程如下：

①创建暴力匹配器对象 matcher = cv2.BFMatcher()，用于在特征空间中寻找最接近的匹配点。

②使用 K 最近邻（KNN）方法对两图像的 SIFT 特征进行匹配，具体为 matcher.knnMatch(featureA, featureB, 2)。这里的 featureA 和 featureB 分别是两张图像的特征描述子，而 2 表示每个特征点要找到最接近的两个匹配点。

③对原始匹配结果进行筛选，保留满足一定比例关系的匹配对。具体操作为遍历 rawMatches 中的每对匹配点，然后判断是否满足以下条件：

len(m) == 2：确保每个特征点都有两个最接近的匹配点。

m[0].distance < m[1].distance * ratio：第一个最近邻匹配点的距离小于第二个最近邻匹配点的距离乘以 ratio。如果满足这个条件，则认为这对匹配是合格的，应该保留下来。

④如果满足条件的匹配对数量大于 4，则继续进行下一步处理；否则返回空值。对于满足条件的匹配对，提取它们的点坐标。

⑤利用 RANSAC 算法估计单应性矩阵 H，以及状态 status。这里使用 cv2.findHomography(ptsB, ptsA, cv2.RANSAC, reprojThresh) 进行计算。

⑥最后将经过筛选后的匹配对 matches、单应性矩阵 H 和状态 status 返回。

2.3.4 计算单应性矩阵

①首先使用 assert 断言，确保提供了至少四对源点和目标点以及源点和目标点数量相同：

```
assert len(source) >= 4, "must provide more than 4 source points"
assert len(destination) >= 4, "must provide more than 4 destination points"
assert len(source) == len(destination), "source and destination must be of equal length"
```

②构建线性方程组。根据 2.1.1，构建 $AX = b$ 形式方程组：

```
A.append([s_x, s_y, 1, 0, 0, 0, (-d_x)*(s_x), (-d_x)*(s_y)])
A.append([0, 0, 0, s_x, s_y, 1, (-d_y)*(s_x), (-d_y)*(s_y)])
```

b += [d_x, d_y]

上面三行代码针对每一对源点和目标点，构建了两个方程，并将其系数分别添加到矩阵 A 中，常数项添加到向量 b 中。具体来说，对于第 i 个点

$$h_{11}s_x + h_{12}s_y + h_{13} - h_{31}s_x d_x - h_{32}s_y d_x = d_x$$
$$h_{21}s_x + h_{22}s_y + h_{23} - h_{31}s_x d_y - h_{32}s_y d_y = d_y$$

③最小二乘法求解单应性矩阵

然后将 A 转换为数组，并利用 np.linalg.lstsq(A, b)函数对②中线性方程组进行最小二乘拟合，得到方程组的近似解 h。最后，为了将 h 转换为 3×3 的单应性矩阵形式，代码使用 np.concatenate((h, [1]), axis=-1) 函数在 h 的末尾添加一个值 1，形成一个 3x3 的单应矩阵。

2.3.5 单应性变换

cv2.warpPerspective(imageA, H, (w, h))：这一行代码对图像 A 进行单应性变换，H 是单应性矩阵，指定了变换的方式，变换后的图像大小是(w, h)，即这行代码实现了图像 A 以一特定单应性矩阵进行单应性变换。

3 实验结果

3.1 运行说明

核心代码在 homography.ipynb 文件中，运行说明如下：

①顺序执行每个单元框，会详细地展示每一步骤结果。

②每个单元框保留了上一次运行的结果，可以不运行直接查阅。

③输入图像存放于 img/input 文件夹，结果输出在 img/result 文件夹下。其中 output1 为使用手动特征点匹配实现单应性变换的结果，output2 为基于 SIFT 实现单应性变换的结果，match.png 为基于 SIFT 特征点匹配结果。

3.2 结果与说明

两张输入图像如下图所示：

3.2.1 特征点匹配

图中的绿线连接了两张图片的匹配的关键点。可以观察到树上的文字和图形征实现了一一匹配。

3.2.2 单应性矩阵计算

在人工特征点匹配中，我使用鼠标事件获取两张输入图中书本四个角的坐标，具体坐标如下图所示：

```
x: 19,  y: 400    x: 167,  y: 142
x: 397,  y: 90    x: 701,  y: 139
x: 837,  y: 648   x: 701,  y: 912
x: 453,  y: 945   x: 167,  y: 905
```

使用自编写的函数计算单应性矩阵和使用 cv2.findHomography 库函数计算的结果如下：

```
[[ 8.55718469e-01 -6.83513827e-01  4.23080469e+02]
 [ 6.81267888e-01  8.37277705e-01 -2.06761941e+02]
 [ 4.69175117e-06 -1.61870972e-05  1.00000000e+00]]
[[ 8.55718469e-01 -6.83513826e-01  4.23080469e+02]
 [ 6.81267888e-01  8.37277705e-01 -2.06761941e+02]
 [ 4.69175116e-06 -1.61870972e-05  1.00000000e+00]]
```

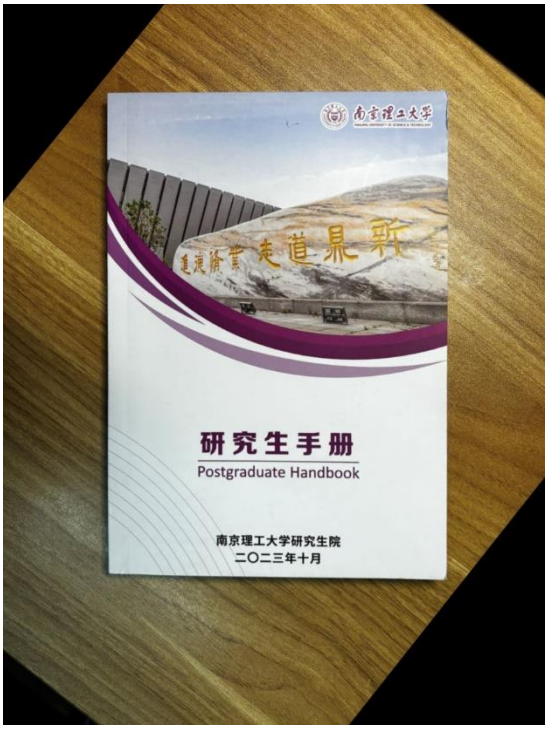
进一步计算两矩阵的差如下图：

```
两矩阵的差：
[[-3.55637741e-12 -1.17703625e-11  4.95305130e-09]
 [ 2.02073913e-11 -1.67619252e-11 -7.67681740e-09]
 [ 9.87479057e-15 -2.82330752e-14  0.00000000e+00]]
```

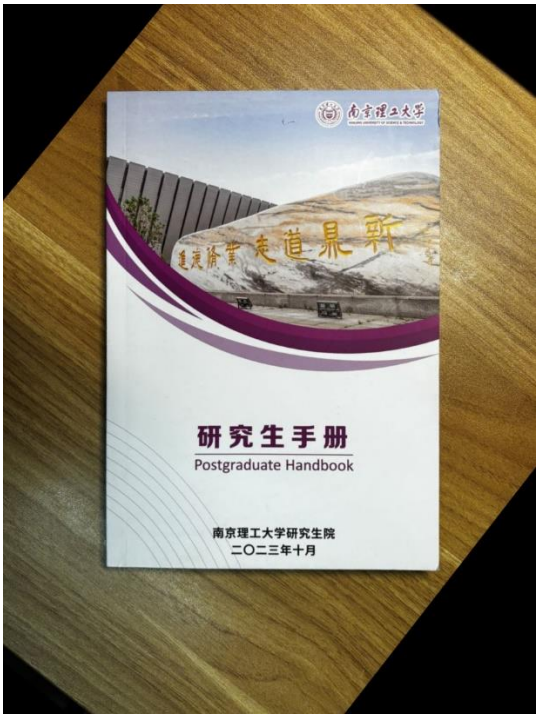
可以看出两中方式计算得到的单应性矩阵相差很小，这也验证了自编写函数计算单应性矩阵的正确性。

3.2.3 单应性变换

使用人工特征点匹配和自编写函数计算得到的单应性矩阵，进行单应性变换结果如下图：



基于 SIFT 实现单应性变换的结果如下图：



从两张图的整体效果分析，使用人工特征点匹配计算得到的单应性矩阵，也可以较好的实现单应性变换。但细致地分析发现两张图有细微的角度区别，这是因为人工标定方式的匹配对较少且有人工误差；而基于 SIFT 的特征点匹配不仅数量更多且更为精准，所以有更好的变换结果。