计算机视觉实践-练习 4

122106222784 贺梦瑶

1、实验目标

实现图片间的单应性变换。

2、单应性介绍

单应性在计算机视觉领域是一个非常重要的概念，它在图像校正、图像拼接、相机位姿估计、视觉 SLAM 等领域有非常重要的作用。单应性变换，可简单理解为用来描述物体在世界坐标系和像素坐标系之间的位置映射关系，对应的变换矩阵称为单应性矩阵，如下所示：

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

考虑第一组对应点 （X1，Y1） 在第一张图像和 （X2，Y2）第二张图像中。然后，Homography H 以下列方式映射它们：

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

坐标变换：

$$x'=\frac{h_{00}x+h_{01}y+h_{02}}{h_{20}x+h_{21}y+h_{22}} \qquad y'=\frac{h_{10}x+h_{11}y+h_{12}}{h_{20}x+h_{21}y+h_{22}}$$

单应矩阵 H 有 8 个自由度，要得到两张图片的 H 矩阵，就必须至少知道 4 个相同对应位置的点，通常，这些点对应是通过匹配图像之间的 SIFT 或 Harris 等特征自动找到的，本次实验使用 SIFT 方法获得匹配点对。

实现单应性转换通常需要以下几个步骤：

（1）获取图像，读入作为参考的图像，以及想要与此模板对齐的图像，所以需要读入两张图像。

（2）寻找特征点，读入图像后，检测两张图像中的 ORB 特征。

（3）对特征点进行匹配，在上一步中我们得到了两张图像中匹配的特征，这时按匹配的评分（类似于相似度）对选取的特征进行排序，并保留其中一小部分的原始匹配。

（4）计算单应性变换 对于计算映射矩阵，我们需要在两张图像中至少得到 4 个对应点来计算单应性变换。其利用随机抽样一致性算法（RANSAC）的精准估计技术，使得在存在大量错误匹配的情况下也能得到相对正确的匹配结果。

（5）图像映射，计算出两张图像中的单应性变换之后，对第一张图像中的所有像素点，通过乘以单应性变换的方式将其映射到对齐以后的图像。

3、实现说明

（1）获得匹配点

```
def matchPics(I1, I2):
    # 使用 SFIT 获得匹配点对

    # Initializing empty lists
    locs1 = []
```

```
    locs2 = []
    matches = []

    # Convert images to 8 bits
    I1_converted = img_as_ubyte(I1)
    I2_converted = img_as_ubyte(I2)

    # Initialize ORB and matcher objects
    orb = cv2.ORB_create()
    matcher = cv2.BFMatcher()

    # Compute descriptors and keypoints
    keypoint1, descriptor1 = orb.detectAndCompute(I1_converted, None)
    keypoint2, descriptor2 = orb.detectAndCompute(I2_converted, None)

    # Matches descriptors of keypoints
    matches = matcher.match(descriptor1, descriptor2)

    # Sort points by distance
    matches = sorted(matches, key=lambda x: x.distance)

    # Store coordinates of keypoints for each match
    locs1 = [keypoint1[m.queryIdx].pt for m in matches]
    locs2 = [keypoint2[m.trainIdx].pt for m in matches]

    # Convert in array
    matches = np.array([[m.queryIdx, m.trainIdx] for m in matches])

    return matches, locs1, locs2
```

(2) RANSAC 随机采样

```
def computeH_ransac(matches, locs1, locs2):
    iterations = 100000
    threshold = 5

    locs1 = np.array(locs1)
    locs2 = np.array(locs2)
```

```
7.
8.      max_inliers = np.zeros(matches.shape[0])
9.      bestH = None
10.     inliers = None
11.
12.     for i in range(iterations):
13.         # Randomly select four matches
14.         indices = np.random.choice(matches.shape[0], 4, replace=Fals
    e)
15.         sample_locs1 = locs1[indices]
16.         sample_locs2 = locs2[indices]
17.
18.         # Compute homography using the four matches
19.         H = computeH(sample_locs1, sample_locs2)
20.
21.         # Apply homography to all matches
22.         warped_locs1 = applyH(H, locs1)
23.         d = np.linalg.norm(warped_locs1 - locs2, axis=1)
24.
25.         # Count inliers
26.         inliers = d < threshold
27.         n_inliers = np.sum(inliers)
28.
29.         # Check if this is the best estimate so far
30.         if n_inliers > np.sum(max_inliers):
31.             max_inliers = inliers
32.             bestH = H
33.             print(bestH)
34.
35.     return bestH, max_inliers.astype(bool)
```

(3) 得到匹配区域

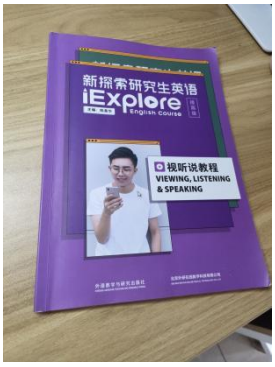
```
1. def visualize_box(template, target, H):
2.     nrow, ncol = template.shape[:2]
3.     row = np.array([0, 0, nrow -1, nrow - 1, 0]).astype(float)
4.     col = np.array([0, ncol-1, ncol-1, 0, 0]).astype(float)
5.     x = H[0,0]*col + H[0,1]*row + H[0,2]
6.     y = H[1,0]*col + H[1,1]*row + H[1,2]
7.     w = H[2,0]*col + H[2,1]*row + H[2,2]
```

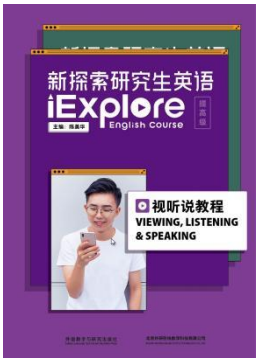
```
    x = x / w; y = y / w
    x = np.round(x).astype(int)
    y = np.round(y).astype(int)
    plt.imshow(target)
    plt.plot(x, y, 'r-')
    plt.savefig(r"./imag/box.jpg")
    plt.show()
```

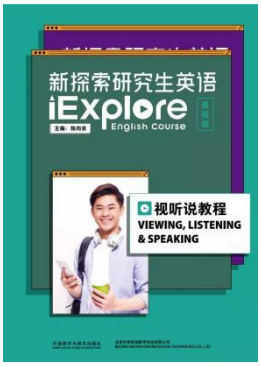
(4) 视图替换

```
def compositeH(H, template, img):
    img_shape = img.shape[1], img.shape[0]
    h, w = template.shape[:2]

    #Create mask of same size as template
    mask = np.ones((h, w))

    #Warp mask by appropriate homography
    warp_mask = cv2.warpPerspective(mask, H, img_shape)

    #Warp template by appropriate homography
    warp_template = cv2.warpPerspective(template, H, img_shape)

    #Use mask to combine the warped template and the image
    composite_img = img.copy()
    composite_img[warp_mask == 1] = warp_template[warp_mask == 1]

    return composite_img
```

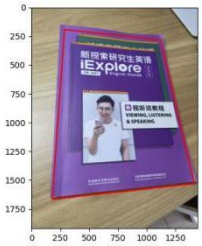
4、结果展示







实验将图一中的书本封面，即图二替换为图三，为了使特征匹配过程中更关注物体间的特征相似度而非颜色，将图一和图二先转换为灰度图进行特征匹配。


以上三幅图分别表示特征匹配，替换区域框选以及实现替换三个步骤的可视化展现，将替换图片经过单应性变换后在原来的位置代替了被替换文件。