计算机视觉第三次作业报告--单应矩阵的估计

人工智能82班 刘志成 2183511589

相关代码已上传至我的github仓库：
https://github.com/zchliu/2020-Fall

实验题目

单应矩阵估计实验，给定两个图像的几对对应点，估计两幅图像变换的单应矩阵

数学模型

单应矩阵如下：

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

由于最终输出的点为一个齐次坐标，因此H的自由度要减一，为8个自由度，这里取$h_{33} = 1$作为约束，因此最终的单应矩阵为：

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

变换关系为：

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

$$x'' = \frac{1}{z'}x' = \frac{h_{11}x + h_{12}y + h_{13}}{h_{31}x + h_{32}y + 1}$$

$$y'' = \frac{1}{z'}y' = \frac{h_{21}x + h_{22}y + h_{23}}{h_{31}x + h_{32}y + 1}$$

因此，每一对点提供两个方程：

$$h_{11}x_B + h_{12}y_B + h_{13} - x_A x_B h_{31} - x_A y_B h_{32} - x_A = 0$$
$$h_{21}x_B + h_{12}y_B + h_{13} - x_A x_B h_{31} - x_A y_B h_{32} - x_A = 0$$

写成矩阵形式：

$$\begin{bmatrix} x_B & y_B & 1 & 0 & 0 & 0 & -x_A x_B & -x_A y_B & -x_A \\ 0 & 0 & 0 & x_B & y_B & 1 & -y_A x_B & -y_A y_B & -y_A \end{bmatrix} \begin{bmatrix} h_{11} \\ h_{12} \\ h_{13} \\ h_{21} \\ h_{22} \\ h_{23} \\ h_{31} \\ h_{32} \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

因此，给定4对点即可算出单应矩阵H

将所有点的的方程放入矩阵M中，求这个矩阵M的svd分解，其对应的最小的奇异值的右向量即为所求的单应矩阵的参数

计算模型

SSD距离

SSD距离计算两个匹配样本点的欧式距离，计算公式如下：

$$d(\vec{x}, \vec{y}) = ||\vec{x} - \vec{y}||^2$$

RANSAC算法

RANSAC算法又称为随机采样一致性算法，算法伪代码如下：

1. 随机选取4对样本点
2. 根据这4个样本点计算单应矩阵H
3. 利用这个单应矩阵H和SSD距离计算所有匹配点的内点数，并记录当前最大的内点数和对应的单应矩阵H，返回1
4. 上述过程迭代100次，得到当前最大内点数和匹配精度最好的单应矩阵H

实验结果

用库函数提取SIFT描述子并求对应点匹配如下：

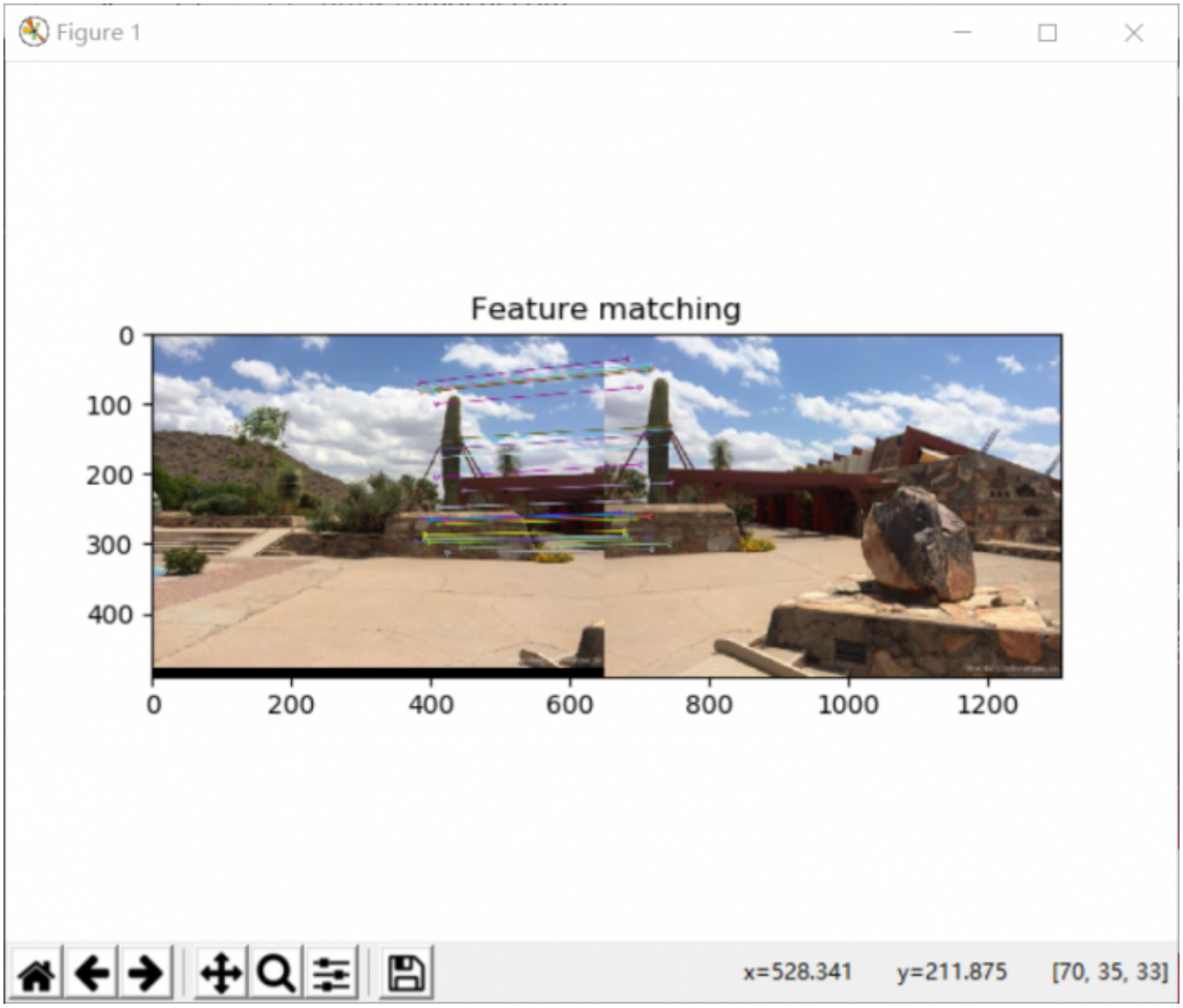
```
img_left = cv2.imread('left.png', 1)
    img_right = cv2.imread('right.png', 1)

    img_left_gray = cv2.cvtColor(img_left, cv2.COLOR_BGR2GRAY)
    img_left_des = cv2.SIFT_create()
    left_kps, left_features = img_left_des.detectAndCompute(img_left_gray, None)

    img_right_gray = cv2.cvtColor(img_right, cv2.COLOR_BGR2GRAY)
    img_right_des = cv2.SIFT_create()
    right_kps, right_features = img_right_des.detectAndCompute(img_right_gray, None)

    matcher = cv2.BFMatcher()
    raw_matches = matcher.knnMatch(left_features, right_features, 2)
```

效果如下



直接估计单应矩阵代码如下：

```
def my_findHomography(right_kps, left_kps):

    length = len(right_kps)

    M = np.zeros((2 * length, 9))
    b = np.zeros((2 * length, 0))
    for i in range(length):
        right_kp = right_kps[i]
        left_kp = left_kps[i]
        M[2 * i, :] = np.array([right_kp[0], right_kp[1], 1, 0, 0, 0, -left_kp[0] * right_kp[0], -left_kp[0] * ri
        M[2 * i + 1, :] = np.array([0, 0, 0, right_kp[0], right_kp[1], 1, -left_kp[1] * right_kp[0], -left_kp[1]
    U, S, V = svd(M)
    h = V.T[:, 8]
    h = h / h[8]
    H = np.array([[h[0], h[1], h[2]], [h[3], h[4], h[5]], [h[6], h[7], h[8]]])
    return H
```

效果如下：

可以看出匹配受到噪声点的影响，右下角石头的拼接不够精确

加入RANSAC算法的代码如下：

```
def count_inliers(right_kps, left_kps, H, threshold):

    length = len(right_kps)
    inliers = 0
    outliers = 0
    for i in range(length):
        distance = my_SSD(right_kps[i], left_kps[i], H)
        if distance < threshold:
            inliers = inliers + 1
        else:
            outliers = outliers + 1
    return inliers, outliers


def my_SSD(right_kp, left_kp, H):

    right_homogeneous = np.array([[right_kp[0]], [right_kp[1]], [1]])
    left_homogeneous = np.array([[left_kp[0]], [left_kp[1]], [1]])
    left_predict = np.dot(H, right_homogeneous)
    left_predict = left_predict / (left_predict[2] + epsilon)
    distance = np.sum((left_predict - left_homogeneous)**2)
    return distance


def my_RANSAC(right_kps, left_kps, threshold):

    length = len(right_kps)

    temp_left_kps = np.zeros((4, 2))
    temp_right_kps = np.zeros((4, 2))

    max_inliers = -1
    good_H = np.zeros((3, 3))

    # 循环迭代100次，找到最佳的H
    for j in range(100):
        for i in range(4):
            rand_num = random.randint(0, length - 1)
            temp_left_kps[i, :] = left_kps[rand_num, :]
            temp_right_kps[i, :] = right_kps[rand_num, :]
        H = my_findHomography(temp_right_kps, temp_left_kps)
        inliers, outliers = count_inliers(right_kps, left_kps, H, threshold)
        if inliers > max_inliers:
            max_inliers = inliers
            good_H = H

    return good_H
```

效果如下：

可以看到，加入RANSAC以后的算法匹配精度比之前要更好