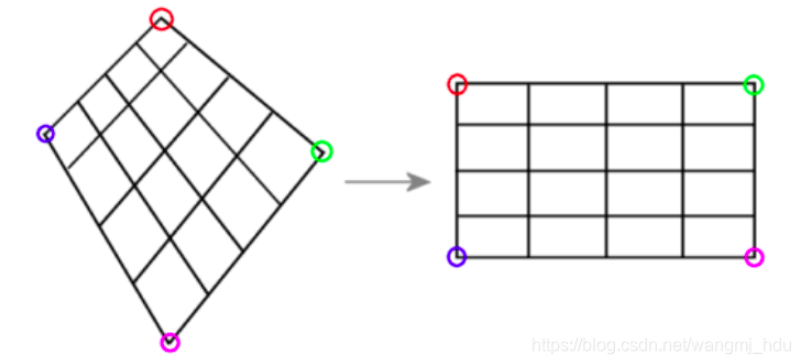
单应性变换实验报告

1. **实验目标**

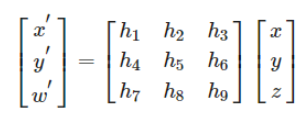
单应性变换,计算图片之间的单应性变换,需要详细的实验过程和结果分析。

1. **实验原理**

单应性（3 \* 3的矩阵）是指从图像中的一个点映射到另一张图像中的对应点的变换。



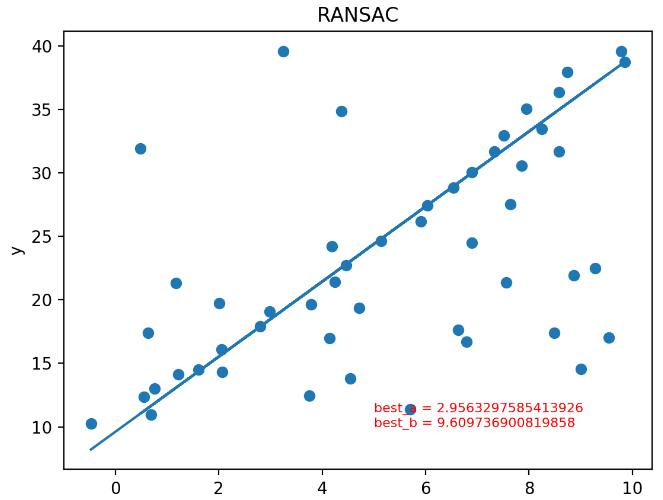
在这里，平面是指图像或者三维中的平面表示。单应性变换具有很强的实用性，比如图像配准，图像纠正和纹理扭曲，以及创建全景图像，我们将频繁的使用单应性变换。本质上，单应性变换H，按照下面的方程映射二维中的点（齐次坐标意义下）：



通常对于齐次坐标上图中的w’ = z = 1。

RANSAC算法的中文名称是随机抽样一致算法(Random Sample Consenus)，简单的说，通过RANSAC算法，我们将数据分为inliers和outliers,inliers是对于模型拟合有效的点，也称之为内点；outliers是对于模型拟合无效的点，也就是错误的数据点，称之为外点。而我们在使用观测数据拟合模型的过程中，外点的存在对于使用数据拟合模型是有害的，那么我们该如何剔除这些外点呢？RANSAC算法就是能够剔除外点的一个迭代性的算法。

RANSAC算法在图像匹配中的作用是：剔除外点，使模型估计更加准确。



算法的实现流程：

1. 选择出估计模型的最小数据样本（对于二维和三维直线拟合来说，确定一条直线最少需要2个点；对于三维平面拟合来说，确定一个三维平面最少需要3个点）
2. 使用这个最小的数据样本，算出拟合的模型。（也就是直线方程或者平面方程）

将所有的模型带入这个拟合的模型，计算出内点 的数量。（数据点和拟合的模型的误差在一定阈值范围内的数据点的数量）

3.比较当前模型和之前迭代的得到的最好的模型的内点数量（内点数量越多，模型越好），记录最大的内点数的模型参数和内点数量。

4.重复1-4步，直到达到迭代终止条件（例如达到最大迭代数、内点数量达到迭代终止条件）

1. **实验步骤**

计算图像之间的单应性变换，其实就是求解两张图片中的H矩阵。可以分为以下几个步骤。

* 1. 提取两张图片的特征点并匹配

在这个步骤采用SIFT特征点检测器进行操作，得到两张图像的特征点和描述子，并保存下来。

def detectAndDescribe(img):

    # 构建 SIFT 特征检测器

    sift = cv2.SIFT\_create()

    # 进行特征提取

    kps, features = sift.detectAndCompute(img, None)

    return kps, features

def drawpos(img\_left, img\_right, kps\_left, kps\_rigt):

    hl, wl = img\_left.shape[:2]

    hr, wr = img\_right.shape[:2]

    vis = np.zeros((max(hl, hr), wl + wr, 3), dtype=np.uint8)

    vis[0:hl, 0:wl] = img\_left

    vis[0:hr, wl:] = img\_right

    for kp in kps\_left:

        pos\_l = (int(kp.pt[0]), int(kp.pt[1]))

        cv2.circle(vis, pos\_l, 3, (0, 0, 255), 1)

    for kp in kps\_rigt:

        pos\_r = (int(kp.pt[0] + wl), int(kp.pt[1]))

        cv2.circle(vis, pos\_r, 3, (0, 255, 0), 1)

    return vis

之后找到两张图片中最近的点和次近的点，得到好的匹配结果。

def matchKeyPoint(kps\_l, kps\_r, features\_l, features\_r, ratio):

    #

    Match\_idxAndDist = []  # 存储最近点位置、最近点距离、次近点位置、次近点距离

    for i in range(len(features\_l)):

        # 从 features\_r 中 找到与 i 距离最近的2个点

        min\_IdxDis = [-1, np.inf]  # 距离最近的点,

        secMin\_IdxDis = [-1, np.inf]  # 距离第二近的点,

        for j in range(len(features\_r)):

            dist = np.linalg.norm(features\_l[i] - features\_r[j])

            if min\_IdxDis[1] > dist:

                secMin\_IdxDis = np.copy(min\_IdxDis)

                min\_IdxDis = [j, dist]

            elif secMin\_IdxDis[1] > dist and secMin\_IdxDis[1] != min\_IdxDis[1]:

                secMin\_IdxDis = [j, dist]

        Match\_idxAndDist.append(

            [min\_IdxDis[0], min\_IdxDis[1], secMin\_IdxDis[0], secMin\_IdxDis[1]]

        )

    goodMatches = []

    # 如果i与最近的2个点的距离差较大，那么就不是好的匹配点

    # 即 |fi-fj|/|fi-fj'|>ratio 则取消匹配点

    for i in range(len(Match\_idxAndDist)):

        if Match\_idxAndDist[i][1] <= Match\_idxAndDist[i][3] \* ratio:

            goodMatches.append((i, Match\_idxAndDist[i][0]))

    # 获取匹配较好的点对

    goodMatches\_pos = []

    for idx, correspondingIdx in goodMatches:

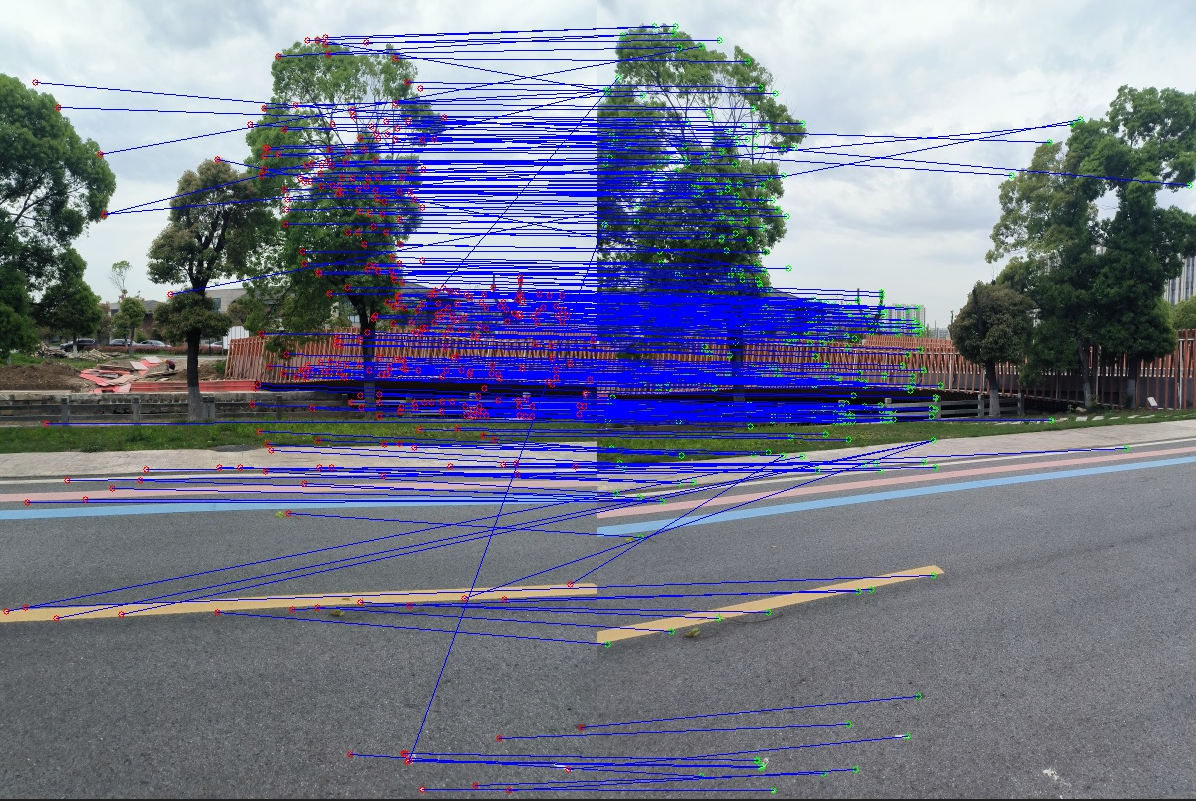
        psA = (int(kps\_l[idx].pt[0]), int(kps\_l[idx].pt[1]))

        psB = (int(kps\_r[correspondingIdx].pt[0]), int(kps\_r[correspondingIdx].pt[1]))

        goodMatches\_pos.append([psA, psB])

    return goodMatches\_pos

最终的匹配结果如下。



3.2通过RANSAC算法求解H矩阵

从上图中可以看出，两张图片中通过特征匹配，有些特征点并没有很好的对应。这会导致计算H矩阵时候出现较大的误差。这时我们可以通过RANSAC算法进行迭代，获取最优的H矩阵。

H矩阵的计算代码如下：

#  求解H矩阵

# P 源图像点坐标 m 目标图像点坐标

def solve\_homography(P, m):

    try:

        A = []

        for r in range(len(P)):

            A.append(

                [

                    -P[r, 0],

                    -P[r, 1],

                    -1,

                    0,

                    0,

                    0,

                    P[r, 0] \* m[r, 0],

                    P[r, 1] \* m[r, 0],

                    m[r, 0],

                ]

            )

            A.append(

                [

                    0,

                    0,

                    0,

                    -P[r, 0],

                    -P[r, 1],

                    -1,

                    P[r, 0] \* m[r, 1],

                    P[r, 1] \* m[r, 1],

                    m[r, 1],

                ]

            )

        u, s, vt = np.linalg.svd(

            A

        )  # Solve s ystem of linear equations Ah = 0 using SVD

        # pick H from last line of vt

        H = np.reshape(vt[8], (3, 3))

        # normalization, let H[2,2] equals to 1

        H = (1 / H.item(8)) \* H

    except:

        print("Error on compute H")

    return H

使用RANSAC的内外点判断，最终优化H矩阵。

# 利用 RANSAC算法，计算H矩阵

def fitHomoMat(matches\_pos, nIter=1000, th=5.0):

    dstPoints = []  # i.e. left image(destination image),

    srcPoints = []  # i.e. right image(source image) ,

    for dstPoint, srcPoint in matches\_pos:

        dstPoints.append(list(dstPoint))

        srcPoints.append(list(srcPoint))

    dstPoints = np.array(dstPoints)

    srcPoints = np.array(srcPoints)

    # 利用RANSAC算法, 获取最优的H矩阵,

    NumSample = len(matches\_pos)

    threshold = th

    NumIter = nIter

    NumRamdomSubSample = 4

    MaxInlier = 0

    Best\_H = None

    for run in range(NumIter):

        # 随机采样

        SubSampleIdx = random.sample(range(NumSample), NumRamdomSubSample)

        # 计算 H

        H = solve\_homography(srcPoints[SubSampleIdx], dstPoints[SubSampleIdx])

        # find the best Homography have the the maximum number of inlier,

        NumInlier = 0

        pos\_Inlier = []

        for i in range(NumSample):

            if i not in SubSampleIdx:

                concateCoor = np.hstack((srcPoints[i], [1]))  # 添加z =1

                dstCoor = H @ concateCoor.T  # 计算目标点

                if dstCoor[2] <= 1e-8:  #  避免目标点 z 维度接近 0

                    continue

                dstCoor = dstCoor / dstCoor[2]  # 计算目标点坐标

                # 如果计算的目标点和匹配的目标点距离较近，则将这一对点定义为 Inlier

                if np.linalg.norm(dstCoor[:2] - dstPoints[i]) < threshold:

                    NumInlier = NumInlier + 1

                    pos\_Inlier.append((srcPoints[i], dstPoints[i]))

        if MaxInlier < NumInlier:

            MaxInlier = NumInlier

            Best\_H = H

            save\_Inlier\_pos = pos\_Inlier

    return Best\_H, save\_Inlier\_pos

经过实验，得到最终的H矩阵为：

[ [ 6.11594071e-01 -3.52729237e-02 2.65329770e+02]

[-2.16686810e-01 8.90029535e-01 2.89016318e+01]

[-6.17180321e-04 -1.52773928e-05 1.00000000e+00]]

再次绘制特征匹配的结果，可以发现，很明显的精准了很多。

