计算机视觉与应用实践实验报告

目录

一、实验目的

- 学习单应性变换，代码实现计算图片之间的单应性变换。
- 厘清实验步骤并对结果进行分析。

二、实验原理

本部分主要讲述单应性变换的原理，包括单应性变换定义，单应性矩阵的求解和单应性矩阵的参数化形式。

2.1 单应性变换定义

图像的单应性变换是指两个二维平面之间的投影变换过程，二者之间的映射关系便是图像的单应性变换。比如，一个二维平面的地面和它在照相机中的成像平面便是一个单应性变换例子。现定义三维空间中存在一个点 P 以及该点在成像设备上的投影点 p，二者之间的映射可以用矩阵相乘运算来表达。使用齐次坐标表示这两个点，则有以下公式成立：

$$P = [X \quad Y \quad Z \quad 1]^T$$

$$p = [x \quad y \quad 1]^T$$

其中 P 表示三维空间的齐次坐标，p 表示二维图像平面的齐次坐标，二者的映射关系则可以将单应性矩阵表示为(P 进行映射时不使用 Z 轴，Z 轴表示景深不影响映射时的坐标位置)：

$$p = sHP$$

这里参数 s 表示任意尺度的缩放因子，H 表示单应性矩阵。通过单应性矩阵 H 变换，可将图像投影至任意平面。图像中目标的平移、旋转、透视变换等都可通过单应性矩阵 H 完成映射。

图像中目标的平移：

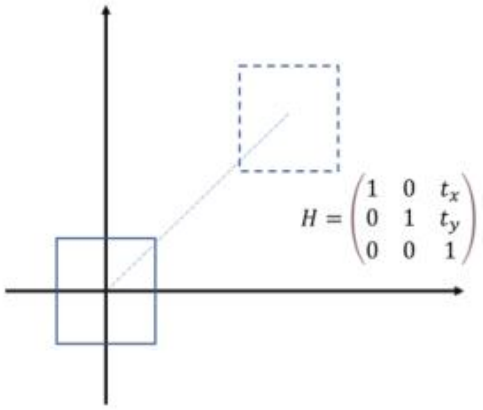


图 2.1 单应性矩阵表示平移

如上图 2.1 所示，将图形从原点平移至坐标系另一坐标，可用数学表达式描述为：$x' = x + t_x, y' = y + t_y$。也可用齐次坐标和矩阵乘法表示:

$$\begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$$

图像中目标的旋转：

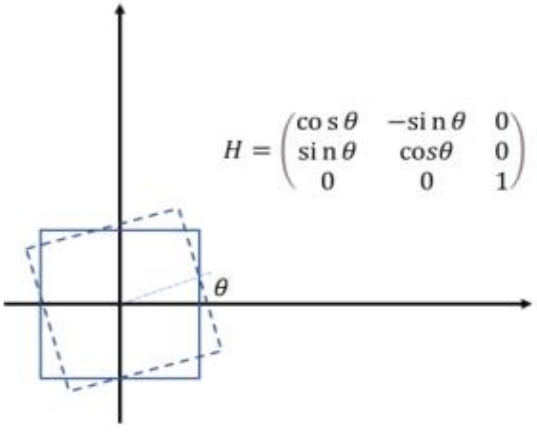


图 2.2 单应性矩阵表示旋转

如上图 2.2 所示，将该目标以坐标$(0, 0)$为中心，沿逆时针方向旋转θ角度，用公式可表示为：$x' = x \cdot cos\theta - y \cdot sin\theta, y' = x \cdot sin\theta + y \cdot cos\theta$。又可写成矩阵相乘的形式如下：

$$\begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \begin{bmatrix} cos\theta & -sin\theta & 0 \\ sin\theta & cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$$

图像中目标的仿射变换：

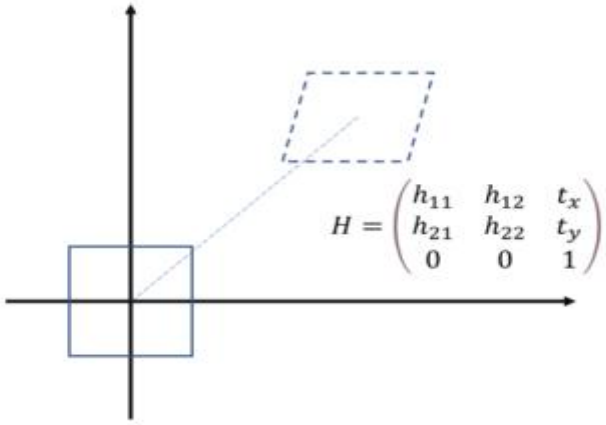


图 2.3 单应性矩阵表示仿射变换

从上图 2.3 中可以看到，相较于平移和旋转运动(不改变物体形状)，仿射变换不仅改变目标的坐标位置，此外还能够令目标形状发生改变，但仍然保持目标本身的平直性，即原本平行的线经过仿射变换后仍然保持平行关系。

图像中目标的透视变换：

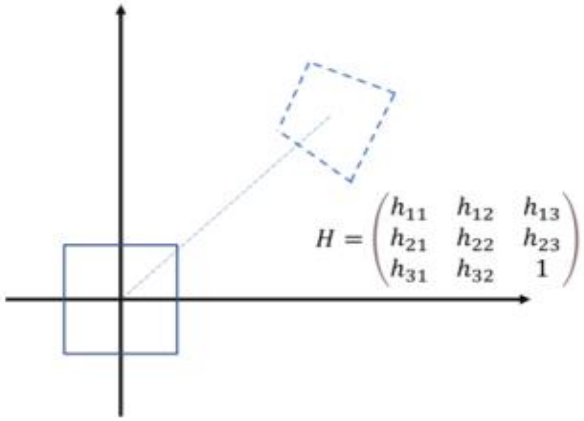


图 2.4 单应性矩阵表示透视变换

从上图 2.4 中可以看出，透视变换不仅仅改变目标的形状，同时使目标中原来平行的线条不再保持平行关系，彻底改变了目标的形状，这种变换类似于在不同角度观察大楼平面，不同角度下大楼平面呈现不同的形状。此外，也可得到结论，透视变换(等同单应性变换)包含图像仿射变换，比仿射变换含义更广，可描述更加复杂的图像几何变换，应用到更复杂的视觉任务中。

2.2 单应性矩阵的求解

根据上述图像单应性矩阵的基本理论，现定义图像中点$a=(x,y,1),b=(x_1,y_1,1)$分别表示两幅图像中互相匹配的两个坐标，H表示单应性矩阵，则根据单应性变换原理有:

$$\begin{pmatrix}x_1\\y_1\\1\end{pmatrix}=\begin{pmatrix}h_{11}&h_{12}&h_{13}\\h_{21}&h_{22}&h_{23}\\h_{31}&h_{32}&h_{33}\end{pmatrix}\begin{pmatrix}x\\y\\1\end{pmatrix}\Leftrightarrow b=Ha^T$$

对矩阵乘法进行展开，即可得到方程组：

$$\begin{cases}x_1=h_{11}x+h_{12}y+h_{13}\\y_1=h_{21}x+h_{22}y+h_{23}\\1=h_{31}x+h_{32}y+h_{33}\end{cases}$$

将上述第三个方程代入至前两个方程中可得：

$$\begin{cases}x_1=\dfrac{x_1}{1}=\dfrac{h_{11}x+h_{12}y+h_{13}}{h_{31}x+h_{32}y+h_{33}}\\y_1=\dfrac{y_1}{1}=\dfrac{h_{21}x+h_{22}y+h_{23}}{h_{31}x+h_{32}y+h_{33}}\end{cases}$$

进一步变形后可推导出：

$$\begin{cases}h_{11}x+h_{12}y+h_{13}=h_{31}xx_1+h_{32}yx_1+h_{33}x_1\\h_{21}x+h_{22}y+h_{23}=h_{31}xy_1+h_{32}yy_1+h_{33}y_1\end{cases}$$

经过移项后有：

$$\begin{cases}0=h_{31}xx_1+h_{32}yx_1+h_{33}x_1-(h_{11}x+h_{12}y+h_{13})\\0=h_{31}xy_1+h_{32}yy_1+h_{33}y_1-(h_{21}x+h_{22}y+h_{23})\end{cases}$$

根据线性代数知识，可将方程化为矩阵和向量相乘，则有：

$$\begin{bmatrix}-x&-y&-1&0&0&0&xx_1&yx_1&x_1\\0&0&0&-x&-y&-1&xy_1&yy_1&y_1\end{bmatrix}h=0$$

其中$h=[h_{11},h_{12},h_{13},h_{21},h_{22},h_{23},h_{31},h_{32},h_{33}]^T$是一个 9 维列向量，若令：

$$A=\begin{bmatrix}-x&-y&-1&0&0&0&xx_1&yx_1&x_1\\0&0&0&-x&-y&-1&xy_1&yy_1&y_1\end{bmatrix}$$

则方程可记为;

$$Ah=0$$

这里的$A\in R^{2\times9}$，这是由一对点得到的矩阵方程。由于单应性矩阵的计算使用齐次坐标系，也就是说存在一个任意尺度的缩放因子，当对矩阵元素乘以任意一个非零常数 k 后坐标之间的映射关系依然不变，故 3×3 的单应性矩阵有 9 个参数，但实质上只有 8 个自由度。由于矩阵 H 有 8 个自由度，而一对点提供两个线性方程组, 因此需要至少 4 对点提供 8 个线性方程, 即可求解出矩阵 H 的值。

假设现在有 $n\geq4$ 个点对，则得到矩阵 $A\in R^{2n\times9}$，可以直接对 A 进行 SVD(Singular Value Decomposition)分解后得 $U*D*V^T$，随后取 V 的最后一列便是矩阵 H 的值。

奇异值分解（SVD）:设$A\in C_r^{m\times n}$，则存在正交矩阵$U\in C^{m\times m}$与正交矩阵$V\in C^{n\times n}$使得

$$A=U\begin{bmatrix}\Sigma_r&0\\0&0\end{bmatrix}V^T=UDV^T$$

其中，$\Sigma_r=diag(\sigma_1,\cdots,\sigma_r)$是矩阵$A$的正奇异值。

求解齐次线性方程组($\boldsymbol{Ah=0}$)：问题等价于$min_h\|Ah\|^2$，则

$$\begin{aligned} & min_h\|Ah\|^2 \\ =\ & min_h\|UDV^Th\|^2 \\ =\ & min_h\|DV^Th\|^2 \end{aligned}$$

令$y=V^Th$，则上式变为$min_y\|Dy\|^2=min_y(\sigma_1^2y_1^2+\sigma_2^2y_2^2+\cdots+\sigma_r^2y_r^2)$，其中 r ≤ n。因为$\sigma_1>\sigma_2>\cdots>\sigma_r>0$，所以不管 r 是否等于 n，在‖h‖ = 1 的前提下的最优解均在$y=[0,0,\cdots,1]^T$时取得（r = n 时存在最小二乘解，r < n 时存在数值解），此时$h=Vy$，即V的最后一列。

由上述单应性矩阵求解过程可知，至少需要 4 对且任意三点不共线的关键点坐标才能计算出两幅图像之间的单应性矩阵。在传统算法中，使用局部特征描述符进行匹配得到大量坐标映射关系。但是这些映射关系并不全都是正确的，往往存在一部分坐标匹配是有误差或不正确，即噪声数据。此时，可以使用 RANSAC(Random sample consensus 随机抽样一致)算法排除噪点数据，从而估计出最佳的单应性矩阵参数。

RANSAC 的作用是从一组包含噪声的数据集合中估计出一个近似的数学模型。该算法包含两大特性：随机性与假设性。RANSAC 算法的随机性体现在依靠正常数据在全部数据中出现的概率随机对被观测数据进行抽样，经过多次迭代随机抽样，从而估计出一个近似正确的数学模型。算法的假设性是指假设随机选取出的抽样数据是正常数据，在这个假设基础之上估计出满足问题的模型，进而预测剩余未被抽样到的数据。随后根据所有数据在当前模型上的贴合程度进行评分表示当前估计数学模型的优劣程度，如此迭代抽样，直至寻找到评分最好的数学模型。此时该数学模型能够拟合大多数正常数据，是最理想的数学模型估计。算法 2.1 描述 RANSAC 算法求解单应性矩阵的过程:

RANSAC 算法求解单应性矩阵

输入：给定匹配点集$X=[x_1,x_2,\cdots,x_N]_{3\times N}$和$Y=[y_1,y_2,\cdots,y_N]_{3\times N}$，采用齐次坐标，其行数为 3，列数为 N 即坐标点个数。

输出：单应性矩阵

1. 随机选择 4 对点，对坐标进行归一化处理使点集的中心位于原点，随后求解出由这四对归一化的匹配点得出的单应性矩阵H。
2. 根据求解出的单应性矩阵H计算在该单应性变换下，点集X的映射结果$\widetilde{Y}$。
3. 计算$\widetilde{Y}$和Y对应点之间的欧式距离，如距离小于某个预设的阈值，则将该点视为一个内点，依次循环计算在该H模型下内点的数量。
4. 对 1-3 步骤重复，直至找到一个包含最多内点的H矩阵，认为该单应性矩阵是最佳的单应性矩阵。

根据上述单应性变换含义、单应性变换实现图像几何变换以及单应性变化求解过程可知，单应性矩阵可以解决两个问题：第一，使用单应性矩阵来表示两个不同平面之间的单应性变换关系；第二，通过将单应性矩阵作用于一个图像平面，实现平面之间的映射。因此，对于单应性变换估计直接的应用就是实现图像的拼接对齐，将两个不同拍摄设备拍摄角度的图像拼接成一张图像，从而拥有更宽阔的视野范围；实现计算机图形学中的纹理渲染，矫正相机拍摄时可能产生的图像扭曲等方面。

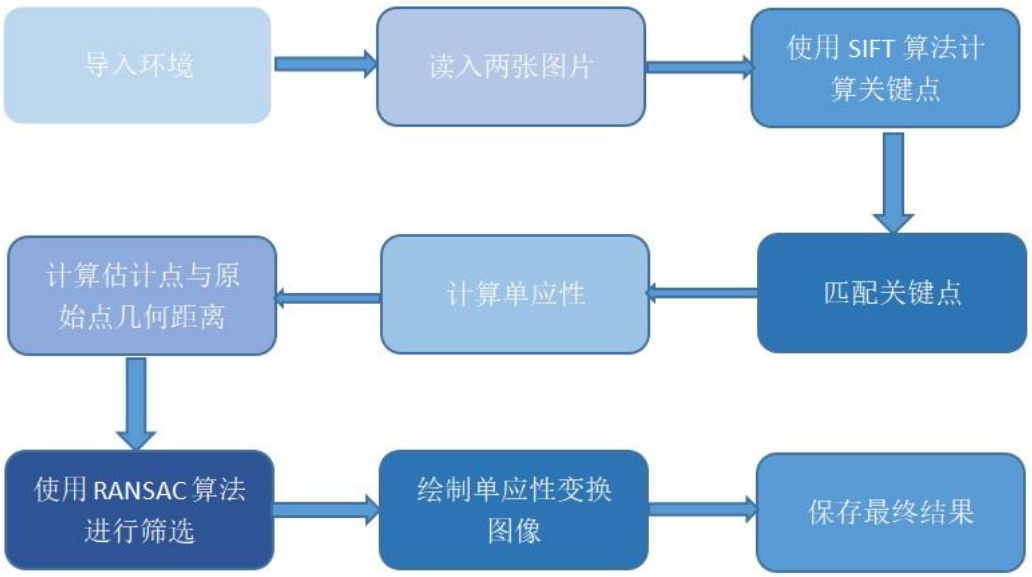
2.3 单应性矩阵的参数化形式

单应性矩阵最直观的参数化形式即 3×3 的矩阵形式，通过矩阵乘法运算可将一个平面的齐次坐标映射到另一平面中。在深度学习中，若将 3×3 矩阵中 9 个元素展开为一个一维向量作为卷积神经网络模型的预测回归值，是不太适用的。首先，若用单应性矩阵作为回归目标，这种参数化形式同时将旋转因子$[h_{11}h_{12};\ h_{21}h_{22}]$和平移因子$[h_{13}h_{23}]$混在一起，增大了问题的难度。其次，使用 3x3 单应性矩阵作为网络预测目标时，若预测的矩阵元素出现细微偏差，可能会给图像的坐标映射带来巨大的偏差。最后，由于 3×3 单应性矩阵元素的方差较大，很难保证所估计出的单应性矩阵是非奇异矩阵，即保证源图像和目标图像是能够通过求解的单应性矩阵互相转变。因此在基于深度学习的单应性变换估计算法中，一般使用基于坐标偏移量的参数化形式，称为四角点坐标偏移矩阵。与 3×3 矩阵相比,它更加适用基于深度学习的单应性变换估计算法。现定义$\Delta u_1 = u_1' - u_1$表示第一个角点坐标在u方向上的偏移量，$\Delta v_1 = v_1' - v_1$表示第一个角点坐标在v方向上的偏移量，则四角点坐标偏移矩阵可用如下公式表示：

$$H_{4pt} = \begin{pmatrix} \Delta u_1 & \Delta v_1 \\ \Delta u_2 & \Delta v_2 \\ \Delta u_3 & \Delta v_3 \\ \Delta u_4 & \Delta v_4 \end{pmatrix}$$

与矩阵参数化形式相同，四角点坐标偏移矩阵参数含有 8 个未知元素。只要获取到四个角点坐标的偏移量，两种参数化形式是可以进行相互转换的，利用 DLT(Direct Linear Transform)直接线性变换即可实现。

三、实验步骤



四、代码实现

4.1 SIFT 算法寻找关键点

由于在求单应性矩阵之前需要找出四个对应点。因此，首先调用 cv2 库的 SIFT 算法，获取关键点和描述子。

```
# 用SIFT算法寻找关键点
def findFeatures(img):
    print("Finding Features...")
    sift = cv2.SIFT_create()
    keypoints, descriptors = sift.detectAndCompute(img, None)

    img = cv2.drawKeypoints(img, keypoints, None)
    cv2.imwrite('sift_keypoints.png', img)

    return keypoints, descriptors
```

4.2 匹配关键点

根据描述子匹配关键点，绘制出匹配了关键点之后的图像。

```
# 匹配给定关键点、描述符和图像列表的特征
def matchFeatures(kp1, kp2, desc1, desc2, img1, img2):
    print("Matching Features...")
    matcher = cv2.BFMatcher(cv2.NORM_L2, True)
    matches = matcher.match(desc1, desc2)
    matchImg = drawMatches(img1,kp1,img2,kp2,matches)
    cv2.imwrite('Matches.png', matchImg)
    return matches
```

4.3 计算单应性矩阵

根据第二部分的公式计算单应性矩阵。

```
#计算出对应四个点的单应性
def calculateHomography(correspondences):
    #循环处理对应关系并创建集合矩阵
    aList = []
    for corr in correspondences:
        p1 = np.matrix([corr.item(0), corr.item(1), 1])
        p2 = np.matrix([corr.item(2), corr.item(3), 1])

        a2 = [0, 0, 0, -p2.item(2) * p1.item(0), -p2.item(2) * p1.item(1), -p2.item(2) * p1.item(2),
              p2.item(1) * p1.item(0), p2.item(1) * p1.item(1), p2.item(1) * p1.item(2)]
        a1 = [-p2.item(2) * p1.item(0), -p2.item(2) * p1.item(1), -p2.item(2) * p1.item(2), 0, 0, 0,
              p2.item(0) * p1.item(0), p2.item(0) * p1.item(1), p2.item(0) * p1.item(2)]
        aList.append(a1)
        aList.append(a2)

    matrixA = np.matrix(aList)
    u, s, v = np.linalg.svd(matrixA)
    h = np.reshape(v[8], (3, 3))
    h = (1/h.item(8)) * h
    return h
```

4.4 计算出估计点和原始点的几何距离

计算出距离，方便后续调用 RANSAC 算法和绘制图像等处理。

```
# 计算估计点与原始点之间的几何距离
def geometricDistance(correspondence, h):

    p1 = np.transpose(np.matrix([correspondence[0].item(0), correspondence[0].item(1), 1]))
    estimatep2 = np.dot(h, p1)
    estimatep2 = (1/estimatep2.item(2))*estimatep2

    p2 = np.transpose(np.matrix([correspondence[0].item(2), correspondence[0].item(3), 1]))
    error = p2 - estimatep2
    return np.linalg.norm(error)
```

4.5 使用 RANSAC 算法进行筛选

在传统 SIFT 算法中，使用局部特征描述符进行匹配得到大量坐标映射关系。但是这些映射关系并不全都是正确的，往往存在一部分坐标匹配是有误差或不正确，即噪声数据。此时，需要使用 RANSAC 算法排除噪点数据，从而估计出最佳的单应性矩阵参数。

```python
# 运行RANSAC算法，随机创建单应性图
def ransac(corr, thresh):
    maxInliers = []
    finalH = None
    for i in range(1000):
        # 随机取四个点计算单应性
        corr1 = corr[random.randrange(0, len(corr))]
        corr2 = corr[random.randrange(0, len(corr))]
        randomFour = np.vstack((corr1, corr2))
        corr3 = corr[random.randrange(0, len(corr))]
        randomFour = np.vstack((randomFour, corr3))
        corr4 = corr[random.randrange(0, len(corr))]
        randomFour = np.vstack((randomFour, corr4))

        # 调用这些点上的单应性函数
        h = calculateHomography(randomFour)
        inliers = []

        for i in range(len(corr)):
            d = geometricDistance(corr[i], h)
            if d < 5:
                inliers.append(corr[i])

        if len(inliers) > len(maxInliers):
            maxInliers = inliers
            finalH = h
        print("Corr size: ", len(corr), " NumInliers: ",
            len(inliers), "Max inliers: ", len(maxInliers))

        if len(maxInliers) > (len(corr)*thresh):
            break
    return finalH, maxInliers
```

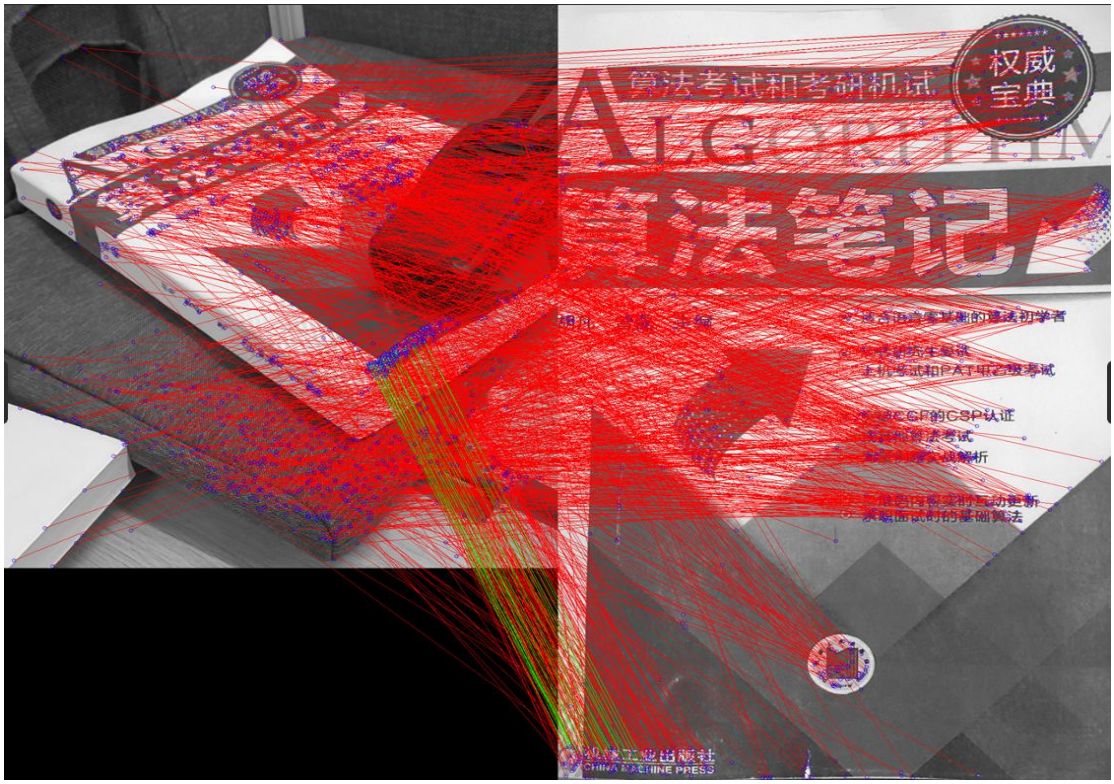
4.6 绘制单应性变换图像

标出对应关键点，将其用不同颜色的直线连接起来，绘制出单应性变换后的图像。

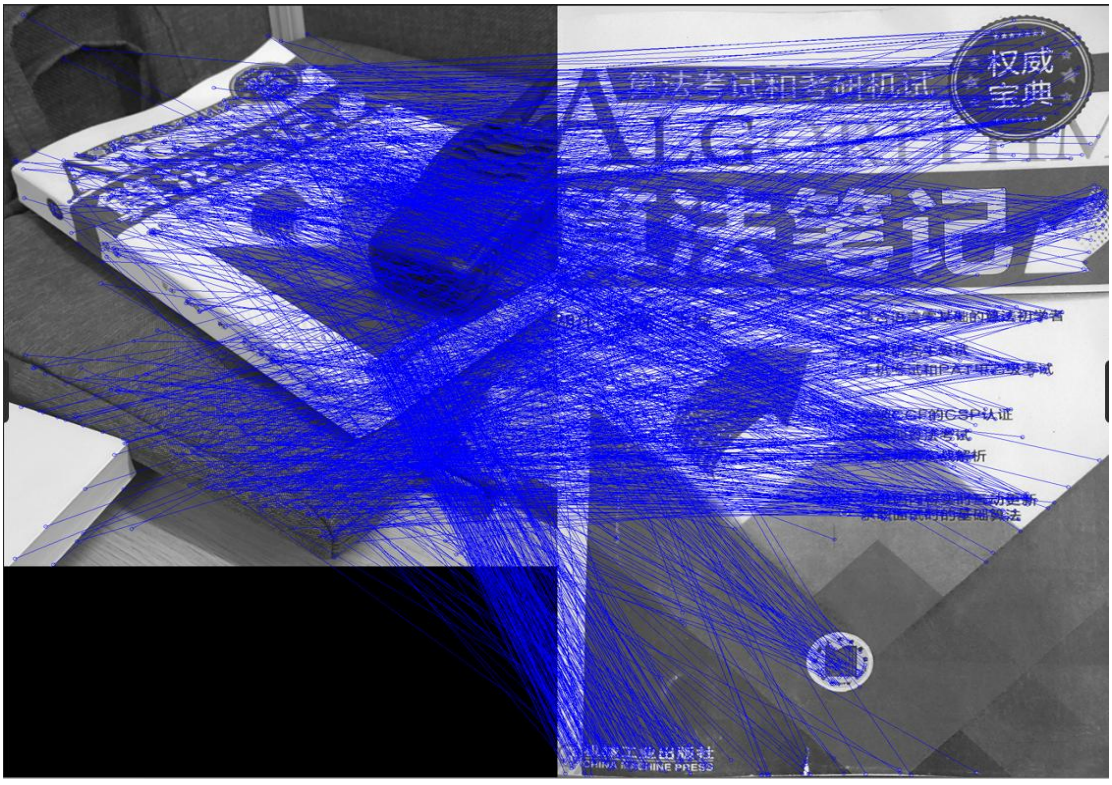
```python
def drawMatches(img1, kp1, img2, kp2, matches, inliers = None):
    rows1 = img1.shape[0]
    cols1 = img1.shape[1]
    rows2 = img2.shape[0]
    cols2 = img2.shape[1]

    out = np.zeros((max([rows1,rows2]),cols1+cols2,3), dtype='uint8')
    out[:rows1,:cols1,:] = np.dstack([img1, img1, img1])
    out[:rows2,cols1:cols1+cols2,:] = np.dstack([img2, img2, img2])

    for mat in matches:
        img1_idx = mat.queryIdx
        img2_idx = mat.trainIdx

        (x1,y1) = kp1[img1_idx].pt
        (x2,y2) = kp2[img2_idx].pt

        inlier = False

        if inliers is not None:
            for i in inliers:
                if i.item(0) == x1 and i.item(1) == y1 and i.item(2) == x2 and i.item(3) == y2:
                    inlier = True

        cv2.circle(out, (int(x1),int(y1)), 4, (255, 0, 0), 1)
        cv2.circle(out, (int(x2)+cols1,int(y2)), 4, (255, 0, 0), 1)

        if inliers is not None and inlier:
            cv2.line(out, (int(x1),int(y1)), (int(x2)+cols1,int(y2)), (0, 255, 0), 1)
        elif inliers is not None:
            cv2.line(out, (int(x1),int(y1)), (int(x2)+cols1,int(y2)), (0, 0, 255), 1)

        if inliers is None:
            cv2.line(out, (int(x1),int(y1)), (int(x2)+cols1,int(y2)), (255, 0, 0), 1)
    return out
```

五、实验结果

本次实验做了对比研究。下图是两组不同的图片，第一组图片旋转角度很大，图片可匹配的信息很少，可以看出最终的内点数为 43。而第二组图片拍摄角度很好，可匹配的信息很多，最终的内点数为 413。



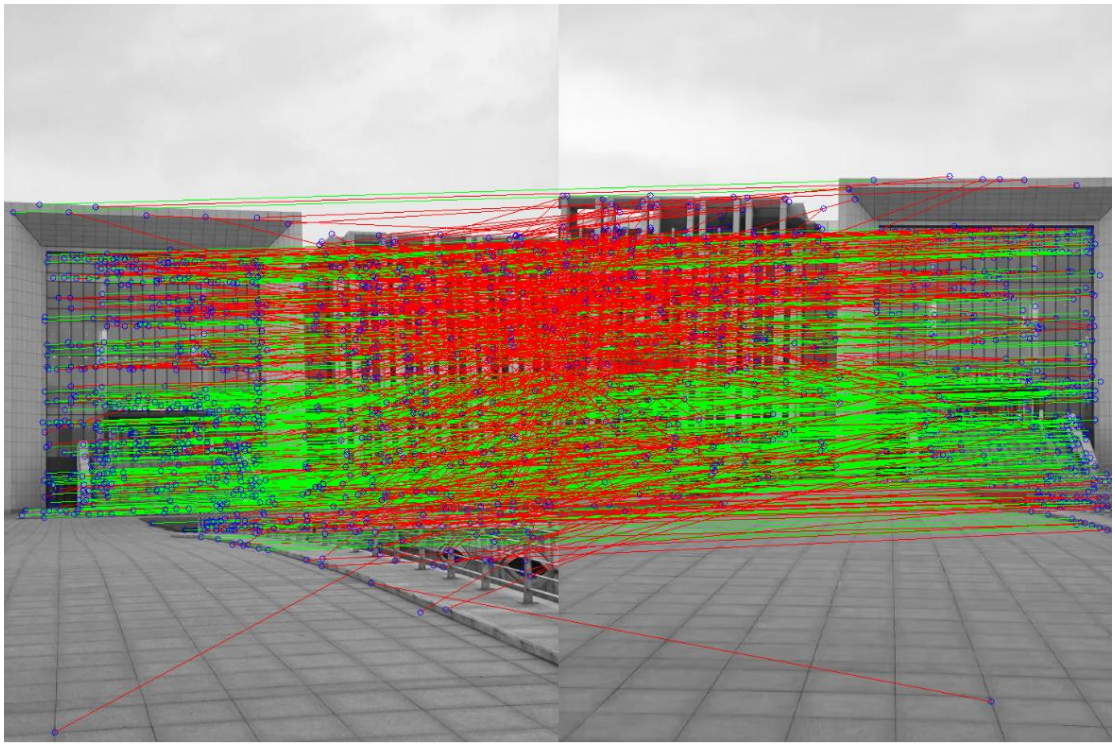
下图时最终的关键点匹配图片。

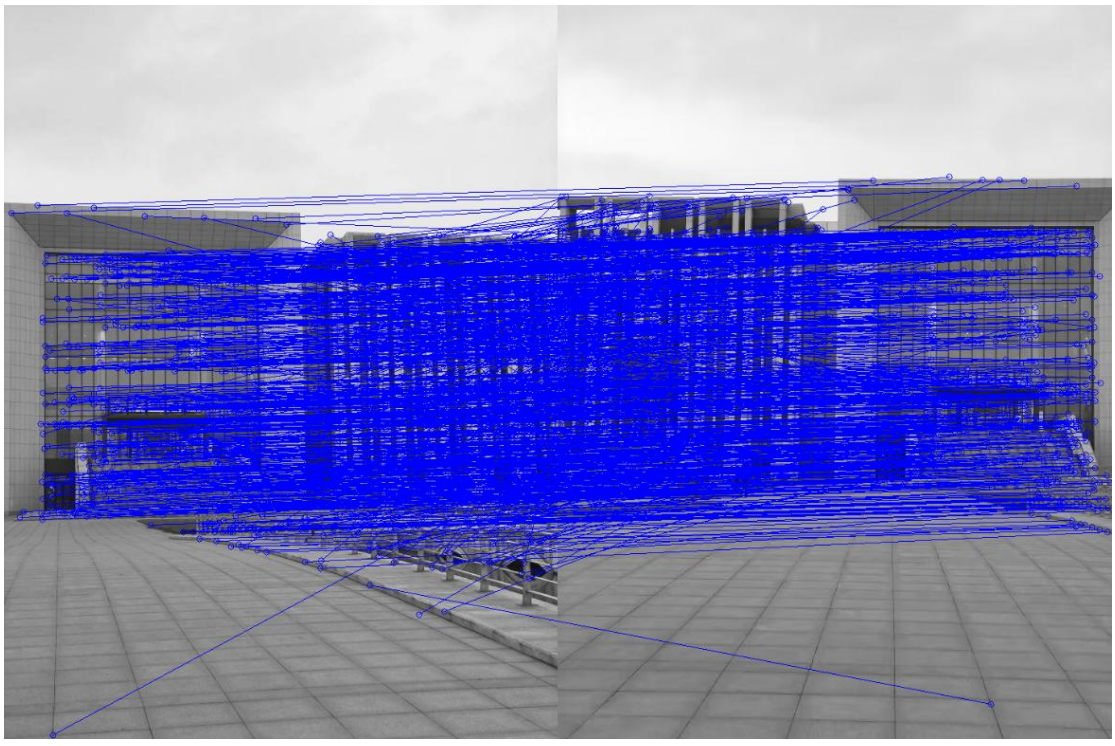


单应性矩阵：

$$\begin{bmatrix} 1.26500491e-01 & -1.88162804e-01 & 5.87559072e+01 \\ -2.17943307e-01 & -1.25872140e+00 & 1.34450783e+03 \\ -2.38584973e-04 & -8.98277546e-04 & 1.00000000e+00 \end{bmatrix}$$

上述是根据所拍摄的两张图片计算出来的单应性矩阵。最终的内点数为 43。





单应性矩阵：

$$\begin{bmatrix} 8.76595206e-01 & -1.35701522e-02 & 3.69239742e+02 \\ -7.15763313e-02 & 9.69787445e-01 & -2.24026627e+01 \\ -1.68824898e-04 & -1.56829658e-06 & 1.00000000e+00 \end{bmatrix}$$

上述是根据所拍摄的两张图片计算出来的单应性矩阵。最终的内点数为 413。

六、实验总结和分析

单应性变换是计算机视觉中常用的图像变换方法之一，它可以用来将一个平面上的点映射到另一个平面上。计算两张图片之间的单应性变换可以用来进行图像配准、物体检测、图像拼接等任务。以下是关于计算两张图片的单应性变换的实验总结。

首先，进行实验前的准备工作非常重要。需要准备两张待处理的图片，这两张图片需要有明显的区别，并且至少存在一些共同的特征点。另外，需要选择一种适合的特征点检测和匹配算法，如 SIFT、SURF、ORB 等。本次实验主要用了 SIFT 算法。此外，还需要进行一些预处理操作，例如图像去噪、图像配准等。

其次，在实验中需要进行特征点检测和匹配。在这个阶段中，使用选定的特征点检测和匹配算法对两张图片进行特征点提取和匹配。特征点匹配是整个实验的核心，需要通过选择正确的匹配算法和匹配策略来确保匹配的准确性和鲁棒性。

最后，进行单应性变换的计算和应用。在这个阶段中，使用 RANSAC 算法来估计单应性变换矩阵，将待处理的图片进行变换，得到最终的处理结果。此外，在进行单应性变换之前，需要进行一些预处理操作，如图像配准、坐标系转换等。

综上所述，计算两张图片的单应性变换需要进行多个阶段的操作，包括准备工作、特征点检测和匹配以及单应性变换计算和应用。在实验中需要选择适合的算法和策略，同时需要注意处理过程中的误差和异常情况，以确保最终处理结果的准确性和鲁棒性。此外，通过实验还可以培养对图像处理和计算机视觉的兴趣和能力，从而进一步拓展相关领域的知识和技能。