南京理工大学

NANJING UNIVERSITY OF SCIENCE & TECHNOLOGY

# “计算机视觉与应用实践”

# 课程作业

姓 名：余 涛 学 号：122106222833

专 业：计算机科学与工程

项目名称：计算图片之间的单应性变换

时间：2023 年 4 月 29 日

# 1、实验目标

（1）、单应性变换，计算图片之间的单应性变换；
（2）、需要详细的实验过程和结果分析。

# 2、具体要求

（1）、单应性变换，计算图片之间的单应性变换；
（2）、需要详细的实验过程和结果分析。

# 3、总体思路

单应性变换是将一个平面内的点映射到另一个平面内的二维投影变换。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）。

$$\begin{bmatrix} x' \\ y' \\ w' \end{bmatrix} = \begin{bmatrix} h_1 & h_2 & h_3 \\ h_4 & h_5 & h_6 \\ h_7 & h_8 & h_9 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix}$$

或者：

$$x' = Hx$$

对于图像平面内（甚至是三维中的点，后面我们会介绍到）的点，齐次坐标是个非常有用的表示方式。点的齐次坐标是依赖于其尺度定义的，所以，x=[x,y,w]=[ax,ay,aw]=[x/w,y/w,1]都表示同一个二维点。因此，单应性矩阵 H 也仅依赖尺度定义，所以，单应性矩阵具有 8 个独立的自由度。我们通常使用 w=1 来归一化点，这样，点具有唯一的图像坐标 x 和 y。这个额外的坐标是的我们可以简单地使用一个矩阵来表示变换。

# 4、项目实现

## 一、原理与实现

**1. 原理理解：**

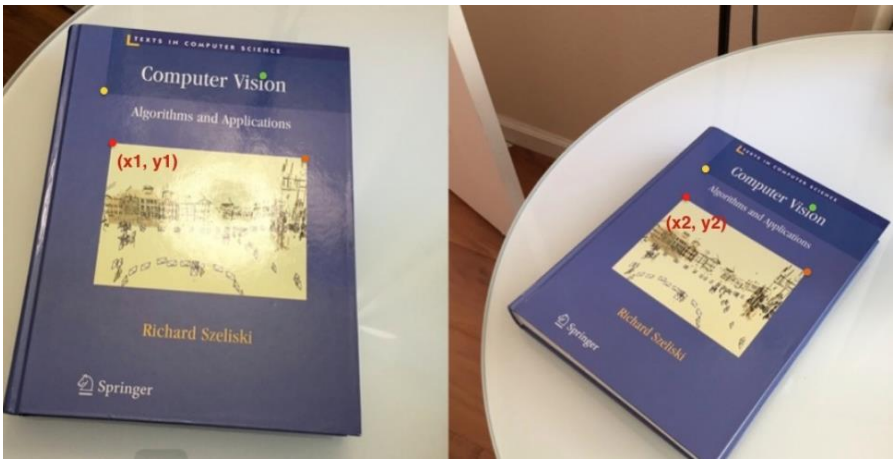
如图 1 中所示的两个平面图像（书的顶部）。红点表示两个图像中的同一物理点。在计算机视觉术语中，我们称这些为相应的点。下图中以四种不同的颜

色（红色、绿色、黄色和橙色）显示四个相应的点。单应性变换将一个图像中的点映射到另一个图像中的相应点。


图 1

进行点和变换的处理时。我们会按照列优先的原则存储这些点。因此 n 个二维点集将会存储为齐次坐标意义下的一个 3×n 数组。这种格式使得矩阵乘法和点的变换操作更加容易。对于其他的例子，比如对于聚类和分类的特征，我们将使用典型的行数组来存储数据。

如下图所示的平面的两幅图像。红点表示两幅图像中的相同物理点，我们称之为对应点。这里显示了四种不同颜色的四个对应点 - 红色，绿色，黄色和橙色。一个 Homography 是一个变换（3×3 矩阵），将一个图像中的点映射到另一个图像中的对应点。单应性变换其实就是一个平面到另一个平面的变换关系。




**2. 单应性矩阵的理解及直接线性变换算法：**

在齐次坐标中，假设一点 p(xi,yi,1)经过 H 矩阵的变换变为 p‘(xi’,yi’,1)，即 p’ =Hp，通常，对于直接线性变换，一个完全射影变换具有 8 个自由度，H 矩阵有 8 个自由度，这样至少需要 4 对特征点对求解。4 个特征点对可以建立 8

个方程。那么对于有 n 对特征点的情况(超定方程)，解 p’ = Hp 方程组可以转化为对齐次方程组 Ax = 0 的求解。而对 Ax = 0 的求解转化为 min ||Ax||2 的非线性优化问题（超定方程，通过最小二乘拟合得到近似解）。

$$\begin{bmatrix} wx' \\ wy' \\ w \end{bmatrix} = \begin{bmatrix} h_{00} & h_{01} & h_{02} \\ h_{10} & h_{11} & h_{12} \\ h_{20} & h_{21} & h_{22} \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \quad \begin{matrix} x_i' = \dfrac{h_{00}x_i + h_{01}y_i + h_{02}}{h_{20}x_i + h_{21}y_i + h_{22}} \\ y_i' = \dfrac{h_{10}x_i + h_{11}y_i + h_{12}}{h_{20}x_i + h_{21}y_i + h_{22}} \end{matrix}$$

$$\mathbf{x'} \qquad \mathbf{H} \qquad \mathbf{x}$$



进一步可得：

$$\begin{aligned} x_i' &= \frac{h_{00}x_i + h_{01}y_i + h_{02}}{h_{20}x_i + h_{21}y_i + h_{22}} \\ y_i' &= \frac{h_{10}x_i + h_{11}y_i + h_{12}}{h_{20}x_i + h_{21}y_i + h_{22}} \end{aligned}$$

$$\begin{aligned} x_i'(h_{20}x_i + h_{21}y_i + h_{22}) &= h_{00}x_i + h_{01}y_i + h_{02} \\ y_i'(h_{20}x_i + h_{21}y_i + h_{22}) &= h_{10}x_i + h_{11}y_i + h_{12} \end{aligned}$$



这样便可构造系数矩阵：

$$\begin{bmatrix} x_i & y_i & 1 & 0 & 0 & 0 & -x_i'x_i & -x_i'y_i & -x_i' \\ 0 & 0 & 0 & x_i & y_i & 1 & -y_i'x_i & -y_i'y_i & -y_i' \end{bmatrix}$$

通过系数矩阵我们可以构造出齐次线性方程组(Ax = 0)：

$$\begin{bmatrix} x_i & y_i & 1 & 0 & 0 & 0 & -x_i'x_i & -x_i'y_i & -x_i' \\ 0 & 0 & 0 & x_i & y_i & 1 & -y_i'x_i & -y_i'y_i & -y_i' \end{bmatrix} \begin{bmatrix} h_{00} \\ h_{01} \\ h_{02} \\ h_{10} \\ h_{11} \\ h_{12} \\ h_{20} \\ h_{21} \\ h_{22} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$



## 二、具体步骤及过程

### 1. OpenCV 中单应性变换的函数用法：

```
# pts_src and pts_dst 分别为原图像和目标图像中对应像素点
# 组成的 numpy arrays，对应点的个数不少于 4 组
h, status = cv2.findHomography(srcPoints, dstPoints[, method[, ransacReprojThreshold[, mask[, maxIters[, confidence]]]]])
```

```
# 计算得到的 h（homography）可用于仿射变换
# Size 为原图的大小
im_dst = cv2.warpPerspective(im_src, h, size)
```

## 2. 基于 OpenCV 交互实现单应性变换：

```
class GetRoiMouse():
    def __init__(self, img):
        self.lsPointsChoose = []
        self.tpPointsChoose = []
        self.pointsCount = 0  # 顶点计数
        self.pointsMax = 4  # 最大顶点个数
        self.mouseWindowName = 'get four top'
        self.img = img  # 输入的图像

    def mouseclick(self):  # 显示一个窗口
        cv2.namedWindow(self.mouseWindowName)
        # opecv 可以设置监听鼠标
        # setMouseCallback(windowName,onMouse())
        # 在 onMouse 中写点击鼠标时要进行的工作
        cv2.setMouseCallback(self.mouseWindowName, self.on_mouse)
        cv2.imshow(self.mouseWindowName, self.img)
        cv2.waitKey(0)
    # 检测当前点个数，满足要求时关闭图像显示窗口
    def checkPointsNum(self):
        if len(self.lsPointsChoose) == 4:
            print('I get 4 points!')
            cv2.destroyAllWindows()

    # OpenCV 的鼠标响应函数，可以在内部定义鼠标的各种响应
    def on_mouse(self, event, x, y, flags, param):
        # 左键点击
        if event == cv2.EVENT_LBUTTONDOWN:
            print('left-mouse')
            self.pointsCount += 1
            print(self.pointsCount)
            point1 = (x, y)
            # 画出点击的位置
            img1 = self.img.copy()
            cv2.circle(img1, point1, 10, (0, 255, 0), 2)
            self.lsPointsChoose.append([x, y])
            self.tpPointsChoose.append((x, y))
            # 将鼠标选的点用直线连起来
            for i in range(len(self.tpPointsChoose) - 1):
```

```
                cv2.line(img1, self.tpPointsChoose[i], self.tpPointsChoose[i + 1], (0, 0,
255), 2)
            cv2.imshow(self.mouseWindowName, img1)
            self.checkPointsNum()
```

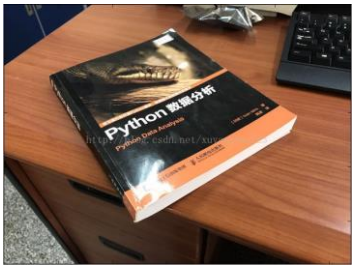
### 3. 定义实现方法主函数：

```
if __name__ == '__main__':
    # 读取原图像
    img_src = cv2.imread('img_src.jpg')
    # 调用上述类，获取原图像的像素点
    mouse1 = GetRoiMouse(img_src)
    mouse1.mouseclick()
    # 为方便下一步运算，需要将像素点的类型转换为浮点型数据
    pts_src = np.float32(mouse1.lsPointsChoose)

    # 读取目标图像
    img_dst = cv2.imread('img_dst.jpg')
    mouse2 = GetRoiMouse(img_dst)
    mouse2.mouseclick()
    # 获取对应点
    pts_dst = np.float32(mouse2.lsPointsChoose)
    # ----------------------------------------------------
    # 目标图像的尺寸
    dw, dh = img_dst.shape[1], img_dst.shape[0]
    # 通过 findHomography 计算变换矩阵 h
    h, status = cv2.findHomography(pts_src, pts_dst, cv2.RANSAC, 5)
    # 将变换矩阵 h 带入仿射变换实现矫正
    img_out = cv2.warpPerspective(img_src, h, (dw, dh))
    # 展示结果 图像，拼接起来
    images = np.concatenate((img_src[0: dh, 0:dw, 0:3], img_dst, img_out), axis=1)
    # 窗口显示
    cv2.imshow('homography', images)
    cv2.waitKey(0)
    cv2.destroyAllWindows()
```

## 三、结果展示

（1）、从左往右分别是原图，目标图，以及对原图进行单应性变化后的图片：




## 三、代码运行说明

（1）、代码使用 python 编写，运行时请使用 pycharm 等 python 编译器。

（2）、Homomorphic_change 总共包含:image_file,homoorphic.py,main.py。

（3）、使用时直接运行 homoorphic.py 文件，运行时首先要求手动标出这个图片中书本的四个角的位置，以此来计算单应性举证 H。此外 image_file 包含图片 book.jpg(原图)，book2.jpg(目标图)。

。