单应性变换实验报告

目录

一、实验目的

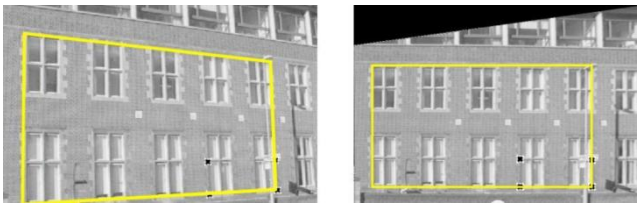
了解单应性变换原理，实现图像之间的单应性变换。

二、实验原理

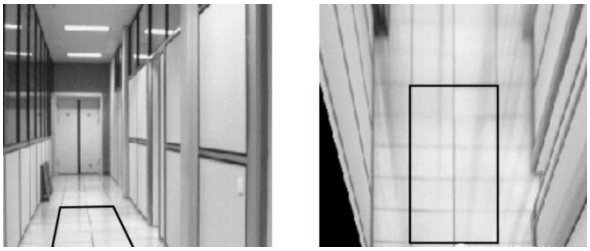
1、单应性在计算机视觉中的应用

图像在图像校正、图像拼接、相机位姿估计、视觉 SLAM 等领域有非常重要的作用。

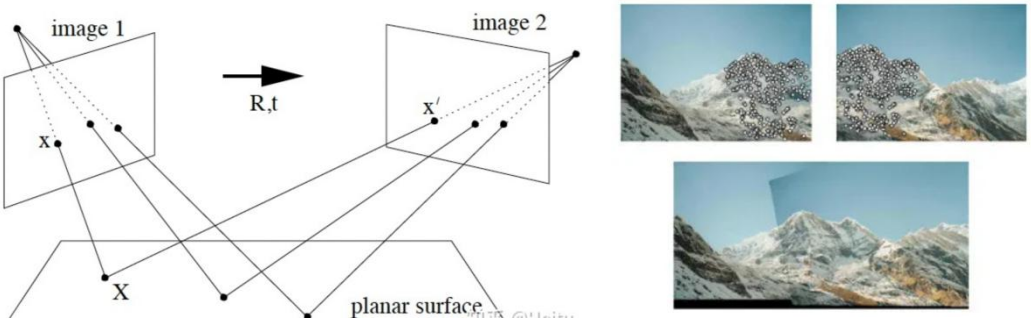
（1）图像校正：用单应性矩阵进行图像校正，至少需要四个对应点就可以实现。



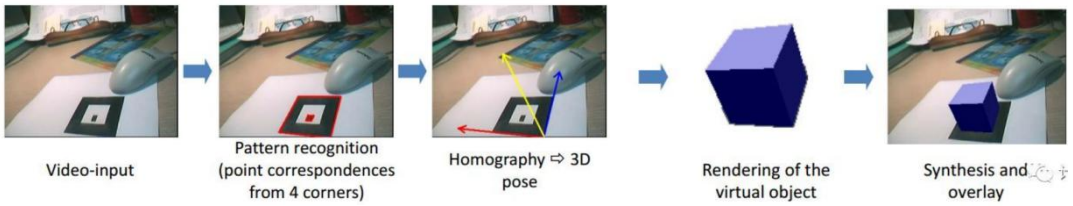
（2）视角变换：单应性矩阵用于视角变换，可将左边普通视图转换为右边的俯瞰视图。



（3）图像拼接：使用单应性矩阵，将不同角度拍摄的图像都转换到同一视角下，实现图像拼接。如下图所示，使用单应性矩阵可以将 image1 和 image2 都变换到同一平面。



（4）增强现实（AR）：平面二维标记图案（marker）经常用来做 AR 展示。根据 marker 不同视角下的图像可以方便的得到虚拟物体的位置姿态并进行显示。



2、单应变换

单应变换也就是射影变换，在生活中很常见，比如在一张拍摄的照片中，方形地砖不再是方形，汽车的圆形轮胎在画面中是椭圆，无穷远点也不再是无穷远点。但是，在经过单应变换后，直线仍然是直线。

单应性变化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。假设两张图像中的对应点对的齐次坐标为[x ', y ', 1]和[x, y, 1]，单应矩阵 H 定义为：

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

则有：

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

矩阵展开后有 3 个等式，将第 3 个等式代入前两个等式中可得如下公式，可以得知一个点对对应两个等式。

$$x'=\frac{h_{11}x+h_{12}y+h_{13}}{h_{31}x+h_{32}y+h_{33}}$$

$$y'=\frac{h_{21}x+h_{22}y+h_{23}}{h_{31}x+h_{32}y+h_{33}}$$

如果给定一个单应矩阵 H，给它的元素乘上同一个数，得到的单应 a*H 和 H 的作用相同，因为新单应矩阵无非把齐次点 x1 变成了齐次点 a*x1，都是一回事。因此可以把 a 假设成 $1/h_{22}$，那么 H 就变成了只有 8 个自由元素的矩阵。

8 自由度下 H 计算过程有两种方法：

第一种，直接设置 $h_{33}=1$，上述等式变为：

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

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

第二种，给 H 添加约束条件，将 H 矩阵模变为 1，如下：

$$x'=\frac{h_{11}x+h_{12}y+h_{13}}{h_{31}x+h_{32}y+h_{33}}$$

$$y'=\frac{h_{21}x+h_{22}y+h_{23}}{h_{31}x+h_{32}y+h_{33}}$$

$$h_{11}^2+h_{12}^2+h_{13}^2+h_{21}^2+h_{22}^2+h_{23}^2+h_{31}^2+h_{32}^2+h_{33}^2=1$$

以第二种方法继续推导，将上述等式乘以分母展开。

$$(h_{31}x+h_{32}y+h_{33})x' = h_{11}x+h_{12}y+h_{13}$$

$$(h_{31}x+h_{32}y+h_{33})y' = h_{21}x+h_{22}y+h_{23}$$

$$h_{11}x+h_{12}y+h_{13}-h_{31}xx'-h_{32}yx'-h_{33}x'=0$$

$$h_{21}x+h_{22}y+h_{23}-h_{31}xy'-h_{32}yy'-h_{33}y'=0$$

假如已知两幅图片中对应的 N 个点对（特征点匹配对），可以得到如下线性方程组。

$$
\underset{4\ \text{POINTS}}{} \underbrace{\begin{bmatrix} x_1 & y_1 & 1 & 0 & 0 & 0 & -x_1x_1' & -y_1x_1' & -x_1' \\ 0 & 0 & 0 & x_1 & y_1 & 1 & -x_1y_1' & -y_1y_1' & -y_1' \\ & & & & & & & & \end{bmatrix}}_{\text{2N x 9}} \underbrace{\begin{bmatrix} h_{11} \\ h_{12} \\ h_{13} \\ h_{21} \\ h_{22} \\ h_{23} \\ h_{31} \\ h_{32} \\ h_{33} \end{bmatrix}}_{\text{9 x 1}} = \underbrace{\begin{bmatrix} 0 \\ 0 \\ \\ \\ \\ \\ \\ \\ \end{bmatrix}}_{\text{2N x 1}}
$$

写成矩阵形式。

$$
\underset{\text{2Nx9}}{\mathbf{A}}\ \underset{\text{9x1}}{\mathbf{h}} = \underset{\text{2Nx1}}{\mathbf{0}}
$$

由于单应矩阵 H 包含了||H||=1 约束，因此根据上图的线性方程，8 自由度的 H 我们至少需要 4 对对应的点才能计算出单应矩阵。然而，在真实的应用场景中，计算的点对中都会包含噪声，比如点的位置偏差几个像素，甚至出现特征点对误匹配的现象，如果只使用 4 个点对来计算单应矩阵，那会出现很大的误差。因此，为了使得计算更精确，一般都会使用远大于 4 个点对来计算单应矩阵。另外上述方程组采用直接线性解法通常很难得到最优解，所以实际中一般会用其他优化方法。如奇异值分解，LM 算法等进行求解。

三．实验测试

1、代码流程

（1）读取替换图片，定义 vector 存储图像的四个角。读取待替换图像，定义结果图像。

```
// 读取图片
// 定义vector存储图像的四个角
Mat im_src = imread("../truck.jpg");

Size size = im_src.size();
vector<Point2f> pts_src;
pts_src.push_back(Point2f(0,0));
pts_src.push_back(Point2f(size.width - 1, 0));
pts_src.push_back(Point2f(size.width - 1, size.height -1));
pts_src.push_back(Point2f(0, size.height - 1 ));

// 读取广告牌图像
Mat im_dst = imread("../billboard.jpg");
Mat im_temp = im_dst.clone();
// 定义结果图像
userdata data;
data.im = im_temp;
imshow("Image", im_temp);
```

（2）定义鼠标事件，用于要替换图片的区域。

```
// 定义鼠标事件，用于点击广告牌要贴如汽车图片的区域
cout << "从左上角依次顺时针点击要贴入广告牌的区域" << endl;
setMouseCallback("Image", mouseHandler, &data);
waitKey(0);
```

```
void mouseHandler(int event, int x, int y, int flags, void* data_ptr)
{
    if  ( event == EVENT_LBUTTONDOWN )
    {
        userdata *data = ((userdata *) data_ptr);
        circle(data->im, Point(x,y),3,Scalar(0,255,255), 5, CV_AA);
        imshow("Image", data->im);
        if (data->points.size() < 4)
        {
            data->points.push_back(Point2f(x,y));
        }
    }

}
```

（3）计算单应矩阵，并将图片进行透视变换。

```
// 计算单应矩阵，并将car图片进行透视变换
Mat H = findHomography(pts_src, data.points, 0); //计算单应矩阵
warpPerspective(im_src, data.im, H, data.im.size()); //透视变换
imshow("perspect",data.im);
```

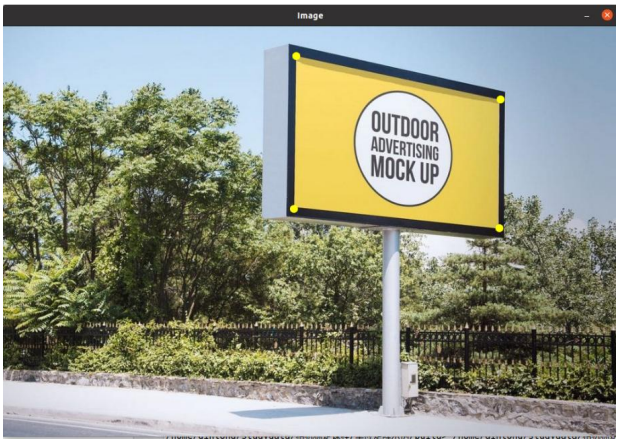
（4）进行图像填充并显示。

```
// 进行图像填充
Point pts_dst[4];
for (int i = 0; i < 4; i++)
{
    pts_dst[i] = data.points[i];
}
fillConvexPoly(im_dst, pts_dst, 4, Scalar(0), CV_AA);
im_dst = im_dst + data.im;

// 显示结果
imwrite("../truck_result.jpg", im_dst);
imshow("Image", im_dst);
waitKey(0);
```

2、结果展示

目标：顺时针点击广告牌的四个角，能够将要替换的图片按照单应变换和透视变换，转换到相应区域，并进行填充显示。



一共选了三张图片进行替换，原图、转换后的图像、替换后的图像显示如下。