**计算机视觉实践-练习4\_1**

**一、简介**

单应性变换，可以简单理解为用来描述物体在世界坐标系和像素坐标系之间的位置映射关系，对应的变换矩阵称为单应性矩阵。单应性在图像矫正，图像拼接等领域有着非常重要的作用。

**二、依赖条件**

matplotlib

numpy

cv2

**三、任务目标**

计算图片之间的单应性变换

**四、算法流程**

1.获取图片和坐标

1. （b）

图1 （a）目标图像（b）替换图像

首先加载目标图像和待替换的图像，使用鼠标操作分别获取目标图像和替换图像需要进行操作的位置信息。使用opencv中的鼠标响应函数，标记图像中的四个点，并用直线连接，将四个点的位置信息存储起来。

文本

描述已自动生成

图2 使用on\_mouse函数获取待操作图像的位置信息

2.计算单应矩阵

通过findHomography计算变换矩阵h，将变换矩阵h代入仿射变换实现矫正。

屏幕上有字

描述已自动生成

图3 经过仿射变换处理后的替换图像

3.图像拼接

使用opencv中的copyTo（）函数将处理后的图像进行拼接

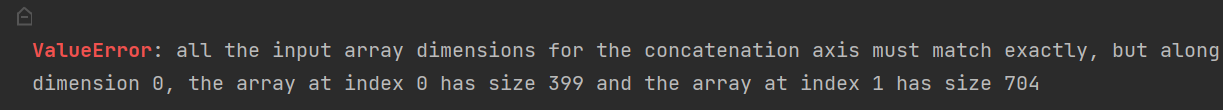
图形用户界面

描述已自动生成

图4 拼接后的图像

**五、实验中遇到的问题**

1. 使用np.concatenate()时遇到以下问题



解决方法：使用cv2.copyTo()进行图像拼接

**计算机视觉实践-练习4\_2**

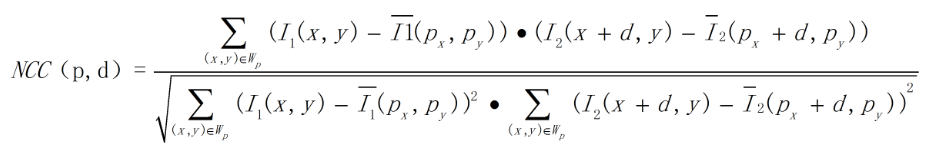
**一、简介**

立体匹配是立体视觉研究中的关键部分。其目标是在两个或多个视点中匹配相应像素点，计算视差。

**NCC视差匹配法**

对于原始的图像内任意一个像素点 (px,py)构建一个n × n的邻域作为匹配窗口。然后对于目标相素位置(px+d,py)同样构建一个n × n大小的匹配窗口，对两个窗口进行相似度度量，注意这里的dd dd有一个取值范围。对于两幅图像来说，在进行NCC计算之前要对图像处理，也就是将两帧图像校正到水平位置，即光心处于同一水平线上，此时极线是水平的，否则匹配过程只能在倾斜的极线方向上完成，这将消耗更多的计算资源。

NCC计算公式如公式（1）下所示：

（1）

**二、依赖条件**

Numpy

PIL

Matplotlib

Scipy

**三、实验目的**

通过立体匹配得到两张图像的视差图。

**四、实验内容**

（一）算法流程：

1.采集图像

2.极线校正：校正的目的是使两帧图像极线处于水平方向，或者说是使两帧图像的光心处于同一水平线上。

3.特征匹配：右视图中与左视图待测像素同一水平线上相关性最高的即为最优匹配。完成匹配后，我们需要记录其视差d，即待测像素水平方向xl与匹配像素水平方向xr之间的差值d=xr−xl，最终我们可以得到一个与原始图像尺寸相同的视差图D。

（二）实验结果与分析：

原图：

分别使用普通滤波器和高斯滤波器：

图表

描述已自动生成

图表, 散点图

描述已自动生成

图表, 散点图

描述已自动生成

图表, 散点图

描述已自动生成

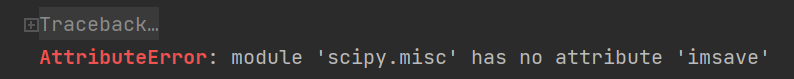
从上到下窗口值分别为3,5,7,9。

从这四张图片中，我们可以看到随着窗口值的变大，图像噪声逐渐减少，但同时细节也逐渐变少。一些轮廓明显的如画布，绿色玩偶都随着窗口值变大轮廓更加清晰。但是后面的玩偶等，细节越往后越体现不出来。

使用高斯滤波器，减少噪声的效果更明显，但是细节丢失也更加严重。窗口值变大后，细节丢失就非常的严重了。基本上看不到粉色玩偶。

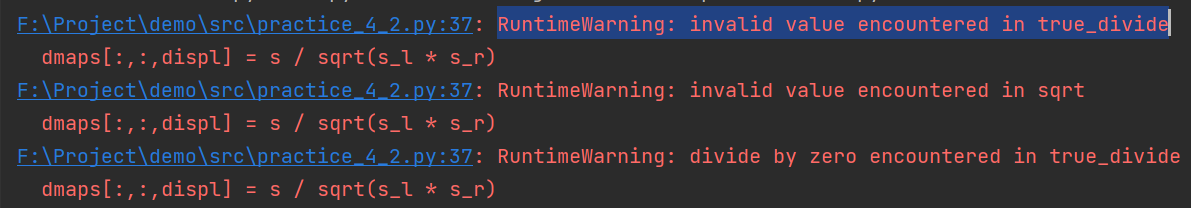
**五、实验中遇到的问题**

1.保存实验结果的时候遇到以下问题



解决方法：这是因为1.20版本之前的scipy有imsave，但是1.20版本之后换成了imadeio.imwrite

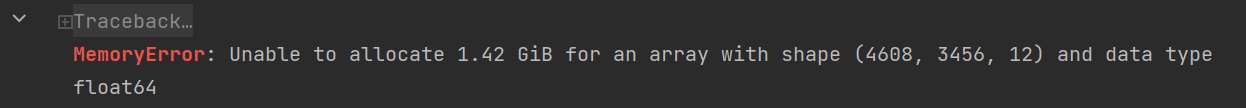
2.



解决方法：这是因为在使用numpy时出现了0除以0的情况，只要添加以下代码就能解决。



3.出现cpu内存不够用的情况



解决方法：这是因为输入的图片太大，而我又使用了循环处理，导致cpu不够用。只要把输入图片改小就可以了。