计算机视觉应用与实践实验报告

目录

一、实验目的

- 学习图像立体匹配算法，了解不同的处理方式，掌握重要步骤
- 重点学习 NCC 视差匹配方法，即给定左右两张视图，根据 NCC 计算视差图
- 代码实现立体匹配，输出匹配后的视差图

二、实验原理

本部分主要介绍立体匹配的实验原理。根据约束条件的差异，立体匹配算法一般可以归为两类：局部立体匹配(local stereo matching)和全局立体匹配(global stereo matching)。全局匹配算法是通过求解能量函数的最优解获得高质量的视差图，但是需要复杂的算法，实时性比较差。局部匹配算法相对于全局算法，局部算法的结构简单，且容易实现，算法实时性较好。局部立体匹配算法是对输入的两幅图像中以匹配点为中心，创建相同大小的匹配窗口，用窗口中邻域像素点的信息代表当前的匹配点，然后在左右图像对中寻找最相似的匹配点。本次实验主要实现了传统的局部匹配算法。

传统算法将立体匹配分为四个步骤：匹配代价计算，代价聚合，视差计算和视差优化。下面对这四个步骤分别做详细介绍。

在详细介绍这四个步骤之前，首先要介绍立体视觉的几何原理。

2.1 立体视觉几何原理

一个多视图成像的特殊例子是主体视觉，即使用两台只有水平偏移的照相机

观测同一场景。当照相机的位置如上设置，两幅图像具有相同的图像平面，图像的行是垂直对齐的，那么称图像对是经过矫正的。假设两幅图像经过了矫正，那么对应点的寻找限制在图像的同一行上。一旦找到对应点，由于深度是和偏移成正比的，那么深度(Z 坐标)可以直接由水平偏移来计算。如下图 2.1 所示。

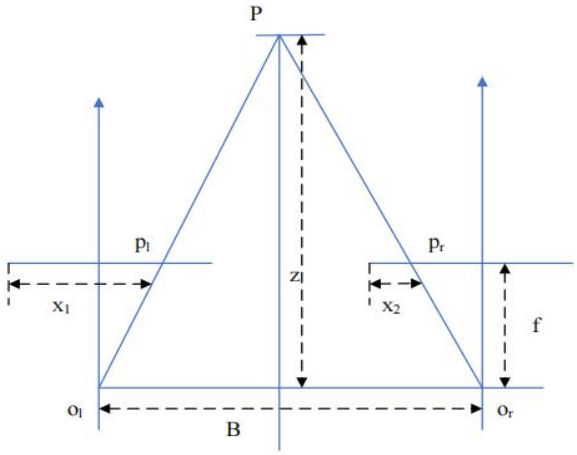


图 2.1 视差原理

图中两摄像机的位置是平行的，左右摄像机的光心分别为o_l和o_r，P 点在左右摄像机的投影成像点分别是点P_l和P_r，成像点距离成像平面左侧边缘的长度即为成像点的横坐标，P_l和P_r横坐标分别为x_1和x_2。根据前述的约束条件，若已知空间中一点在其中一副图片中的位置，就可以沿着对应的极线寻找到该点在另一副图片中的对应位置。在理想情况下，两摄像机光轴是严格平行的，此时两摄像机的水平扫描线位于同一个二维平面上，空间点 P 的视差为$x_1 - x_2$，表示 P 点在左右图像对中的水平位移。

由三角形相似原理可以得到以下几何关系：

$$\frac{z-f}{z} = \frac{B-(x_1-x_2)}{B}$$

化简后可得：

$$z = \frac{Bf}{x_1 - x_2}$$

上式f中代表是摄像机的焦距，z是 P 点的深度值，B 是基线$o_l o_r$长度。$x_1 - x_2$是 P 点视差，常用字母d表示。则上式可表示为：

$$z = \frac{Bf}{d}$$

由上式可知，视差与深度成反比。已知视差d和摄像机的参数时就可以求出空间物体的深度信息，进而可以实现空间物体的三维重建。

2.2 匹配代价计算

匹配代价衡量的是视差范围内左右两幅图像对的对应像素点的相似程度，对应的两点的相似程度越大，匹配代价值越小。如下图 2.2 所示是代价计算的模型，左图为参考图像，右图为目标图像，对于左图中的一个像素点，在视差搜索范围内需要计算该点与右图中全部视差值对应像素点的匹配代价。

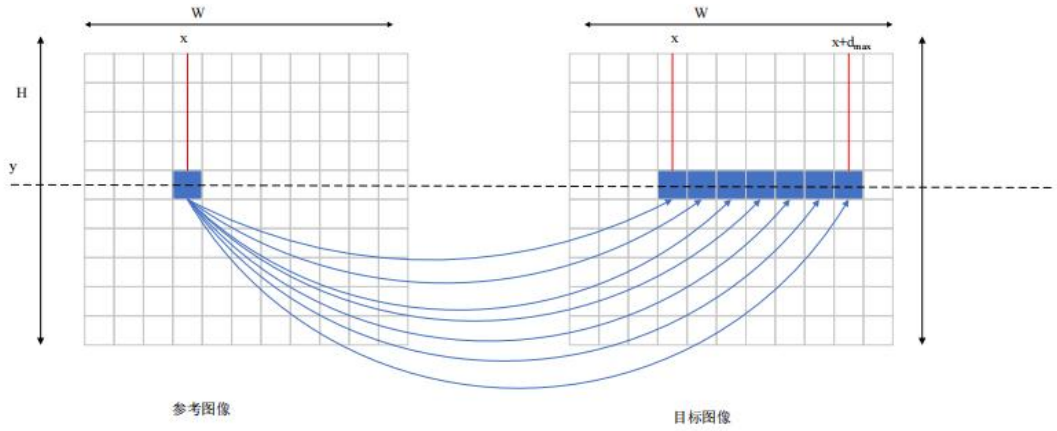


图 2.2 代价计算模型

常用的公式主要有灰度差绝对值之和 (SAD)和灰度差平方和 (SSD)，两种方法主要是通过获取灰度信息关系来描述对应点间的相似性的，其计算公式分别为：

$$SAD(x,y,d)=\sum_{(i,j)\in m}|I_L(x+i,y+j)-I_R(x+i+d,y+j)|$$

$$SSD(x,y,d)=\sum_{(i,j)\in m}[I_L(x+i,y+j)-I_R(x+i+d,y+j)]^2$$

其中，I_L和I_R分别代表左图像和右图像，d表示的是视差，i和j表示像素偏移量，m是支撑窗口大小。灰度差绝对值之和(SAD)和灰度差平方和(SSD)这两种度量函数计算求解的匹配代价值越小说明待匹配的像素点之间越相似。

另外常用的匹配代价还有归一化互相关(NCC)，本次实验也主要使用了这种算法。下面做详细介绍。

归一化互相关（NCC）：

对于原始图像内任意一个像素点构建一个的邻域作为匹配窗口。然后对于目标像素位置同样构建一个大小的匹配窗口，对两个窗口进行相似度度量。其中，d 有一个取值范围。对于两幅图像来说，在进行 NCC 计算之前要对图像进行处理，也就是将两帧图像矫正到水平位置，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。如下是 NCC 计算公式：

$$NCC(x,y,d)=\frac{\sum_{(i,j)\in m}I_L(x+i,y+j)I_R(x+i+d,y+j)}{\sqrt{\sum_{(i,j)\in m}I_L(x+i,y+j)^2\sum_{(i,j)\in m}I_R(x+i,y+j)^2}}$$

其中，I_L和I_R分别代表左图像和右图像，d 表示的是视差，i 和 j 表示像素偏移量，m 是支撑窗口大小。且 NCC 的取值范围在[−1,1]之间。若 NCC=−1，则表示两个匹配窗口完全不相关，相反，若 NCC=1 时，表示两个匹配窗口相关程度非常高。

2.3 代价聚合

通常全局算法不需要代价聚合，而局部算法需要通过求和、求均值或其他方法对一个支持窗口内的匹配代价进行聚合而得到参考图像 2.1 上一点 p 在视差 d 处的累积代价，这一过程称为代价聚合。处理过程其实就是一个滤波的过程，对每一幅代价图进行聚合。第一步代价计算只是得到了图像上所有孤立像素的视差值，但是这些时差值都是孤立的，引入了过多噪声，比如一片区域的视差值都是 10，可是引入噪声后就会导致这一片的视差值都不一样，那么就需要一个滤波的过程，也就是局部立体匹配方法，即采用窗口卷积达到局部滤波的目的。通过匹配代价聚合，可以降低异常点的影响，提高信噪比进而提高匹配精度。代价聚合策略通常是局部匹配算法的核心，策略的好坏直接关系到最终视差图（Disparity maps）的质量。匹配代价聚合的示意图如下图 2. 3 所示。

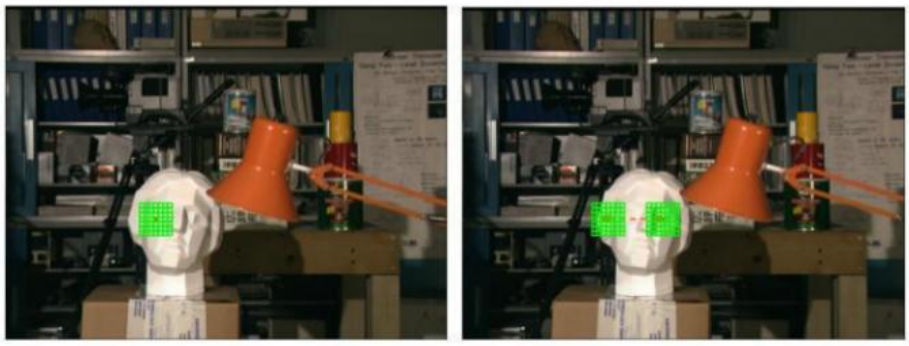


图 2.3 代价聚合示意图

2.4 视差计算

局部立体匹配算法的思想，在支持窗口内聚合完匹配代价后，获取视差的过程通常采用“胜者为王”策略（WTA，Winner Take All），即在视差搜索范围内选择累积代价最优的点作为对应匹配点，与之对应的视差即为所求的视差。视差计算的公式可以表示为：

$$d(x,y) = argmin_{d\in D} C(x,y,d)$$

上式中$C(x,y,d)$是像素点的代价聚合值，D 为视差搜索范围，$d(x,y)$是获取的像素点(x,y)的最终视差值。WTA 视差计算的示意图如下图 2.4 所示。

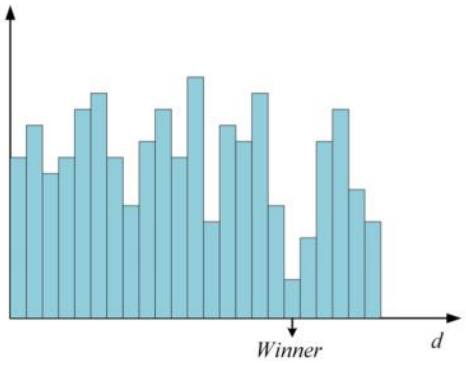


图 2.4 WTA 视差计算示意图

其中，横坐标表示视差值，纵坐标是聚合代价值，胜者为王策略(winner takes all, WTA)是采用灰度差值作为相似性测度，其最优匹配点就是匹配代价值最小时对应的匹配点，最佳匹配点的视差如图 2.4 中箭头部分所示。

2.5 视差优化

由视差计算获得的初始视差图中难免会存在异常值，比如遮挡点、误匹配点或者噪声等，需要对视差图进行后处理修正，最终获得高质量的视差图。

视差优化的目的：

（1）提高匹配精度。视差计算过程是选定最小的代价值对应的视差值，这里的最优视差是一个整数值（整数值才有离散化的视差空间），整数级的像素精度在实际应用中并不能满足需求，所以视差需要优化到亚像素精度级。

（2）剔除错误视差值。错误的视差是指在原本图像中的像素点 A 与 B 是同名点，而得到的匹配结果却是像素点 A 与 C 是同名点。错误匹配的原因有很多，比如遮挡和弱纹理等，错误匹配总是存在的。

（3）优化弱纹理区域。最极端的情况下，比如一面白墙，纹理特征非常不明显，就很难寻找对应的匹配点。弱纹理区域是立体匹配面临的难点问题。

（4）填补视差空洞。错误视差值被剔除后，被剔除的像素就会形成空洞，如何填充无效值空洞使视差图更完整也是视差修正的主要内容。

常用的视差修正方法有左右一致性检测(LRC)、背景填充、自适应中值滤波和亚像素增强等。

如下图 2.5 所示是局部立体匹配算法的示意图，左边是参考图像I_L，右边是目标图像I_R。局部立体匹配算法匹配依据是支撑窗口内的邻域像素点，设 p(x, y)是参考图像中的一个待匹配点，选择一个以 p 为中心的的支撑窗口（例如 3*3），利用窗口内的像素点表示 p，以减少异常点导致的误匹配。左右图像经过极线校正之后没有竖直方向的视差，即参考图像的待匹配点和目标图像中的像素点只有水平方向存在视差。根据极线约束和视差范围约束，待匹配点 p 在目标图像中的视差搜索范围为$[x, x + d]$，在视差搜索范围内，在目标图像中构建与参考图像相同尺寸的 3*3 窗口，滑动窗口的过程中比较每个窗口与参考图像窗口的灰度的相似性，在极线方向上与待匹配点最相似的窗口的中心匹配点即为最优的匹配点。

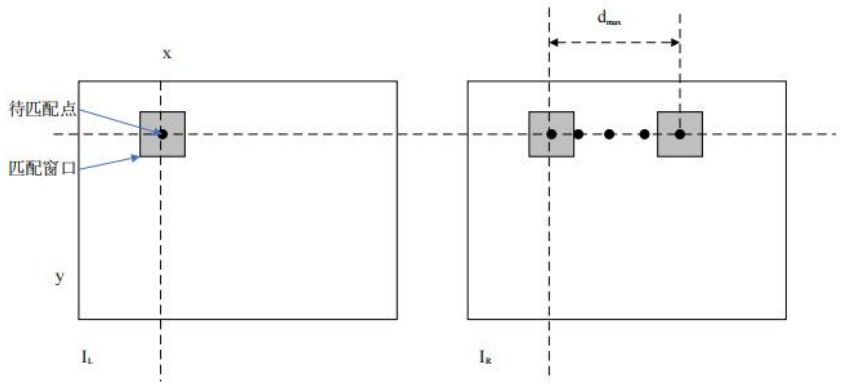
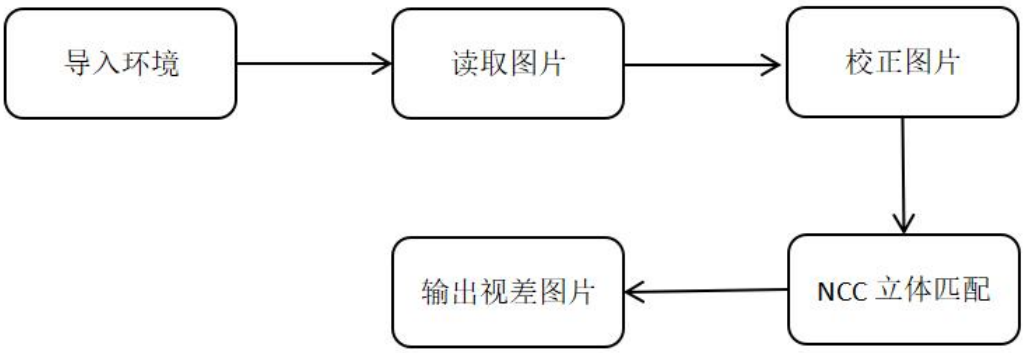


图 2.5 局部匹配算法

三、实验步骤



四、代码实现

导入环境，读取图片并校正。

```
from pylab import *
from PIL import Image
import numpy as np
from scipy import ndimage

"""载入图像，并使用该函数计算偏移图"""
im_l = np.array(Image.open('view0.png').convert('L'), 'f')
im_r = np.array(Image.open('view1.png').convert('L'), 'f')
gray()
```

NCC 立体匹配。

```python
def NCC(im_l, im_r, start, steps, wid):
    """ 使用归一化的互相关计算视差图像 该函数返回每个像素的最佳视差"""
    m, n = im_l.shape
    # 保存不同求和值的数组
    mean_l = np.zeros((m, n))
    mean_r = np.zeros((m, n))
    s = np.zeros((m, n))
    s_l = np.zeros((m, n))
    s_r = np.zeros((m, n))
    dmaps = np.zeros((m, n, steps)) # 保存深度平面的数组
    ndimage.filters.uniform_filter(im_l, wid, mean_l) # 计算图像块的平均值
    ndimage.filters.uniform_filter(im_r, wid, mean_r) # 计算图像块的平均值
    norm_l = im_l - mean_l # 归一化图像
    norm_r = im_r - mean_r # 归一化图像
    # 尝试不同的视差
    for displ in range(steps):
        # 将左边图像移动到右边，计算加和
        ndimage.filters.uniform_filter(np.roll(norm_l, -displ - start) *
                                       norm_r, wid, s)  # 和归一化
        ndimage.filters.uniform_filter(np.roll(norm_l, -displ - start) *
                                       np.roll(norm_l, -displ - start), wid, s_l)
        ndimage.filters.uniform_filter(norm_r * norm_r, wid, s_r)  # 和反归一化
        # 保存 ncc 的分数
        dmaps[:, :, displ] = s / np.sqrt(s_l * s_r)
    # 为每个像素选取最佳深度
    return np.argmax(dmaps, axis=2)
```

输出不同窗口宽度的视差图。

```
# 开始偏移，并设置步长
steps = 12
start = 4
wid = np.array([1,3,5,7,9,11])
res=[]
for i in range(0, 6):
    res.append(NCC(im_l, im_r, start, steps, wid[i]))
a = np.array([231,232,233,234,235,236])
name = ['ncc wide=1', 'ncc wide=3', 'ncc wide=5',
        'ncc wide=7','ncc wide=9','ncc wide=11']
for i in range(0, 6):
    subplot(a[i])
    imshow(res[i])
    title(name[i])
    axis('off')
show()
```

五、实验结果与分析

测试图片如下图 5.1 所示。其中，左侧模拟左眼视角，右侧模拟右眼视角。

图 5.1 测试图片

本次实验对比了不同窗口宽度的 NCC 视差匹配图。如下图 5.2 所示分别是窗口宽度为 1，3，5，7，9 和 11 的 6 张视差图。

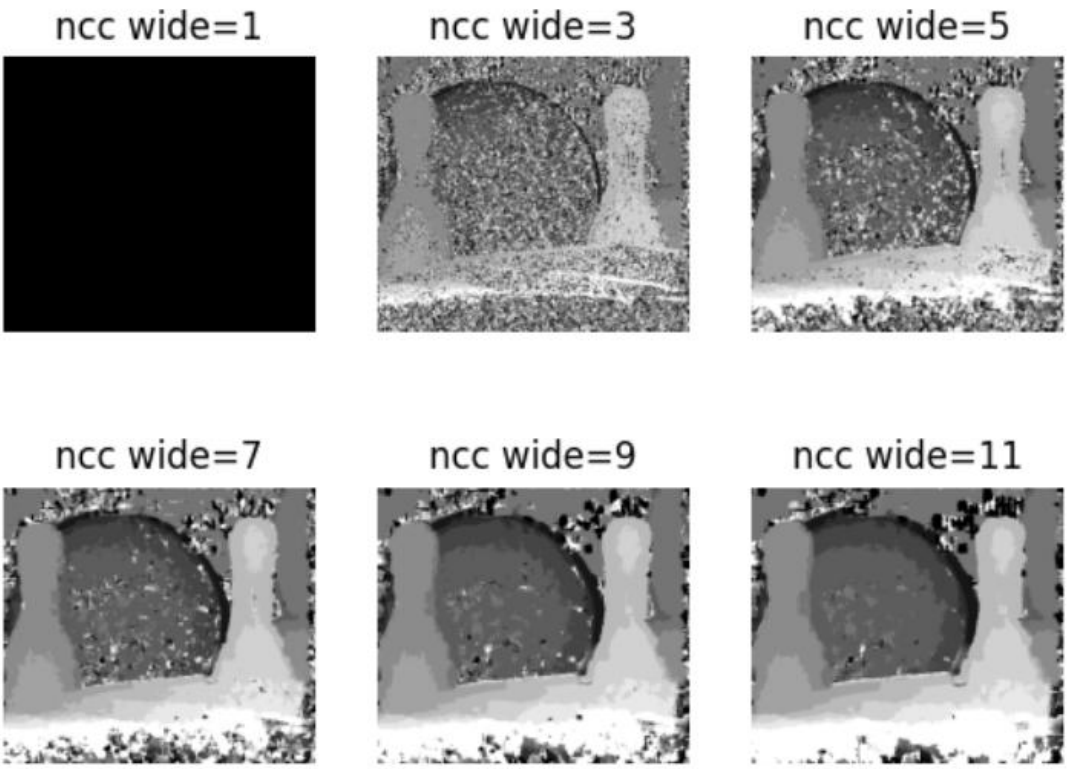


图 5.2 不同窗口宽度的视差图

首先对单张 NCC 视差匹配的结果做分析，比如观察当 wide=9 时的结果，如下图 5.3 所示。

可以看到离相机越近的物体亮度越高，比如靠近相机的被子，它们整体呈现的更亮，而后面的背景则更暗。这是因为前景的位移比背景的位移更多，越靠近摄像机的目标，它在左右视图位移的距离会更多，这是由近大远小导致。结合原理来分析，根据下述公式：

$$z = \frac{Bf}{d}$$

显然 z 和 d 呈反比关系，因此视差越大，z 越小，即目标越近，视差越大，深度越小。

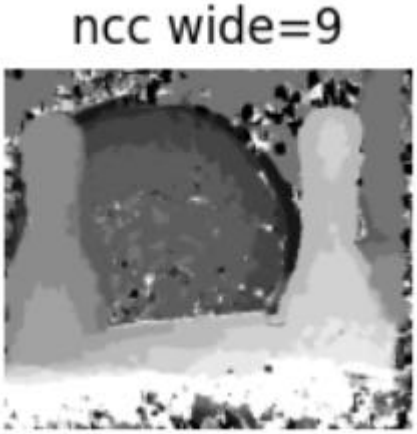


图 5.3 窗口宽度为 9 的视差图

接下来观察不同窗口值的运行结果，初步观察可得知：随着 wide 值的增大，

图像深度信息的图像的轮廓逐渐明显，可以看到离照相机最近的目标是被子，随着 wide 增大，被子的轮廓逐渐清晰，但较远处的物体，比如墙壁，则越来越难以辨认。

进一步观察结果，这次拿出 wide=3 与 wide=11 的结果作对比，如下图 5.4。

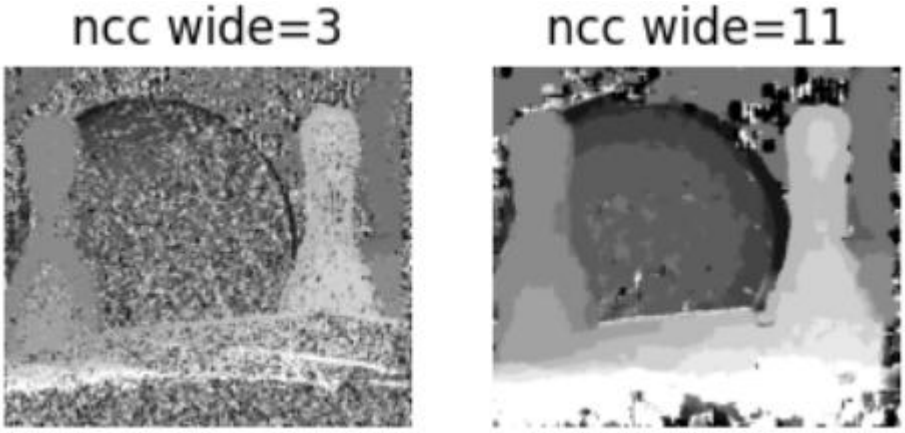


图 5.4 窗口宽度为 3 和 11 的视差图

可以看到 wide 越小，能得到更多的细节，比如见 wide=3 时的结果图，虽然图像整体比较模糊，但是可以看到我们能够得到更多保龄球的细节信息，从原理分析，当 wide 值较小，进行滤波时考虑的因素少，只受某像素点自身影响以及周围少量像素点的影响，所以保留下来更多的自身特征。但同时噪声也大，见 wide=3 时的背景墙，上面出现了许多噪声点。

而当窗口值增大时，鲁棒性增强，噪声明显减少，但我们可以发现图像的细节也在逐渐减少，可物体的轮廓逐渐清晰，尤其是近景目标，如 wide=11 的视差图，我们能够更清晰地看出保龄球的轮廓，但是由于细节减少，远处的物体就比较不易判断。在 wide 值较大情况下，NCC 匹配下充分考虑了周围像素点的影响，能更好的观察出视差。

六、实验总结

本次实验学习并代码实现了将两张图片进行视差匹配，并输出视差图的过程。匹配计算代价主要依靠归一化互相关（NCC），并且比较了 NCC 立体匹配算法在不同窗口宽度下的视差图，发现较大的窗口宽度可以提高算法的匹配精度和鲁棒性，同时也会增加计算量和耗时。因此，在实际应用中需要根据具体情况进行窗口宽度的选择和权衡。

未来可以进一步研究如何通过对立体匹配算法的参数进行优化，来提高算法的匹配精度和效率。例如可以尝试使用深度学习等新技术来改进立体匹配算法，或者结合其他传感器和信息源来提高匹配的准确性和鲁棒性。此外，还可以将立体匹配算法应用于更广泛的领域，例如自动驾驶、机器人视觉等，以满足不同领域对于深度信息的需求。