计算机视觉实践实验报告

目录

一. 实验目的

- 了解图像视觉匹配。
- 通过立体匹配（Stereo Matching）得到两张图像的视差图。

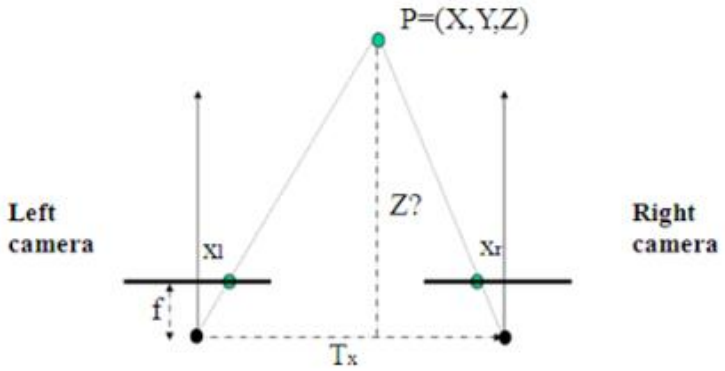
二. 实验原理

立体匹配主要是通过找出每对图像间的对应关系，根据三角测量原理，得到视差图；在获得了视差信息后，根据投影模型很容易地可以得到原始图像的深度信息和三维信息。立体视觉匹配在立体视觉研究中是比较核心的问题。立体视觉应用：车导航，3D场景重建等。以下主要介绍窗口匹配方法及NCC视差匹配。

2.1 几何原理

一个多视图成像的特殊例子是主体视觉(或者立体成像)，即使用两台只有水平(向一侧)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。



通过几何知识运算，得到：

$$\frac{T + x_l - x_r}{T} = \frac{Z - f}{Z}$$

进一步化简，可得：

$$Z = f\frac{T}{x_r - x_l}$$

其中，f是经过矫正图像的焦距，T是两个照相机中心之间的距离，x_l和x_r是左右两幅图像中对应点的x坐标。分开照相机中心的距离称为基线。

2.2 立体匹配的步骤

匹配代价计算:计算匹配代价，即计算参考图像上每个像素点IR(P)，以所有视差可能性去匹配目标图像上对应点IT(pd)的代价值，因此计算得到的代价值可以存储在一个h*w*d*(MAX)的三维数组中，通常称这个三维数组为视差空间图。匹配代价时立体匹配的基础，设计抗噪声干扰、对光照变化不敏感的匹配代价，能提高立体匹配的精度。因此，匹配代价的设计在全局算法和局部算法中都是研究的重点。

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上一点p在视差d处的累积代价CA(p,d)，这一过程称为代价聚合。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(Disparity maps)的质量。

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。即P点的视差为

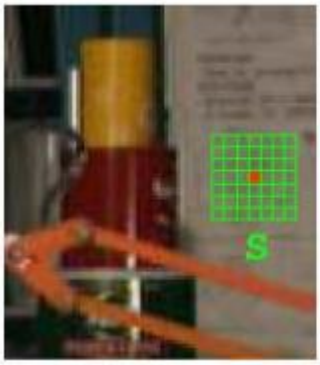
$$d = \mathrm{argmin}CA(p, d)$$

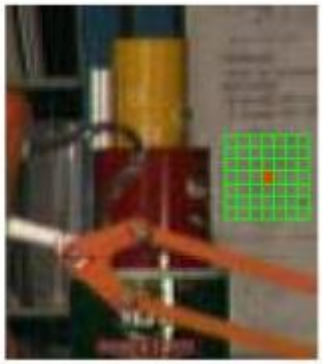
但是这种方法可能导致噪声大，因为如果直接用点的像素值坐标匹配，风险太大，找到的匹配点不一定是正确的，这时可以采用窗口匹配方法，关于这种方法在下面会做详细介绍。

后处理:一般的，分别以左右两图为参考图像，完成上述三个步骤后可以得到左右两幅视差图像。但所得的视差图还存在一些问题，如遮挡点视差不准确、噪声点、误匹配点等存在，因此还需要对视差图进行优化，采用进一步执行后处理步骤对视差图进行修正。常用的方法有插值（Interpolation）、亚像素增强（Subpixel Enhancement）、精细化（Refinement）、图像滤波（Image Filtering）等操作。

2.3 窗口匹配方法

它的主要思路是：以当前点为中心，切出一个图像块，每个图像块依次做对比，找到最接近的图像块。这时就不只是拿单个点做对比了，可以有效减小误差。举例见下图：





2.4 匹配代价

相似性测度函数用于度量参考图像中的匹配基元和目标图像中的匹配基元的相似性，即判断参考图像和目标图像中两点为对应匹配点的可能性，也称为匹配代价，这里记为C(x,y,d)，表示像素点(x,y)在视差为d情况下的匹配误差。下面介绍最常见的三种匹配代价：

- 绝对差值和（Sum of Absolute Differences,SAD）

$$C(x,y,d)=\sum_{x\varepsilon S}|I_R(x,y)-I_T(x+d,y)|$$

- 差值平方和（Sum of squared Differences,SSD）

$$C(x,y,d)=\sum_{x\varepsilon S}(I_R(x,y)-I_T(x+d,y))^2$$

- 截断绝对差值和（Sum of Truncated Absolute Differences，STAD）

$$C(x,y,d)=\sum_{x\varepsilon S}\min\{|I_R(x,y)-I_T(x+d,y)|,T\}$$

- 另外常用的匹配代价还有归一化互相关NCC（Normalized Cross Correlation），在下面做详细介绍。

2.5 归一化互相关（NCC）

对于原始的图像内任意一个像素点构建一个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。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：

$$NCC(p,d)=\frac{\sum_{(x,y)\in W_p}(I_1(x,y)-\overline{I1}(p_x,p_y))\bullet(I_2(x+d,y)-\overline{I}_2(p_x+d,p_y))}{\sqrt{\sum_{(x,y)\in W_p}(I_1(x,y)-\overline{I}_1(p_x,p_y))^2\bullet\sum_{(x,y)\in W_p}(I_2(x+d,y)-\overline{I}_2(p_x+d,p_y))^2}}$$

其中NCC(p,d)得到的值的范围将在[-1,1]之间。

W_p为之前提到的匹配窗口。

$I_1(x,y)$为原始图像的像素值。

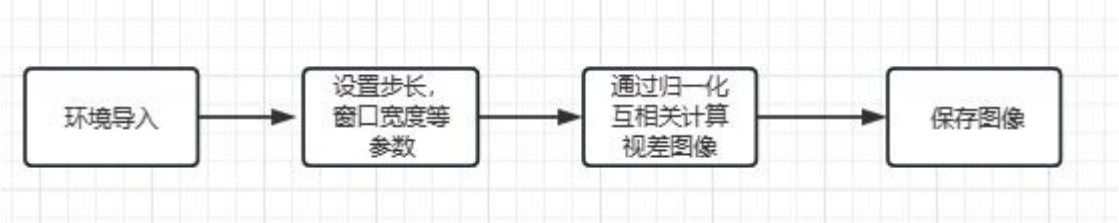
$I_2(x+d,y)$为原始图像在目标图像上对应点位置在x方向上偏移d后的像素值。

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。

匹配流程：

- 采集图像：通过标定好的双目相机采集图像，当然也可以用两个单目相机来组合成双目相机。
- 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。通过校正极线可以方便后续的NCC操作。
- 特征匹配：这里便是我们利用NCC做匹配的步骤啦，匹配方法如上所述，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，最终我们可以得到一个与原始图像尺寸相同的视差图D。

三. 实验步骤



四. 数据集

2003 Stereo datasets:

第一组：

第二组：

五. 程序代码

- 归一化互相关计算视差图像：

```
def plane_sweep_ncc(im_l, im_r, start, steps, wid):
    """ 使用归一化的互相关计算视差图像 """
    m, n = im_l.shape
    # 保存不同求和值的数组
    mean_l = zeros((m, n))
    mean_r = zeros((m, n))
    s = zeros((m, n))
    s_l = zeros((m, n))
    s_r = zeros((m, n))
    # 保存深度平面的数组
    dmaps = zeros((m, n, steps))
    # 计算图像块的平均值
    filters.uniform_filter(im_l, wid, mean_l)
    filters.uniform_filter(im_r, wid, mean_r)
    # 归一化图像
    norm_l = im_l - mean_l
    norm_r = im_r - mean_r
    # 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
        filters.uniform_filter(np.roll(norm_l, -displ - start) * norm_r, wid, s)  # 和归一化
        filters.uniform_filter(np.roll(norm_l, -displ - start) * np.roll(norm_l, -displ - start), wid, s_l)
        filters.uniform_filter(norm_r * norm_r, wid, s_r)  # 和反归一化
        # 保存 ncc 的分数
        dmaps[:, :, displ] = s / sqrt(s_l * s_r)
        # 为每个像素选取最佳深度
    return np.argmax(dmaps, axis=2)
```

- 使用带有高斯加权周边的归一化互相关计算视差图像。

```python
def plane_sweep_gauss(im_l, im_r, start, steps, wid):
    """ 使用带有高斯加权周边的归一化互相关计算视差图像 """
    m, n = im_l.shape
    # 保存不同加和的数组
    mean_l = zeros((m, n))
    mean_r = zeros((m, n))
    s = zeros((m, n))
    s_l = zeros((m, n))
    s_r = zeros((m, n))
    # 保存深度平面的数组
    dmaps = zeros((m, n, steps))
    # 计算平均值
    filters.gaussian_filter(im_l, wid, 0, mean_l)
    filters.gaussian_filter(im_r, wid, 0, mean_r)
    # 归一化图像
    norm_l = im_l - mean_l
    norm_r = im_r - mean_r
    # 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
        filters.gaussian_filter(np.roll(norm_l, -displ - start) * norm_r, wid, 0, s)  # 和归一化
        filters.gaussian_filter(np.roll(norm_l, -displ - start) * np.roll(norm_l, -displ - start), wid, 0, s_l)
        filters.gaussian_filter(norm_r * norm_r, wid, 0, 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)
```

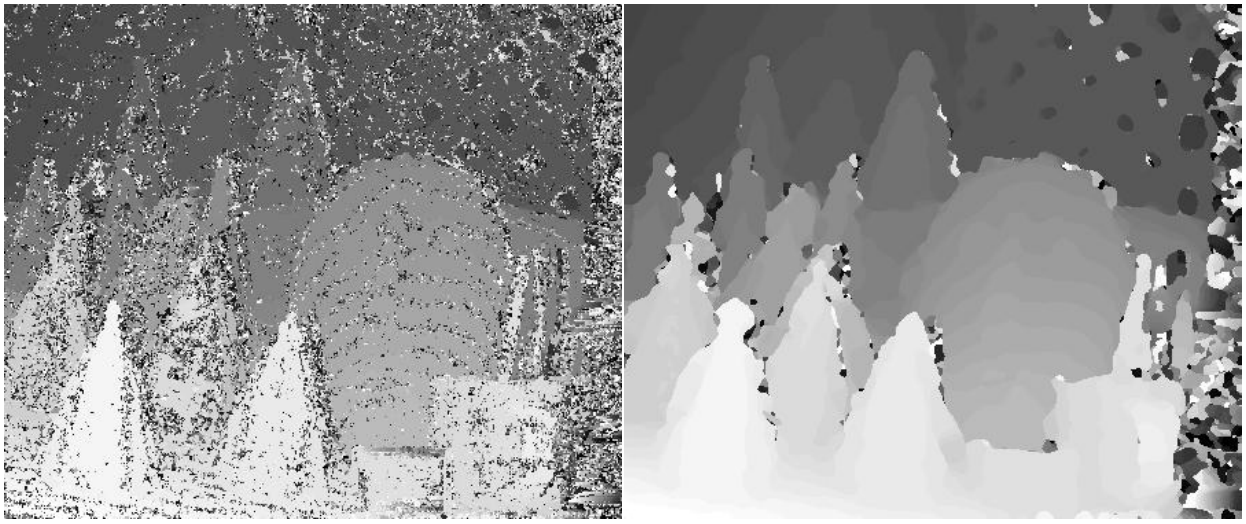
- 载入图像，转为array，设置步长，窗口宽度等参数，调用函数得到视差图。

```python
im_l = array(Image.open(r'im2.png').convert('L'), 'f')
im_r = array(Image.open(r'im6.png').convert('L'), 'f')
# 开始偏移，并设置步长
steps = 50
start = 4
# ncc 的宽度
wid = 9
res = plane_sweep_ncc(im_l, im_r, start, steps, wid)
```

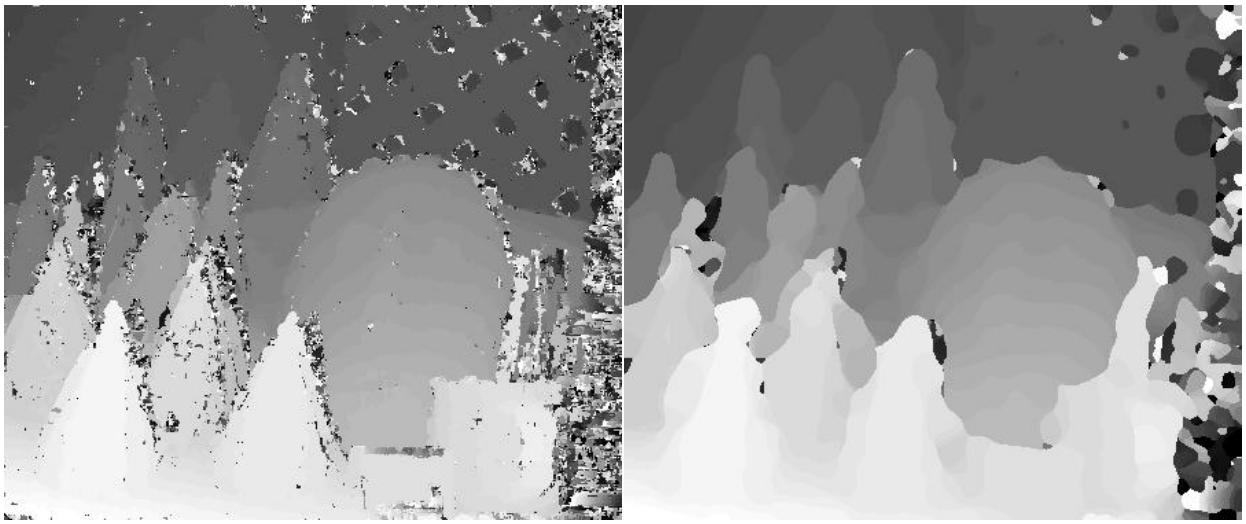
六. 实验结果

- 以下均在步长为50情况下的实验结果

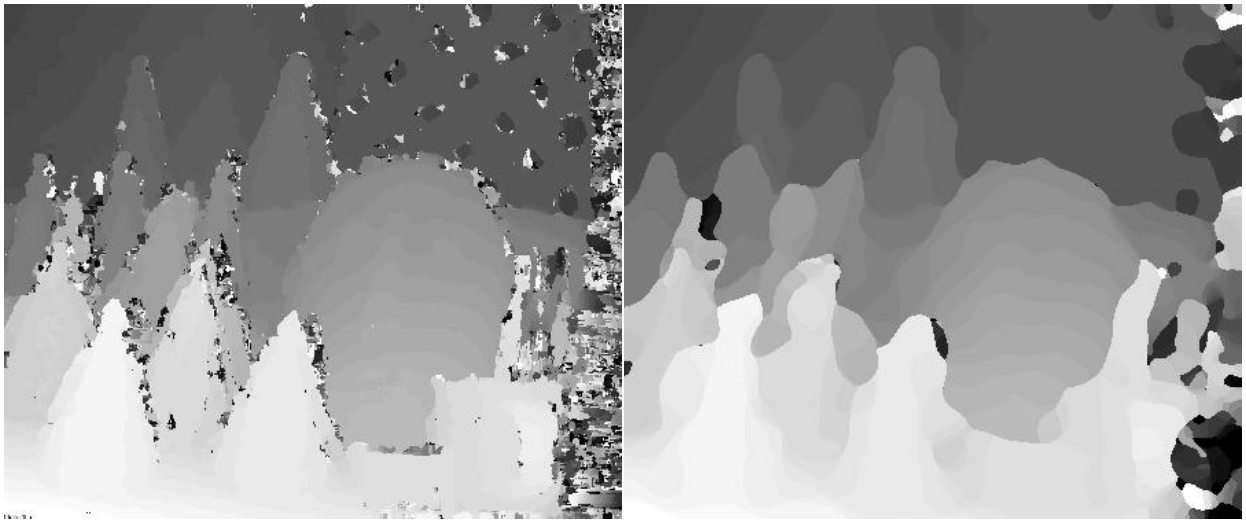
Wid=3（普通滤波器，高斯滤波器）：



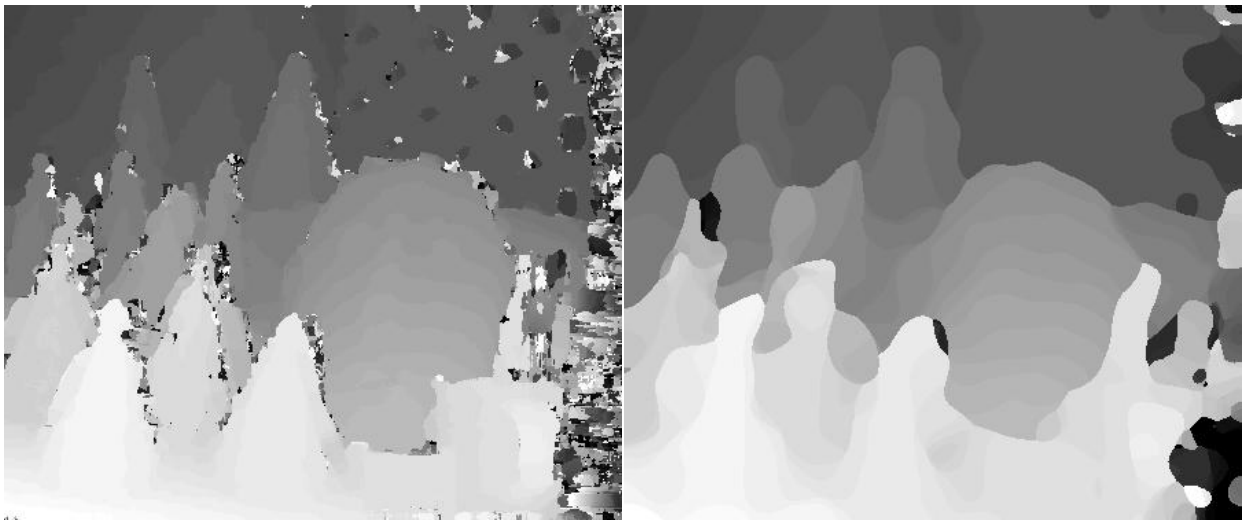
Wid=5（普通滤波器，高斯滤波器）：



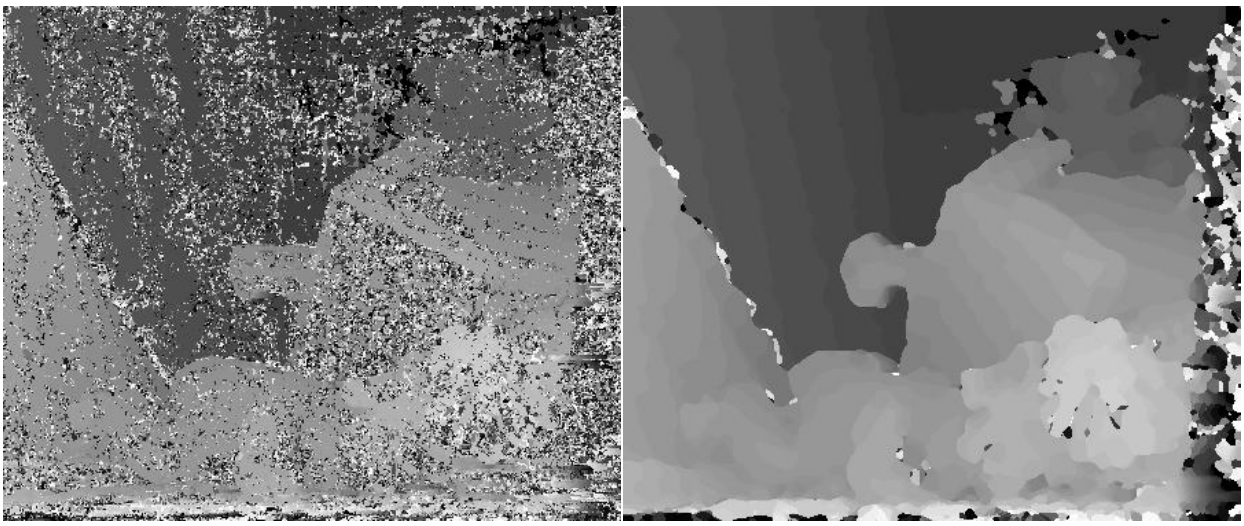
Wid=7（普通滤波器，高斯滤波器）：



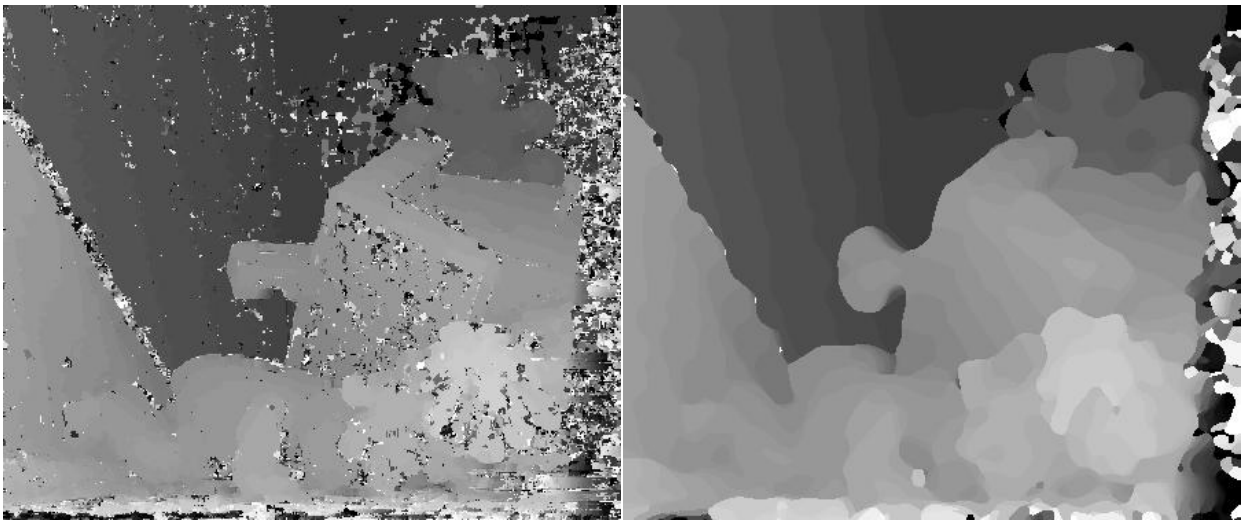
Wid=9（普通滤波器，高斯滤波器）：



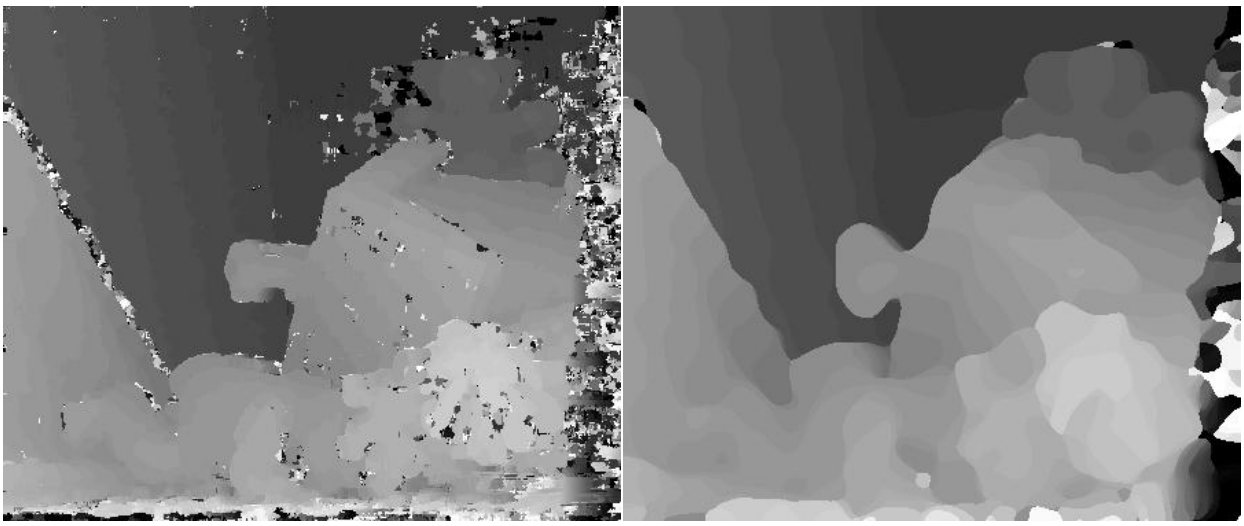
Wid=3（普通滤波器，高斯滤波器）：



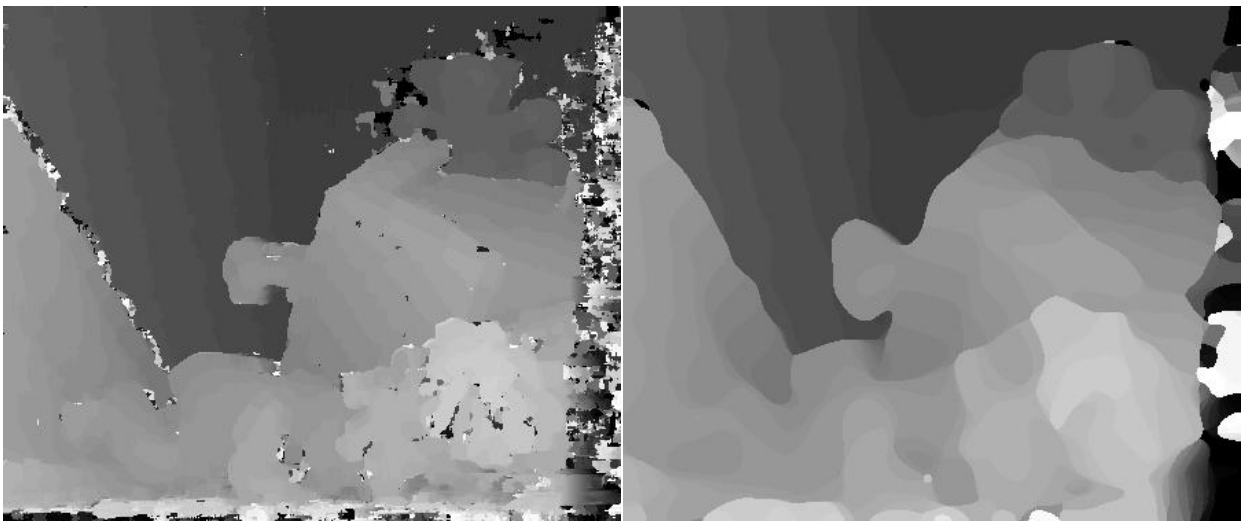
Wid=5（普通滤波器，高斯滤波器）：



Wid=7（普通滤波器，高斯滤波器）：



Wid=9（普通滤波器，高斯滤波器）：



七. 实验分析与总结

- 随着窗口值的增大，细节和噪声都逐渐减少。高斯滤波器对比普通的滤波器，减少噪声的作用更加明显。可以看到当窗口值为3 的时候，视差匹配结果比较模糊，存在很多噪点，而当窗口值逐渐增大的时候，图片噪声逐渐减少，但细节也逐渐减少。对比普通滤波器，使用高斯滤波处理过的，细节丢失更加严重，所以高斯版本的视差图可以分析主要景物的层次关系和去除噪声用，但是窗口大小不可以过大。
- 当图片过大，左右平移如果跨度过大，可能会出现视差图看不到景物轮廓，都是类似于马赛克的色块，可能需要调整步长。