第二次大作业

1、用于计算的两张图片

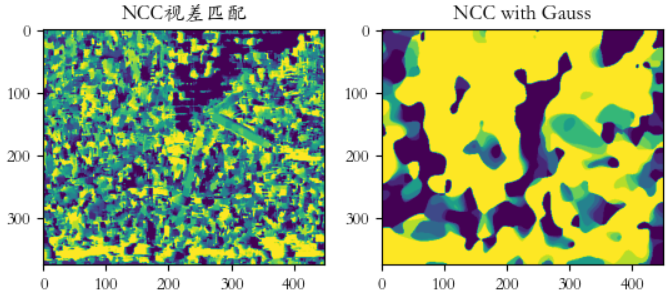
 

2、计算得到的视差图

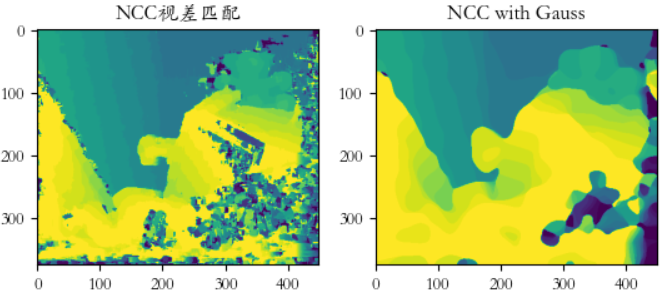
因不同窗口大小、步长、开始偏移对匹配结果均会产生影响，对不同步长、开始偏移以及窗口大小作对比得出的匹配结果进行对比如下：

这里设置开始的超参数分别为：steps=50, start=4, width=9

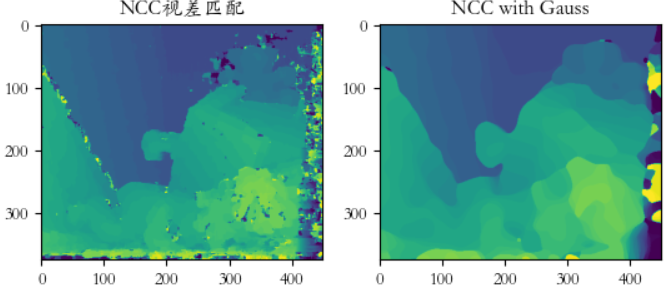
1. 不同步长



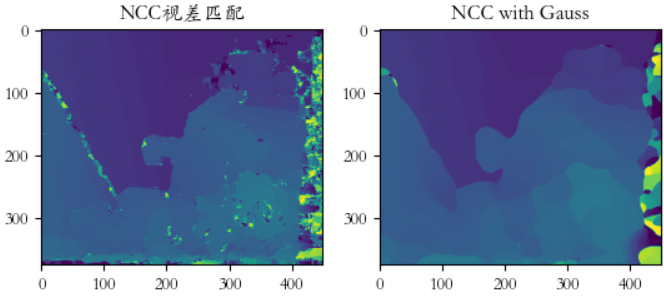
Step=10



Step=30



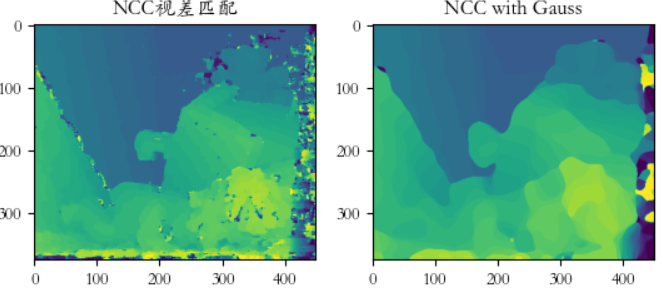
Step=50



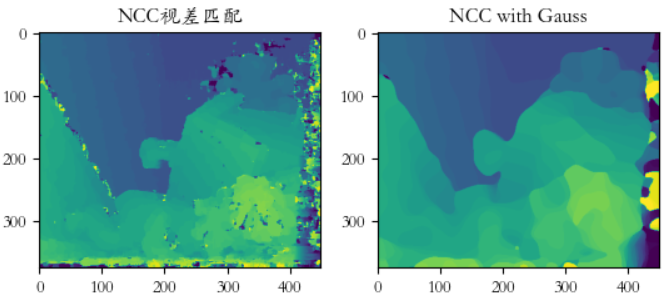
Step=100

对比认为步长在50的时候效果是最好的。

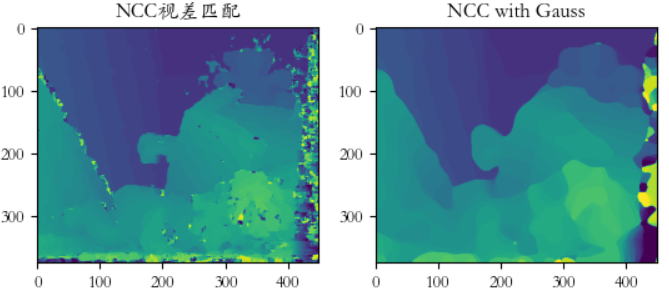
1. 不同开始偏移



start=1



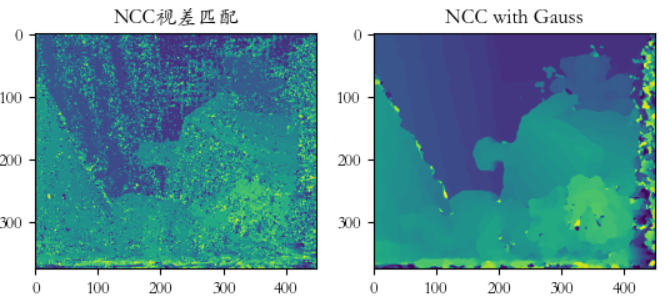
start=4



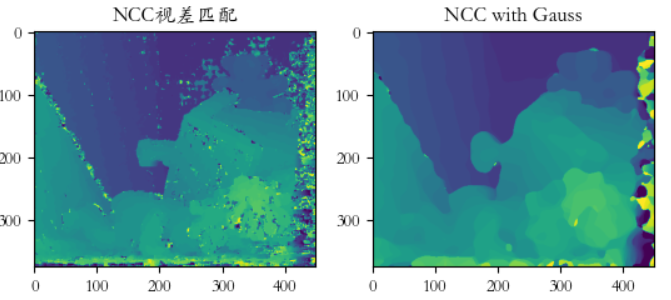
start=8

对比认为start值为4或者8匹配效果都是比较好的。

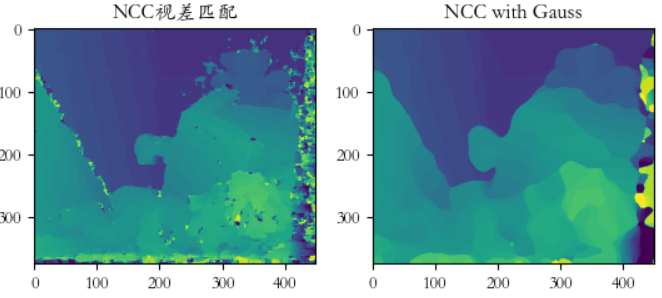
1. 不同窗口宽度



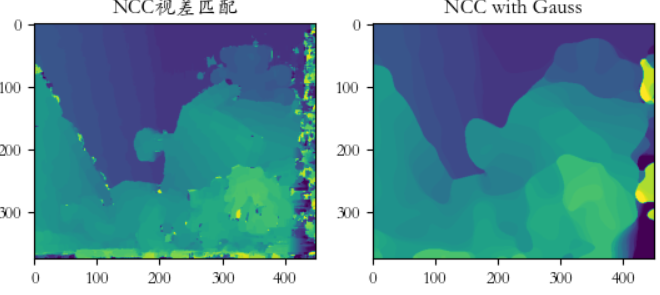
Width=3



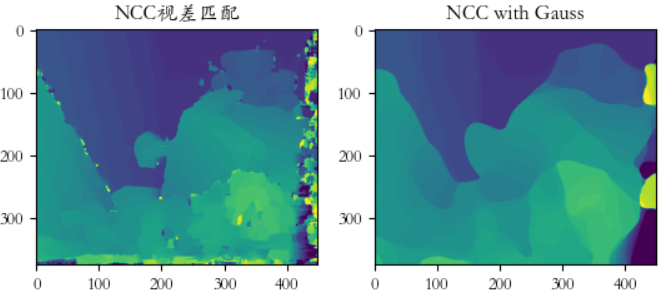
Width=6



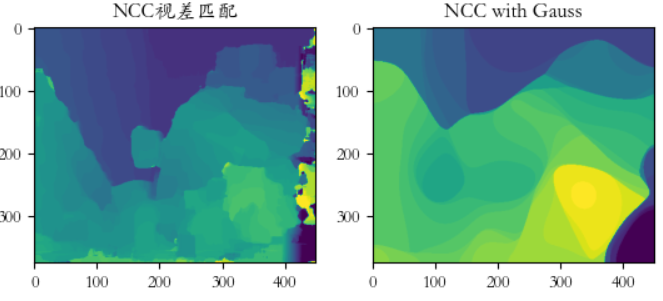
Width=9



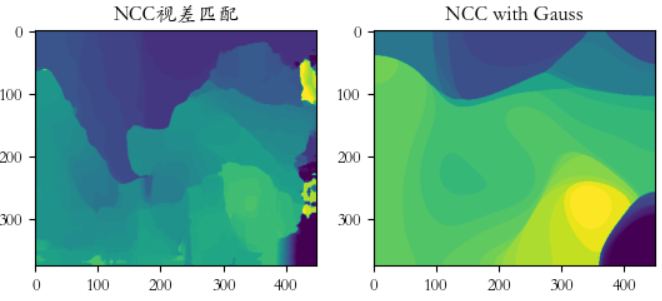
Width=12



Width=15



Width=30



Width=50

对比以上匹配结果当width的值偏小时也就是窗口略小时，会存在比较多的噪声。Width的最优取值应该在6~15之间。当width值很大时图片变得越来越光滑与原图匹配效果越来越差。可以分析得出，当窗口越来越大时与小窗口相似，在步长不变的情况下与此窗口里相似的特征点也会越来越多，导致计算出来的视差没有明显的跨度，最终导致图片的颜色层次越来越少。

3、使用代码（python）

# -\*- coding: utf-8 -\*-  
from PIL import Image  
from numpy import \*  
from numpy.ma import array  
import imageio  
from scipy.ndimage import filters  
import matplotlib.pyplot as plt  
import numpy as np  
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)  
  
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)  
  
im\_l = array(Image.open(r'C:\Users\DELL\Desktop\datasets\2003\teddy-png-2\teddy\im2.png').convert('L'), 'f')  
im\_r = array(Image.open(r'C:\Users\DELL\Desktop\datasets\2003\teddy-png-2\teddy\im6.png').convert('L'),'f')  
# 开始偏移，并设置步长  
steps = 50  
start = 4  
# ncc 的宽度  
wid = 9  
res1 = plane\_sweep\_ncc(im\_l, im\_r, start, steps, wid)  
img\_uint8\_1 = res1.astype(np.uint8)  
imageio.imwrite(r'C:\Users\DELL\Desktop\datasets\2003\teddy-png-2\teddy\shicha1.png', img\_uint8\_1)  
res2 = plane\_sweep\_gauss(im\_l, im\_r, start, steps, wid)  
img\_uint8\_2 = res2.astype(np.uint8)  
imageio.imwrite(r'C:\Users\DELL\Desktop\datasets\2003\teddy-png-2\teddy\shicha2.png', img\_uint8\_2)  
plt.figure()  
ax1 = plt.subplot(1,2,1)  
ax1.set\_title('NCC视差匹配') ## 归一化相关性方法（normalization cross-correlation）  
plt.imshow(img\_uint8\_1)  
ax2 = plt.subplot(1,2,2)  
ax2.set\_title('NCC with Gauss')  
plt.imshow(img\_uint8\_2)  
plt.show()