双目三维重建

3190103683 张嘉浩

1. 实验要求

针对附件图像对（相机平行配置），编写代码实现双目三维重建过程，计算视差图。要求提供代码和结果视差图。

2. 实验步骤

NCC(Normalization Cross-Correlation)，归一化相关性，用于归一化待匹配目标之间的相关程度。对于原始的图像内任意一个像素点(px, py) 构建一个$n \times n$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$n \times n$大小的匹配窗口，对两个窗口进行相似度度量，这里的d有一个取值范围。对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。

$$ncc\left(I_1, I_2\right) = \frac{\sum_x \left(I_1(x) - \mu_1\right)\left(I_2(x) - \mu_2\right)}{\sqrt{\sum_x \left(I_1(x) - \mu_1\right)^2 \sum_x \left(I_2(x) - \mu_2\right)}}$$

- $I_1(x)$为原始图像的像素值
- μ_1为原始窗口内像素的均值
- $I_2(x)$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
- μ_2为目标图像匹配窗口像素均值。

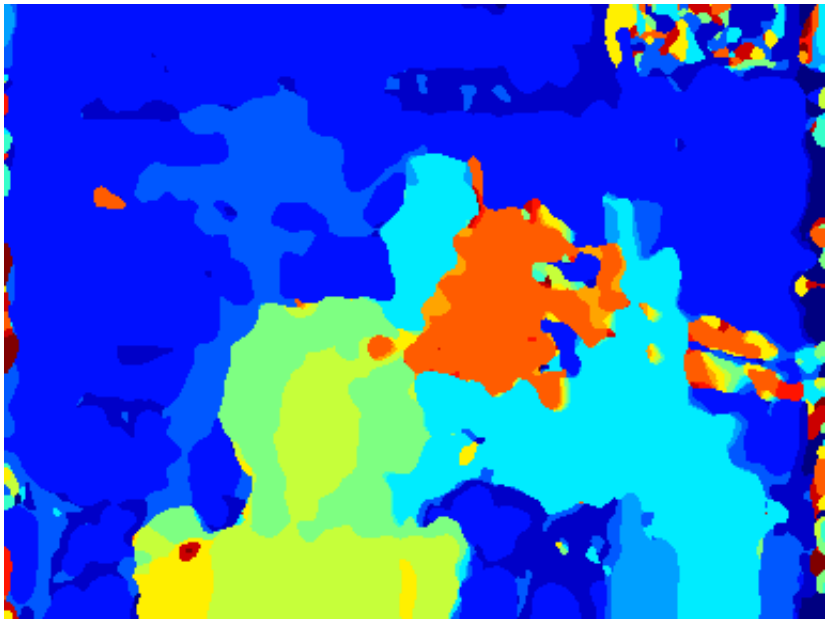
对于左图的每一待匹配点，取其window_size*window_size大小的邻域作为窗口(region A)，在右图的同一水平极线上，在该点坐标的左侧一定范围内使用滑动窗口(region B)计算各点与待匹配点的NCC值，取最大NCC值对应的滑动步长作为视差值。

实现代码如下：

```
# 遍历范围内的所有视差值
for dis in range(steps):
    # 计算NCC
    filters.gaussian_filter(
        np.roll(norm_l, - dis - start) * norm_r, sigma, 0, s)  # 和归一化
    filters.gaussian_filter(
        np.roll(norm_l, - dis - start) * np.roll(norm_l, - dis - start), sigma, 0, s_l)
    filters.gaussian_filter(norm_r * norm_r, sigma, 0, s_r)  # 和反归一化
    depthmaps[:, :, dis] = s / np.sqrt(s_l * s_r)
```

3. 实验结果与讨论

在实验中我调整了多组窗口大小参数，发现窗口越大，图像噪声越少，但是细节丢失越多。最后将窗口大小调整为25，得到如下所示图片：



附录：源代码

`3re.py`

```
import cv2
import numpy as np
from scipy.ndimage import filters

# 读取图像
img_l = cv2.imread("tsukuba_l.png", 0)
img_r = cv2.imread("tsukuba_r.png", 0)

# 设置大概的视差范围[start, start+steps-1]
steps = 15
start = 3

# NCC窗口大小
window_size = 25

# 计算视差图像
m, n = img_l.shape
# 计算等效高斯sigma值
sigma = (window_size-1)/8

# 初始化各数组
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))
```

```
# 保存深度信息的数组
depthmaps = np.zeros((m, n, steps))

# 计算平均值
filters.gaussian_filter(img_l, sigma, 0, mean_l)
filters.gaussian_filter(img_r, sigma, 0, mean_r)

# 归一化图像
norm_l = img_l - mean_l
norm_r = img_r - mean_r

# 遍历范围内的所有视差值
for dis in range(steps):
    # 计算NCC
    filters.gaussian_filter(
        np.roll(norm_l, - dis - start) * norm_r, sigma, 0, s)  # 和归一化
    filters.gaussian_filter(
        np.roll(norm_l, - dis - start) * np.roll(norm_l, - dis - start), sigma, 0, s_l)
    filters.gaussian_filter(norm_r * norm_r, sigma, 0, s_r)  # 和反归一化
    depthmaps[:, :, dis] = s / np.sqrt(s_l * s_r)

# 为每个点选取最佳匹配点并取其视差
res = np.argmax(depthmaps, axis=2) + start

# 绘制伪彩色视差图
res_plot = np.array((res-start)/(steps-1.0)*255.0, dtype=np.uint8)
res_plot = cv2.applyColorMap(res_plot, cv2.COLORMAP_JET)
cv2.imshow("depth", res_plot)
cv2.imwrite("depth_wd"+str(window_size)+".png", res_plot)

cv2.waitKey(0)
```