计算机视觉与应用实践实验报告（五）

目录

计算机视觉与应用实践实验报告（五）.....1
一． 实验内容.....1
二． 实验原理.....1
三． 实验步骤.....5
四． 程序代码.....6
五． 实验结果.....7
六． 实验分析与总结.....8

一．实验内容

图像视差匹配，通过立体匹配计算得到两张图像的视差图。

二．实验原理

立体匹配是指通过参数一致的摄像机拍摄同一场景获得左右 2 幅视图，然后采用数学运算模型计算出左右 2 幅视图中的对应点以及视差，最后求取深度信息的过程。其工作思想是：利用校正后的立体图像计算匹配基元之间的相似性，从而计算视差，得到视差图。可划分为四个步骤：匹配代价计算、匹配代价聚合、视差计算和视差优化。

2.1 匹配代价计算

匹配代价，可以理解为像素之间的差异程度，匹配代价计算的目的是衡量待匹配像素与候选像素之间的相关性。通过匹配代价函数计算两个像素间的匹配代价，代价越小则说明相关性越大，在概率上互为匹配点的可能性也越大，因此匹配代价函数的确定是一切匹配工作的基础。在进行匹配代价计算时，必须选择合适的代价测量函数，并将函数的选取与实际情况联系起来，若选择合适，能准确反映匹配点，精度高且鲁棒性强；若选择不恰当，则易发生误匹配现象。目前常用的成本测量函数大致可分为三类，基于信息差值的代价测量函数，基于相关性的代价测量函数，基于非参数变换的代价测量函数：

1. 基于信息差值的代价度测量函数，以目标像素和待匹配像素的支持窗口

为计算范围，计算两个窗口间相应信息的差值作为目标像素在视差d下的成本。基于信息差值的代价测量函数通常易于实现、实时性好，但对噪声、光照变化等鲁棒性差。

2. 基于相关性的成本测量函数，以目标像素和待匹配像素的支持窗口为计算范围，计算两个窗口间相应信息的相关性作为目标像素在视差d下的代价。这种函数对噪声有一定的鲁棒性。

3. 基于非参数变换的代价测量函数，以目标像素和待匹配像素的支持窗口为计算范围，以窗口内像素点相对于中心像素的灰度大小为标准计算代价，这种函数对噪声和光照具有强鲁棒性。

常用的代价测量函数如图 2.1 所示：

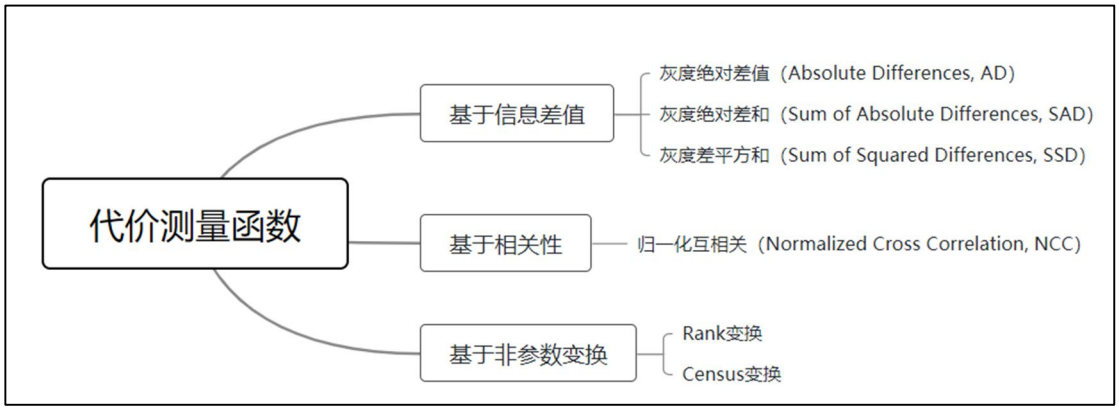

图 2. 1 常用的代价测量函数

匹配代价与相似度有直接关系，一般代价越大，相似程度越小，反之亦反。以参考像素为基准，在目标图像的同一水平极线上计算视差范围内的所有匹配代价，将这些匹配代价储存在大小为$\mathrm{H}\times\mathrm{W}\times\mathrm{D}$的三维矩阵中（其中$H$是图像的长，$W$是图像的宽，$D$为视差范围的最大视差），生成一个三维视差代价空间，简称 DSI（Disparity Space Image），如图 2.2 所示。

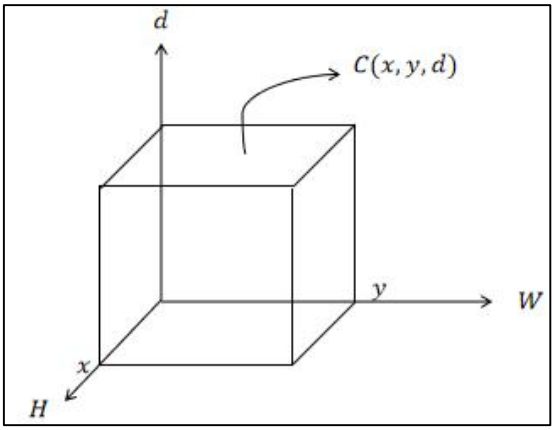

图 2. 2 视差代价空间示意图

定义w_p为以前像素为中心的邻域窗口，$C(x,y,d)$为匹配代价，$I_L(x,y)$和$I_R(x,y)$分别是左图像中参考点和右图像中的待匹配点，d为左图中参考点与右图中当前参与计算匹配代价的待匹配点之间的视差。当前常用的几种相似度测量算法如下所示：

(1) 绝对差值（AD）

$$C_{AD}(x,y,d)=|I_L(x,y)-I_R(x-d,y)| \tag{2.1}$$

(2) 绝对误差和（SAD）

$$C_{SAD}(x,y,d)=\sum_{q\in w_p}|I_L(x,y)-I_R(x-d,y)| \tag{2.2}$$

(3) 平方差和（SSD）

$$C_{SSD}(x,y,d)=\sum_{q\in w_p}\left(I_L(x,y)-I_R(x-d,y)\right)^2 \tag{2.3}$$

(4) 归一化互相关（NCC）

$$C_{NCC}(x,y,d)=\frac{\sum_{(x,y)\in w_p}\left(I_L(x,y)-\bar{I}_L(p_x,p_y)\right)\cdot\left(I_R(x+d,y)-\bar{I}_R(p_x+d,p_y)\right)}{\sqrt{\sum_{(x,y)\in w_p}\left(I_L(x,y)-\bar{I}_L(p_x,p_y)\right)^2\cdot\sum_{(x,y)\in w_p}\left(I_R(x+d,y)-\bar{I}_R(p_x+d,p_y)\right)^2}} \tag{2.4}$$

其中：$\bar{I}_L$和$\bar{I}_R$分别表示左、右窗口的灰度均值。

(5) Census 变换（CT）

$$Census(x,y)=\otimes_{q\in w_p}\xi(I_p,I_q) \tag{2.5}$$

其中：

$$\xi(x,y)=\begin{cases}1, x<y\\ 0, else\end{cases} \tag{2.6}$$

式中⊗表示按位连接，q表示支持窗口内的邻域像素，I_q为邻域像素灰度值。Census 变换的匹配代价计算模型为：

$$C_{Census}(x,y,d)=\sum_{(x,y)\in w_p}Hamming\left(Census_1(x,y)-Census_2(x-d,y)\right) \tag{2.7}$$

这个过程可以视为对深度估计问题空间的初步建模。因为经过矫正之后的立体图片对，参考图像中的某个像素，其对应像素应该在该像素的平行位置。根据一个深度确定两个像素后，即可以用上述的若干种方法计算匹配代价。

2.2 匹配代价聚合

代价聚合的根本目的是让代价值能够准确的反映像素之间的相关性。上一步

匹配代价的计算往往只会考虑局部信息，通过两个像素邻域内一定大小的窗口内的像素信息来计算代价值。但是由于图像噪声、曝光度、镜面反射、弱纹理等客观干扰因素的影响，匹配代价往往不能真实地反映像素间的相似程度，使得匹配结果存在误差。为了减小客观干扰因素的影响，通常会在当前像素的有限邻域窗口内采用相关算法叠加足够多的可靠代价信息，在纹理特征丰富的区域选用小支撑窗口，保障窗口内的像素点具有相似的视差，而在弱纹理区域选用大支撑窗口，保障窗口内包含足够的支撑信息，这就是代价聚合的含义。理想的匹配代价聚合方法应该根据待匹配像素点的局部纹理特征自适应的调节支撑窗口。这一过程可以理解为是对第一步建模方法的优化。

2.3 视差计算

视差计算即通过代价聚合之后的代价矩阵来确定每个像素的最优视差值，通常使用赢家通吃算法（WTA，Winner-Takes-All）来计算，如图 2.3 所示，即某个像素的所有视差下的代价值中，选择最小代价值所对应的视差作为最优视差。

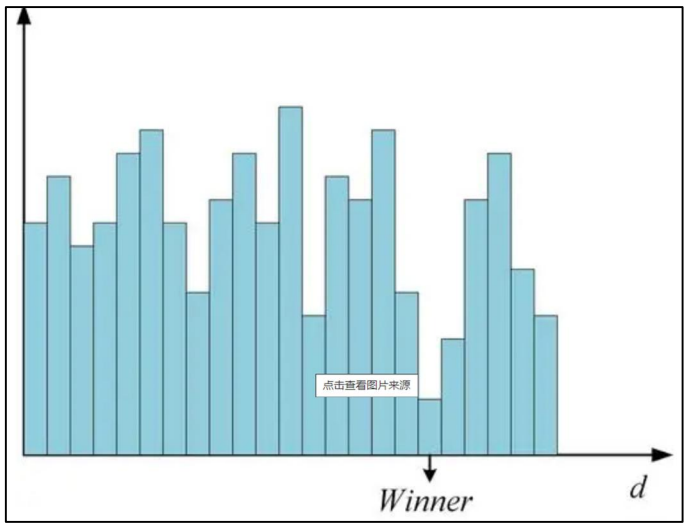

图 2. 3 赢家通吃（WTA）算法示意图

2.4 视差优化

在立体匹配过程中，错误的像素匹配不可避免，上述过程得到的视差结果往往都是不够完美的。视差优化的目的是对上一步得到的视差图进行进一步优化，改善视差图的质量，包括剔除错误视差、适当平滑以及子像素精度优化等步骤，一般采用左右一致性检查（Left-Right Check）算法剔除因为遮挡和噪声而导致的错误视差，即以左视图为参考图像，向右图像中搜索匹配点得到的视差结果，与以右视图为参考图像，向左视图搜索匹配点得到的视差结果，二者应保持一致，

若不一致则判定为错误匹配点；采用剔除小连通区域算法来剔除孤立异常点；采用中值滤波（Median Filter）、双边滤波（Bilateral Filter）等平滑算法对视差图进行平滑；另外还有一些有效提高视差图质量的方法如鲁棒平面拟合（Robust Plane Fitting）、亮度一致性约束（Intensity Consistent）、局部一致性约束（Locally Consistent）等也常被使用。

由于 WTA 算法所得到的视差值是整像素精度，为了获得更高的子像素精度，需要对视差值进行进一步的子像素细化，常用的子像素细化方法是一元二次曲线拟合法，通过最优视差下的代价值以及左右两个视差下的代价值拟合一条一元二次曲线，取二次曲线的极小值点所代表的视差值为子像素视差值。如图 2.4 所示。

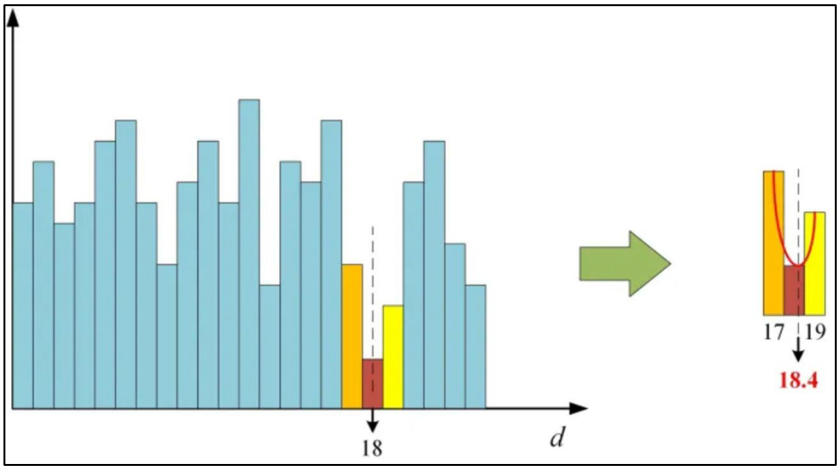

图 2. 4 赢家通吃（WTA）细化算法示意图

2.5 评估准测

实验中使用误匹配百分比（PBM）作为错误率计算的衡量方法。计算公式如下：

$$PBM = \frac{1}{N}\sum_{(x,y)}(|d_c(x,y) - d_T(x,y)| \geq \delta_d) \tag{2.8}$$

式中$d_c(x,y)$是像素点(x,y)实验计算的视差，$d_T(x,y)$是像素点(x,y)的真实视差，N是像素的总个数，δ_d为设置的阈值。PBM 的值越小，那么误匹配率就越低，匹配的准确性越高。

三．实验步骤

立体匹配算法流程图如图 3.1 所示。其中局部匹配算法的步骤一般包括匹配代价计算、代价聚合和视差计算三个步骤，全局算法则包括匹配代价计算，视差计算与视差优化三个步骤，半全局算法 SGM 则四个步骤都有。

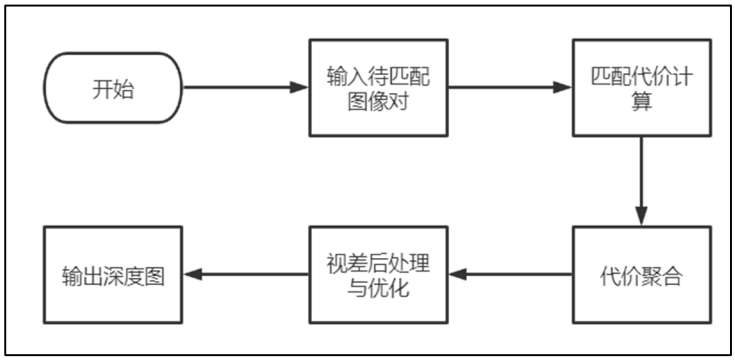

图 3. 1 立体匹配算法流程图

四．程序代码

本次实验选择三种常见的匹配代价测量函数进行匹配代价的计算，包括灰度绝对差和（SAD）、灰度差平方和（SSD）和归一化互相关（NCC）。视差计算选择赢家通吃算法。对比三种代价测量函数的生成结果。窗口大小设为 3，视差范围的最大视差D设为 60。这一部分主要给出三种代价测量函数的代码实现：

- SAD：根据式 2.2 计算 DSI 矩阵。

```
(H,W) = left_image.shape
cost_volume = np.zeros((H,W,max_disparity))

# Loop over internal image
for y in range(filter_radius, H - filter_radius):
  for x in range(filter_radius, W - filter_radius):
    # Loop over window
    for v in range(-filter_radius, filter_radius + 1):
      for u in range(-filter_radius, filter_radius + 1):
        # Loop over all possible disparities
        for d in range(0, max_disparity):
          cost_volume[y,x,d] += np.absolute(left_image[y+v, x+u] - right_image[y+v, x+u-d])
```

图 4. 1 SAD 实现

- SSD：根据式 2.3 计算 DSI 矩阵。

```
(H,W) = left_image.shape
cost_volume = np.zeros((H,W,max_disparity))

# Loop over internal image
for y in range(filter_radius, H - filter_radius):
  for x in range(filter_radius, W - filter_radius):
    # Loop over window
    for v in range(-filter_radius, filter_radius + 1):
      for u in range(-filter_radius, filter_radius + 1):
        # Loop over all possible disparities
        for d in range(0, max_disparity):
          cost_volume[y,x,d] += (left_image[y+v, x+u] - right_image[y+v, x+u-d])**2
```

图 4. 2 SSD 实现

- NCC：根据式 2.4 计算 DSI 矩阵。

```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, mean_r = np.zeros((m, n)), np.zeros((m, n))
    s = np.zeros((m, n))
    s_l, s_r = np.zeros((m, n)), np.zeros((m, n))
    dmaps = np.zeros((m, n, steps)) # 保存深度平面的数组
    # 计算图像块的平均值
    ndimage.filters.uniform_filter(im_l, wid, mean_l)
    ndimage.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
        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)
```

图 4. 3 NCC 实现

- PBM 计算：根据式 2.8 计算 PBM 值。

```python
    number_of_pixels = max(np.sum(mask_image), 1) # Catch error if no pixels selected
    weighted_image = np.absolute(prediction_image - groundtruth_image) >= threshold_disparity
    return "{:5f}".format(1/number_of_pixels*np.sum(weighted_image))
```

图 4. 4 PBM 计算实现

- 赢家通吃算法使用 np.argmin 函数即可实现。

五．实验结果

在两对匹配图像上分别使用上述实现的三种代价测量函数进行视差图的计算，并计算 PBM 值进行对比，δ_d设为 30。在第一对图像上 SAD 和 SSD 效果较差，基于信息差值的代价度测量函数对光照变化，噪声的鲁棒性低。而基于相关性的成本测量函数 NCC 效果较好，对于噪声和光照变化有一定的鲁棒性。第二对图像三者的 PBM 值差不多。

图 5. 1 Adirondack

图 5. 2 三种代价计算函数结果

图 5. 3 cones

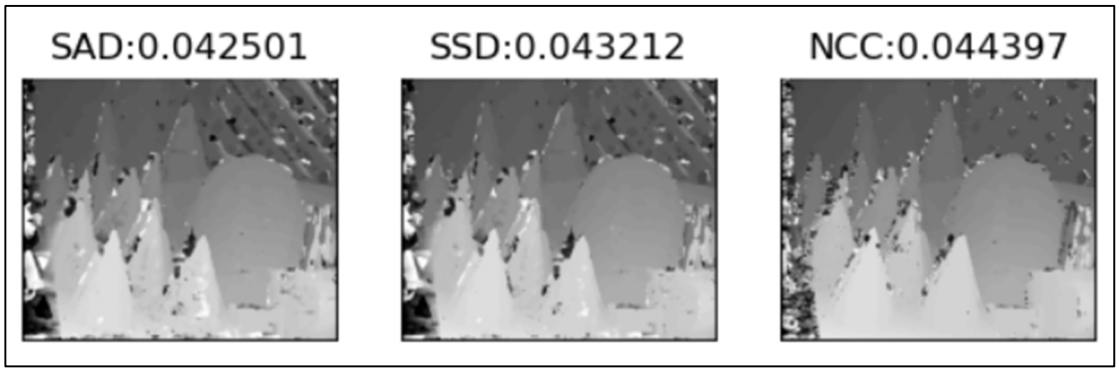

图 5. 4 三种代价计算函数结果

六．实验分析与总结

本次实验使用常见的三种代价测量函数，通过立体匹配计算得到两张图像的视差图，完成了实验内容，理解并掌握了立体匹配计算图像视差图的方法与原理。

在视差匹配中，常用的代价测量函数包括 NCC、SAD 和 SSD，它们都可以评估两个对应像素之间的相似程度。实验证明了 NCC 能够对光照变化具有一定的鲁棒性，SAD 和 SSD 计算简单但对噪声敏感。在实际应用中，不同的代价测量函数会对匹配效果产生不同的影响，需要综合考虑多种因素进行算法设计和优化，结合具体场景和需求选择合适的代价测量函数，以提高匹配效率和精度。