计算机视觉应用与实践（五）

目录

一、 实验目的

了解图像视差匹配的过程，并理解立体匹配的原理。

二、 实验原理

立体匹配是一种计算机视觉技术，用于在两个或多个摄像机或传感器捕捉到的图像中，自动计算物体表面上不同点之间的距离或深度信息。立体匹配的原理是通过比较同一物体在两个不同视角下的图像中对应的像素值，找到它们之间的匹配点，从而推导出这些点之间的距离或深度信息。

本次实验采用的是基于图像块的立体匹配方法，它是一种常见的立体匹配算法。首先，它将图像分成若干个块，然后针对每个块枚举视差并计算当前块与另一张图像中枚举视差所对应的图像块之间的相似度指标，将相似度指标最优的视差作为该块的视差，并最终得到视差图。常用的相似度指标机器计算公式如下所示：

SSD（Sum of Squared Differences），值为左右图像像素块的平方差值之和。

$$L\big(I_l(x,y),I_r(x+d,y)\big)=(I_l(x,y)-I_r(x-d,y))^2$$

SAD（Sum of Absolute Differences），值为左右图像像素块的绝对差值之和

$$L\big(I_l(x,y),I_r(x+d,y)\big)=|I_l(x,y)-I_r(x-d,y)|$$

NCC（Normalized Cross Correlation），值为左右图像像素块的归一化互相关

值

$$L\left(I_l(x,y),I_r(x+d,y)\right)=\frac{I_l(x,y),I_r(x-d,y)-I_{lmean}*I_{rmean}}{\sigma_l\sigma_r(d)}$$

三、 程序代码

首先，读入图像并做一定的处理

```
img1 = cv2.imread('data/imL.png',0)
img2 = cv2.imread('data/imR.png',0)
def img_aug(img):
    img = cv2.GaussianBlur(img, (3, 3), 0)
    img = cv2.equalizeHist(img)
    return img
img1 = img_aug(img1)
img2 = img_aug(img2)
```

然后定义相似度指标的计算函数

```
def cal_ncc(left_block, right_block):
    # 求两个块的像素值平均值
    block_size = left_block.shape[0] * left_block.shape[1]
    left_mean = np.mean(left_block)
    right_mean = np.mean(right_block)

    # 求两个块的像素值标准差
    left_std = np.std(left_block)
    right_std = np.std(right_block)

    # 计算协方差
    covariance = np.sum((left_block - left_mean) * (right_block - right
_mean))

    # 计算NCC值
    ncc = covariance / (left_std * right_std * block_size)

    return -ncc

def cal_sad(left_block, right_block):
    return np.sum(np.abs(left_block-right_block))

def cal_ssd(left_block, right_block):
    return np.sum(np.square(left_block-right_block))
```

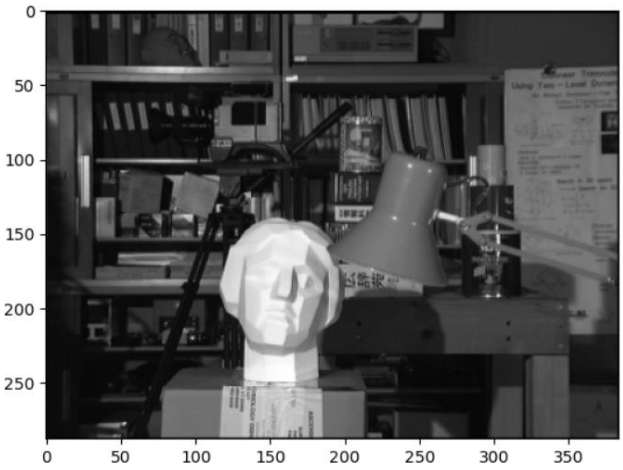
之后实现基于块的立体匹配算法，得到视差图。

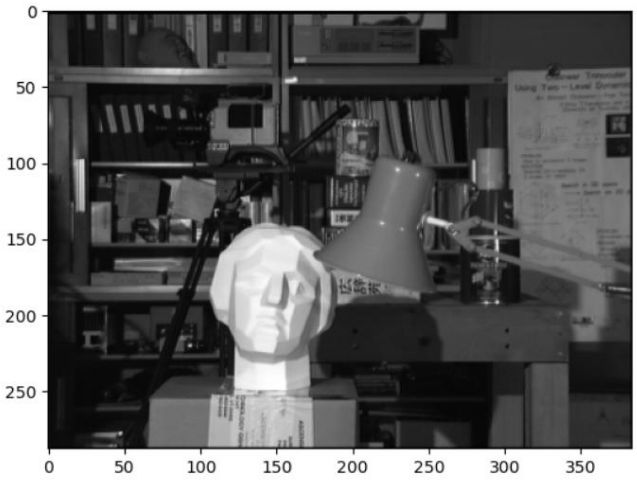
```
def block_match(left_img, right_img, block_size, max_disp,cost_fun ):
```

```
    # 获取图像尺寸和通道数
    height, width = left_img.shape

    # 初始化视差图
    disp_map = np.zeros((height, width))

    # 设置搜索范围
    search_range = range(max_disp)

    # 遍历左图像中的每个块
    for y in range(0, height - block_size + 1):
        for x in range(0, width - block_size + 1):
            # 获取左图像中的当前块
            left_block = left_img[y:y+block_size, x:x+block_size]

            # 初始化最小 SAD 值和最优视差值
            min_cost = np.inf
            best_disp = x

            # 遍历右图像中的每个可能的视差块
            for d in search_range:
                if x-d<0:
                    break
                right_block = right_img[y:y+block_size,x-d:x-d+block_si
ze]

                # 计算当前视差块和左图像中的当前块之间的 SAD 值
                cost = cost_fun(left_block,right_block)

                # 更新最小 SAD 值和最优视差值
                if cost < min_cost:
                    min_cost = cost
                    best_disp = d

            # 将最优视差值存储到视差图中
            disp_map[y:y + block_size, x:x + block_size] = best_disp *
(255 / max_disp)

    return disp_map
```

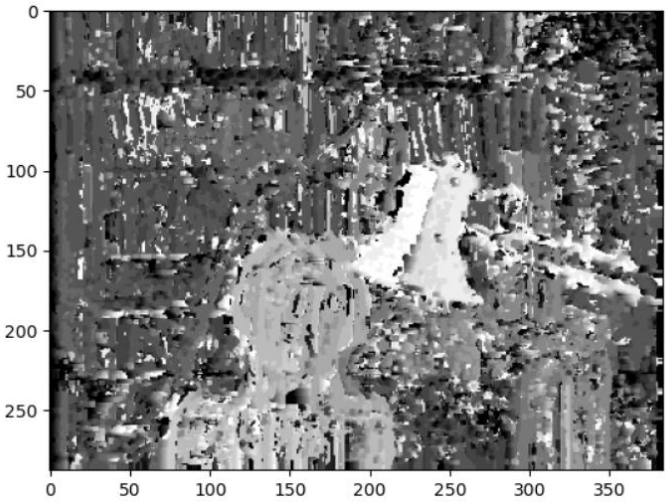
四、 实验结果

输入图像

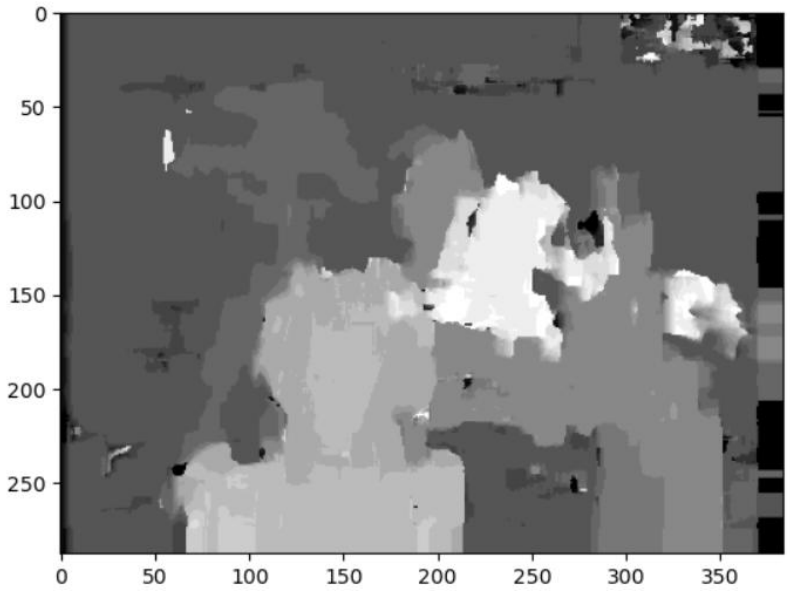




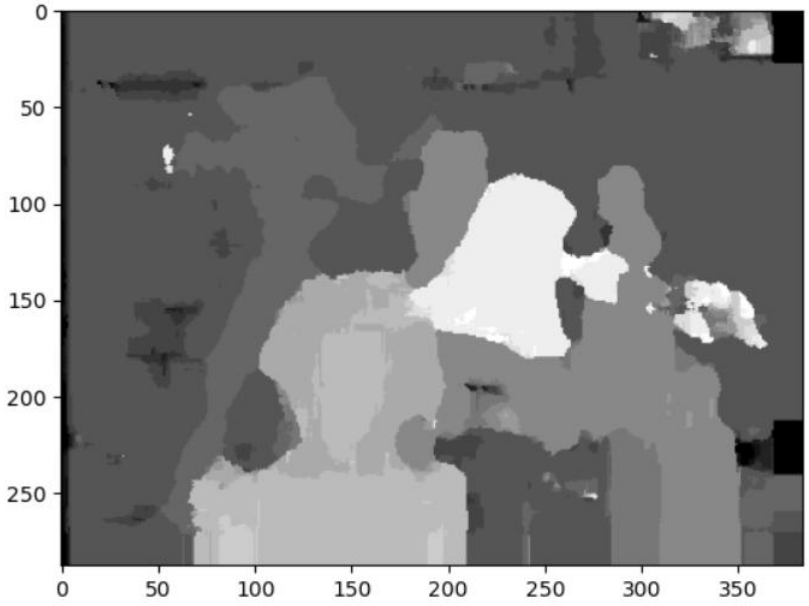
SAD 的实验结果，图像块大小为 3 最大视差为 16 。



NCC 的实验结果，图像块大小为 13 最大视差为 16 。



SSD 的实验结果，图像块大小为 15 最大视差为 16 。



深度图结果显示，SSD（Sum of Squared Differences）方法的匹配精度最优，NCC（Normalized Cross Correlation）方法的匹配精度次优，而 SAD（Sum of Absolute Differences）方法的匹配精度最差。这种现象的原因是三种方法在像素匹配过程中所使用的相似性度量函数不同。SSD 方法是基于像素值的欧式距离，NCC 方法是基于像素值的相关系数，而 SAD 方法是基于像素值的绝对值差异。

SSD 方法在计算匹配误差时，对于较大的像素值差异赋予了较高的权重，因此在存在一定噪声的情况下，匹配结果相对较准确。而 NCC 方法则可以消除光照和对比度等影响，因为它使用的是像素之间的相关性，而不是像素值本身。SAD

方法则对噪声比较敏感，因为它只关注像素之间的绝对差异。

五、 实验总结

通过这次实验，我了解了图像视差匹配的基本流程，实现了基于图像块的立体匹配方法，并对比了三种不同相似度指标对于匹配精度的影响。总的来说，本次实验让我对图像视差匹配和立体匹配方法有了更深入的了解，同时也提高了我的编程能力，对于我未来在计算机视觉领域的研究和应用都会有所帮助。