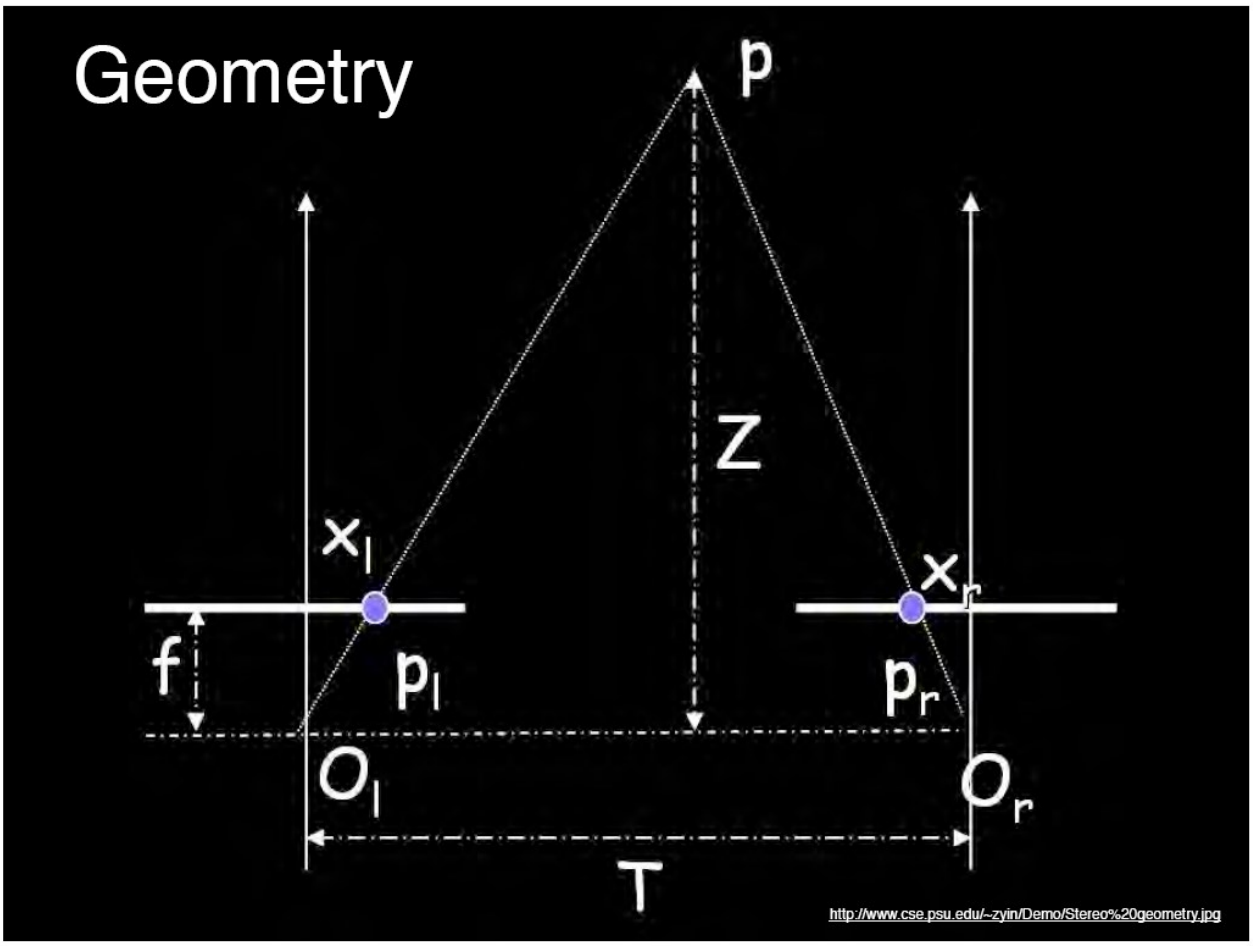
人工智能导论cv实验报告

黄志鹏 PB16150288

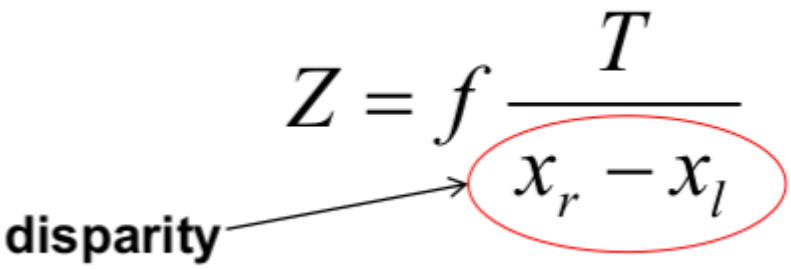
1. 实验内容

1.1 实验原理

- 利用下图的几何分析可以知道, p点深度z 和xl - xr 成反比



反比公式如下:



- 寻找每个pixel的对应pixel, 然后x轴距离相减, 就可以得到xr - xl 的值

$$(x´,y´)=(x+D(x,y), y)$$

1.2 代码原理

全部代码见附件, 下面分析核心的代码:

- 第一步 读入数据 转化为灰度图像

```
    im2 = Image.open(os.path.join(path, "im2.png")).convert('RGB')
    im6 = Image.open(os.path.join(path, "im6.png")).convert('RGB')
    ...
    img = np.mean(img, axis=2, dtype=int)
```

- 第二步 遍历每一个像素 (x, y) 分析 时差 d(x', x)

```
def disparity_x_y(x:int, y:int, imgL:np.ndarray, imgR:np.ndarray, winSize):
    winL = getWindow(x, y, imgL, winSize)
    ssdMin = 9999999999
    d = 0
    for j in range(5, 40):
        if y + j + winSize[1] > imgL.shape[1]:
            continue
        winR = getWindow(x, y + j, imgR, winSize)
        ssdNow = ssd(winL, winR)
        if ssdNow < ssdMin:
            ssdMin = ssdNow
            d = j
    return d
```

- 将上面的结果存储在矩阵m中, 并打印出视差图像

```
def disparityMap(imgL:np.ndarray, imgR:np.ndarray, winSize):
```

```
    m = np.zeros(imgL.shape, dtype=int)
    d = np.zeros(imgL.shape, dtype=int)
    for x in range(imgL.shape[0]):
        for y in range(imgL.shape[1]):
            print((x, y))
            try:
                m[x][y] = disparity_x_y(x, y, imgL, imgR, winSize)
            except ZeroDivisionError:
                m[x][y] = 0
    mmin = m.min()
    mmax = m.max()
    m = (m-mmin)/(mmax-mmin)*255 # (矩阵元素-最小值)/(最大值-最小值)
    scipy.misc.imsave('outfile_m10.jpg', m)
```

- 利用视察和距离的反比关系构造深度图d, 并打印出深度图像

```
    for x in range(m.shape[0]):
        for y in range(m.shape[1]):
            if m[x][y] != 0:
                d[x][y] = 255 / m[x][y]
            else:
                d[x][y] = 0
    dmin = d.min()
    dmax = d.max()
    d = (d-dmin)/(dmax-dmin)*255 # (矩阵元素-最小值)/(最大值-最小值)
    scipy.misc.imsave('outfile_d10.jpg', d)
```

2. 实验结果

- 使用1X1的窗口

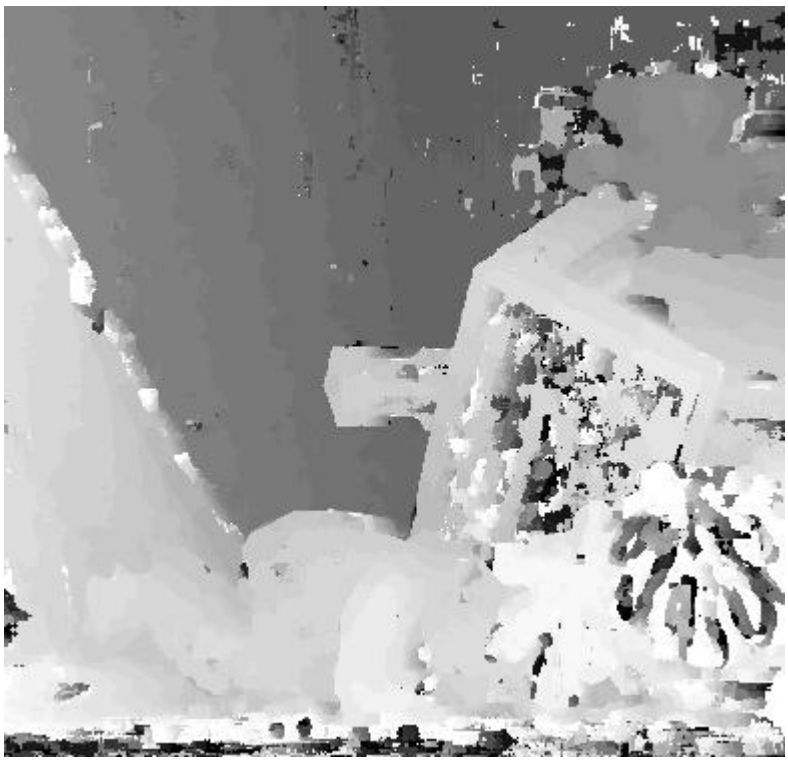
  视差图像:

运算时间：11s

- 使用3X3的窗口

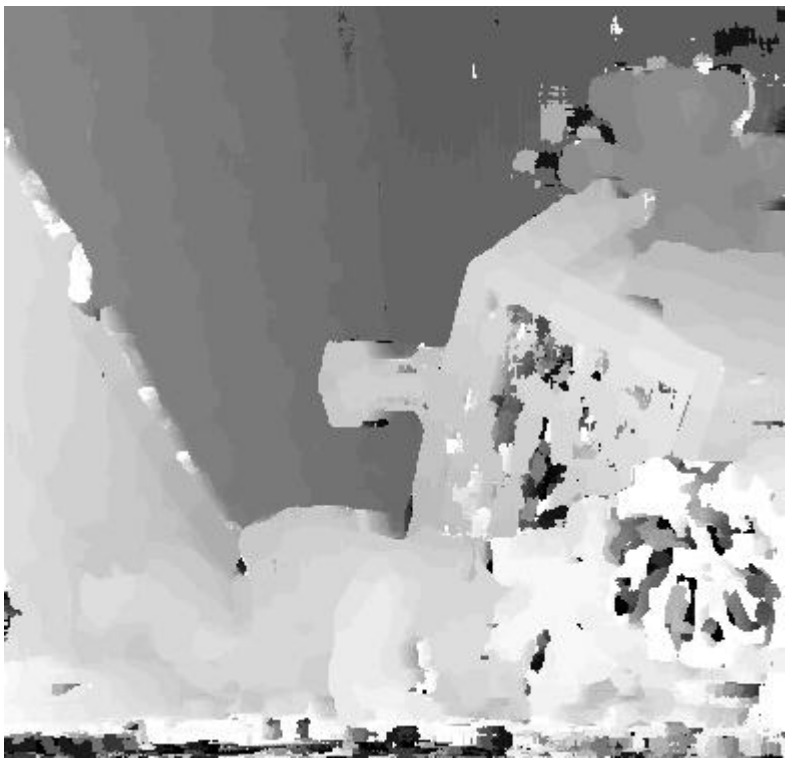
运算时间：32 s

- **使用7X7的窗口**



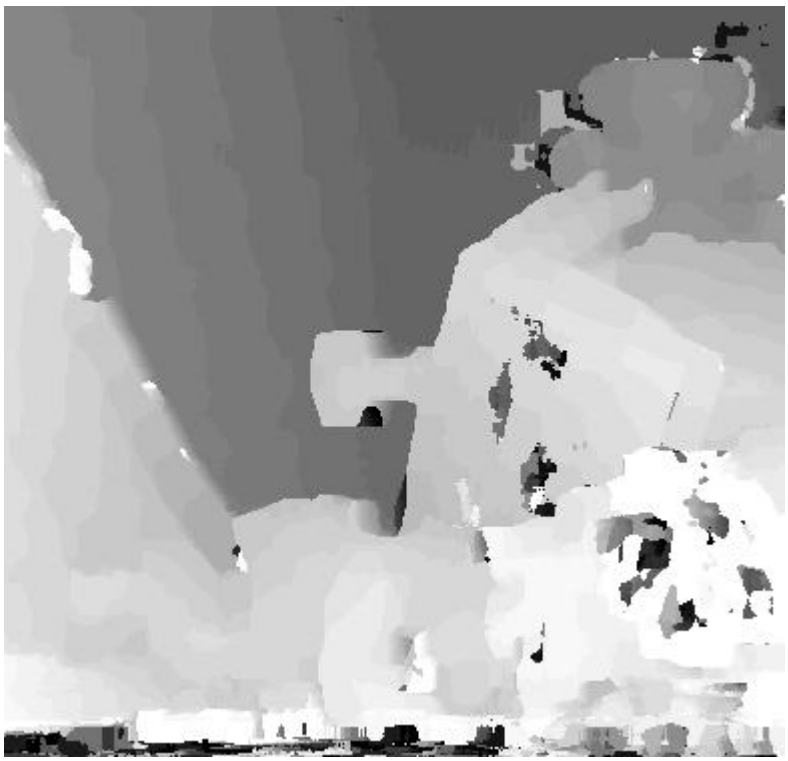
运算时间: 1min

- **使用10X10的窗口**

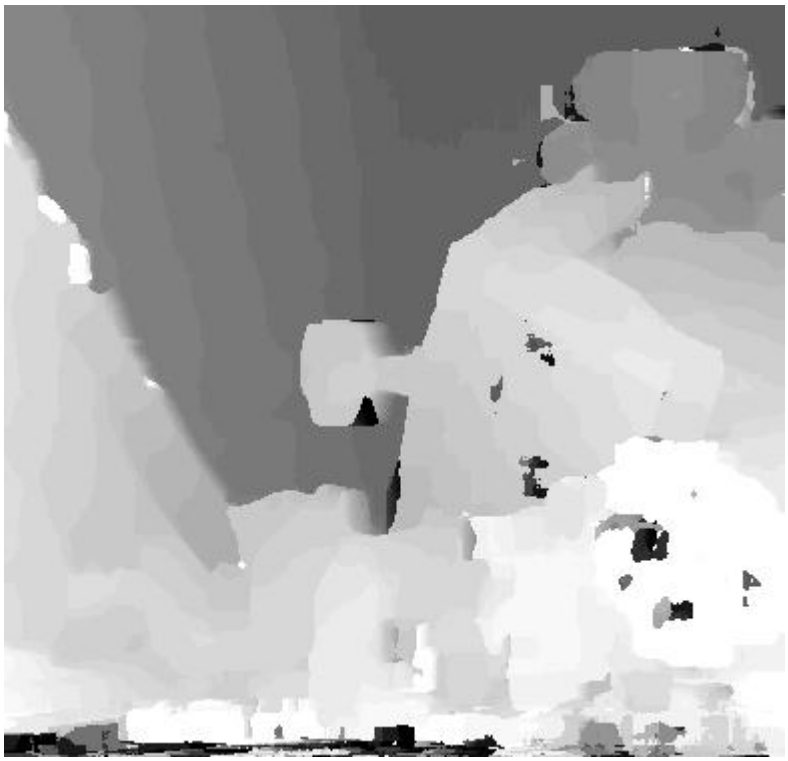


运算时间: 1min20s

- 使用15X15的窗口



- 使用20X20的窗口



3. 实验分析

3.1 实验结果分析

- 从实验结果来看, 可以看出**由7X7窗口和10X10窗口产生的视差图, 与ground true 进行对比, 发现结果大致一致.**但是仍然后一定的**瑕疵和误差**, 分析可能是由于以下的**原因**:

    - 窗口大小对计算结果的影响较为明显, **如果窗口过小, 则随机误差较大, 如果窗口较大,则平均误差较大**
    - 考虑到计算的速度, 没有对原始图像进行**pad操作**, 使得原始图像边缘一侧会有很大的误差, 如果要将这些误差消除, 则要对原始图像的一侧边缘精巧的进行padding, 且ssd的计算方式也需要改, 会增加很多的计算负担
    - 计算有**一定的整数运算**, 可能会造成一定的随机误差, 这会是视差图中一些随机误差产生的愿意之一
    - **ssd**这个相关过于简单, 体现的相似性, 但是**没有体现相关性**

- 实验最初,的代码使用了numpy的**padi**函数, 使得计算的速度**非常缓慢**. 在删除了np.pad函数之后, 代码的速度变得很快.
- 通过对15X15, 20X20结果图片中盆栽部分, 和墙壁背景的分析, 可以看出, 这种算法在**滑动窗口大时**, **对于一些大块均匀, 疏松的, 边界简单的图片部分(如墙壁背景), 效果比较好**, 但是对于如盆栽部分那里那中, **深度突变比较密集**的, 在大滑动窗口的情况下, 可能会跨过目标区域, 然后**产生较大的平均误差, 效果不好**
- 通过对1X1, 3X3结果图片中的盆栽部分, 和墙壁背景部分的分析, 可以看出, 这种算法在**滑动窗口小时, 对一些变化剧烈密集的部位效果明显**, 但是在整块的**均匀的(如背景部分)**, 小窗口容易产生**较大的噪声(随机误差)**

3.2 实验方法之后可以完善的地方

- 通过上面的窗口的分析, 可以尝试使用**多尺寸窗口(大和小) 同时滑动计算**的方法, 来均衡这种不同区域的不平衡. 将它们的结果按照一定的**准则进行取舍.**
- 改进ssd 算法, 而采**用卷积的方式**, 来提高相关性