计算机视觉实践-练习 5

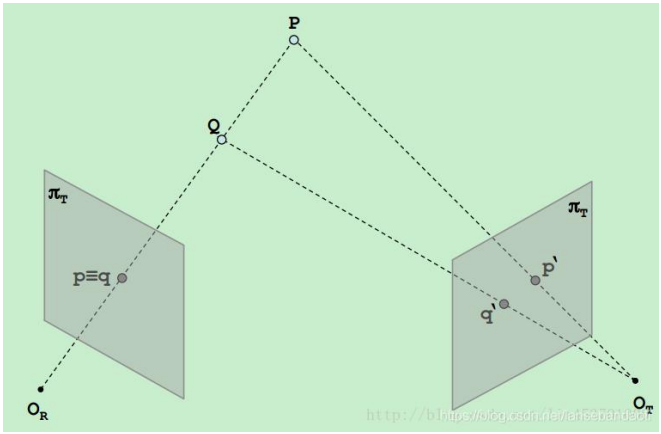
122106222784 贺梦瑶

1、实验目标

通过立体匹配得到两张图像的视差图，即图像视差匹配。

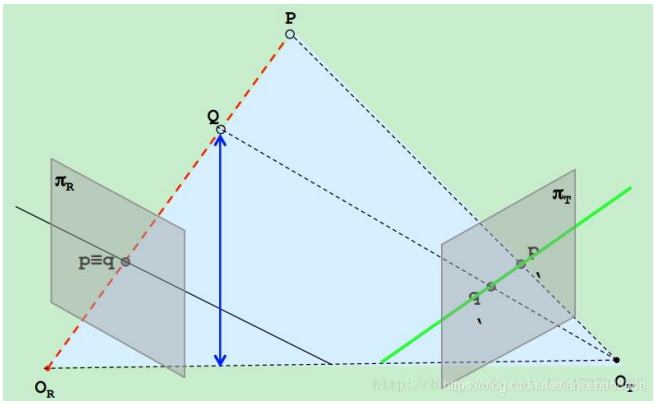
2、模型介绍

立体匹配是立体视觉研究中的关键部分。其目标是在两个或多个视点中匹配相应像素点，计算视差。通过建立一个能量代价函数，对其最小化来估计像素点的视差，求得深度。



点 P 和 Q，映射到左相机 OR 像面上的同一点 p≡q，只要找到 p 和 q 在右相机 OT 像面上的对应点就可以通过三角计算估计深度。找到对应点的过程，即立体匹配。

为了找到对应点，需要增加约束，最常用的是极线约束。



P 和 Q 映射到左相机 QR 像面上的同一点 p≡q，直线 pq 上的点对应点一定位于右像面的直线 p’q’上，p’q’即为直线 pq 的极线，这就是极线约束。

SAD 算法是立体匹配算法中，隶属于局部区域匹配算法中的一个算法，原理是从图 1 中找出一个小窗口，利用极线约束，在图 2 中同一行中 间隔 D 为 D_1 处找到同样大小的一个小窗口，比方说都是 9*9 大小， 然后比较这个小窗口中每一个像素灰度差是多少,比如说灰度差为 x_1，接着 更换间隔 D 为 D_2，再次计算灰度差 x_2，比较灰度差最小的即为图 2 中的用一位置。

3、实现说明

(1) SAD 算法

```
1. imgDiff=np.zeros((img_size[0],img_size[1],maxDisparity))
2. e = np.zeros(img_size)
3. for i in range(0,maxDisparity):
```

```
    e=np.abs(rimg[:,0:(img_size[1]-i)]- limg[:,i:img_size[1]]) #视差为多少，那么生成的图像就会少多少像素列,e 负责计算视差为 i 时，两张图整体的差距
    e2=np.zeros(img_size) #计算窗口内的和
    for x in range((window_size),(img_size[0]-window_size)):
        for y in range((window_size),(img_size[1]-window_size)):
            e2[x,y]=np.sum(e[(x-window_size):(x+window_size),(y-window_size):(y+window_size)])#其实相当于用 111 111 111 的卷积核去卷积，如果用 tensorflow 会不会更快一些，其实就是先做差再加和以及先加和再做差的关系
        imgDiff[:,:,i]=e2
dispMap=np.zeros(img_size)
```

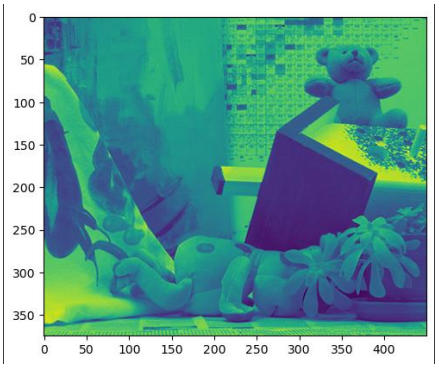
(2) 获得灰度最小视差

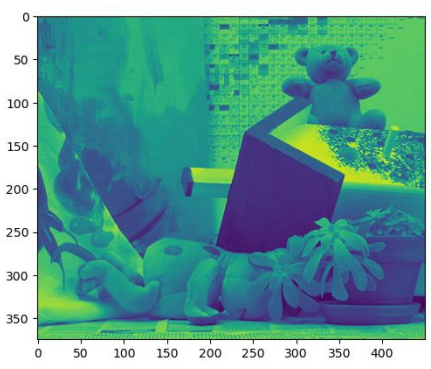
```
for x in range(0,img_size[0]):
    for y in range(0,img_size[1]):
        val=np.sort(imgDiff[x,y,:])
        if np.abs(val[0]-val[1])>10:
            val_id=np.argsort(imgDiff[x,y,:])
            dispMap[x,y]=val_id[0]/maxDisparity*255#其实 dispmap 计算的是视差的大小，如果视差大，那么就相当于两张图片中同样物品的位移大，就是距离近
print('用时:',time.time()-tic1)
```

4、结果展示

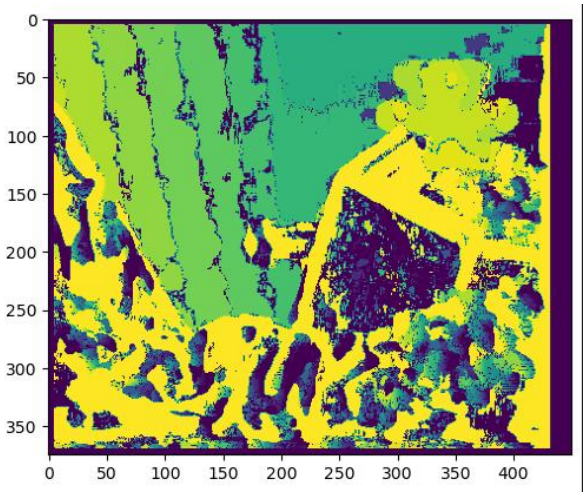
下图分别为左视与右视两张图像：

经处理后得到灰度图像：





通过匹配得到视差图：



SAD 算法虽然简单高效，但是容易受光照、曝光等噪声影响，视差图的效果往往不是特别鲁棒。