点云作业第四讲——地面分割+地物聚类

主讲人 zhp曾

整体流程

深蓝学院
shenlanxueyuan.com

- 读取数据
- 预处理
  - 体素滤波、高度滤波、分块分割
- 地面分割
- 前景聚类

预处理

深蓝学院 shenlanxueyuan.com

- Voxel Grid
- 高度滤波
- 分块分割

```python
#分区
x_filter_1 = voxel_filtered_pc[:,0] >= 0.0
y_filter_1 = voxel_filtered_pc[:,1] >= 0.0
x_filter_2 = voxel_filtered_pc[:,0] < 0.0
y_filter_2 = voxel_filtered_pc[:,1] < 0.0
filter_1 = np.logical_and(x_filter_1,y_filter_1)
filter_2 = np.logical_and(x_filter_1,y_filter_2)
filter_3 = np.logical_and(x_filter_2,y_filter_2)
filter_4 = np.logical_and(x_filter_2,y_filter_1)
#分区做ransac去平地
segmented_points_1 = ground_segmentation(data=voxel_filtered_pc[filter_1,:])
segmented_points_2 = ground_segmentation(data=voxel_filtered_pc[filter_2,:])
segmented_points_3 = ground_segmentation(data=voxel_filtered_pc[filter_3,:])
segmented_points_4 = ground_segmentation(data=voxel_filtered_pc[filter_4,:])
#合并
segmented_points = np.vstack((segmented_points_1,segmented_points_2,segmented_points_3,segmented_points_4))
```

```python
for i in range(len(h_sorted) - 1):
    if h_sorted[i] == h_sorted[i + 1]:
        continue
    else:
        pt_idx = h_indices[begin:i+1]#左闭右开区间
        filtered_points.append(np.mean(pc_points[pt_idx],axis=0))
        begin = i + 1

filtered_points = np.array(filtered_points,dtype=np.float64)
return filtered_points
```

$x >= 0, y >= 0$

$x >= 0, y < 0$

$x < 0, y < 0$

$x < 0, y >= 0$

4个大区域

每个子区域中分别使用RANSAC

```python
# 屏蔽开始
z_filter = data[:, 2] < lidar_height
z_filter_down = data[:, 2] > lidar_height_down
filt = np.logical_and(z_filter_down, z_filter)
data_filtered = data[filt, :]
```

地面分割

● RANSAC流程

1、确定迭代次数：

2、在迭代次数内：

2.1 随机选取三个不共面的点组成平面

2.2 求平面方程

2.3 求所有点到平面距离

2.4 统计inliner数量

3、迭代选择inlier个数最多的平面

确定迭代次数

- 指定迭代次数；
- 计算理论迭代次数；

$$N = \frac{log(1-p)}{log(1-(1-e)^s)}$$

```python
def ground_segmentation(data,inliner_ratio = 0.2):
    # 作业1
    # 屏蔽开始
    #使用ransac
    sigma = 0.2 #阈值
    num_iter = math.ceil(math.log(1-0.999) / math.log(1-pow(inliner_ratio,3)))
```

选取三个不共面的点

- 判断共面关系

1、满足满秩矩阵

2、利用比例关系

```
while True:
    sample_index = random.sample(range(sz),3)
    p = data[sample_index,:]
    if np.linalg.matrix_rank(p)==3:
        break
```

```
vector1 = xyz[1,:] - xyz[0,:]
vector2 = xyz[2,:] - xyz[0,:]

# 共线性检查 ; 0过滤
if not np.all(vector1):
    # print('will divide by zero..', vector1)
    return None
dy1dy2 = vector2 / vector1
# 2向量如果是一条直线, 那么必然它的xyz都是同一个比例关系
if  not ((dy1dy2[0] != dy1dy2[1])  or  (dy1dy2[2] != dy1dy2[1])):
    return None
```

计算平面方程

- 平面参数abcd
- 点法式
  - 法向量+平面上一点

计算平面参数abcd

计算平面方程：

```python
#求由x点组成的平面的方程
a = (X[1,1] - X[0,1])*(X[2,2] - X[0,2]) - (X[2,1] - X[0,1])*(X[1,2] - X[0,2])
b = -(X[1,0] - X[0,0])*(X[2,2] - X[0,2]) + (X[2,0] - X[0,0])*(X[1,2] - X[0,2])
c = (X[1,0] - X[0,0])*(X[2,1] - X[0,1]) - (X[2,0] - X[0,0])*(X[1,1] - X[0,1])
ABC = np.zeros((3,1))
ABC[0] = a
ABC[1] = b
ABC[2] = c
d = np.dot(X[0,:],ABC)[0]
print('a',a,'b',b,'c',c,'d',d)
```

计算距离：

```python
#求所有点到平面的距离
vector = data - X[0,:]
distance = np.dot(vector,ABC)/np.linalg.norm(ABC)
distance = np.abs(distance)
```

$$d = \frac{|Ax_1 + By_1 + Cz_1 + D|}{\sqrt{A^2 + B^2 + C^2}}$$

点法式

- 点法式：

平面 π：

π 上一点： $M_0\,(x_0, y_0, z_0)$

垂直于 π 的法向量： $n = (A, B, C)$

则： $n \bullet \overline{M_0M} = (A, B, C) \bullet (x - x_0, y - y_0, z - z_0) = 0$

- 计算距离：

$$d = \frac{\overrightarrow{M_0M_1} \cdot \vec{n}}{\|\vec{n}\|}$$

```python
# 2. solve model: 计算平面单位法向量 n
p12 = p2 - p1
p13 = p3 - p1
n = np.cross(p12, p13)
n = n / np.linalg.norm(n)  # 单位化

# 3. computer distance(error function):
count = 0
for point in data:
    d = abs(np.dot((point-p1), n))
```

求所有点到平面距离

- 注意是使用何种类型的参数来计算点到平面距离

```python
#用LSQ精化模型参数
def LSQ(data):
    H = np.cov(data.T)
    eigenvalues,eigenvectors = np.linalg.eig(H)
    sorted_idx = np.argsort(np.real(eigenvalues))
    a,b,c = eigenvectors[sorted_idx[0]]
    xyz_means = data.mean(axis=0)
    d = -(a*xyz_means[0] + b*xyz_means[1] + c*xyz_means[2])
    params = [a,b,c,d]
    return params
```

```python
dist = abs((best_a*data[:,0]+best_b*data[:,1]+best_c*data[:,2]-best_d)/(np.sqrt(best_a*best_a+best_b*best_b+best_c*best_c)))
#精化参数,注意平面参数变为点法式参数了, dist计算方式不同了
params = LSQ(data[dist<sigma,:])
#print(params)
scene_idx = []
for idx,point in enumerate(data):
    distance = abs(params[0]*point[0] + params[1]*point[1] + params[2]*point[2] + params[3])
    if distance >= sigma:
        scene_idx.append(idx)
segmengted_cloud = data[scene_idx,:]
# 屏蔽结束
```

Inliner统计

- 只使用inlier作为判断条件的不足；

  导致某个点数较多的非地面平面占据inlier个数；

  避免将平直墙面检测为地面，必须将夹角加入判断条件；

```
alphaz_threshold = math.pi / 18.0
```

```
#法向量与Z轴(0, 0, 1)的夹角
z = np.array([0,0,1])
alphaz = math.acos(abs(np.dot(n,z)/np.linalg.norm(n)))

if total_inliner > pre_total and alphaz < alphaz_threshold:
```

前景聚类

深蓝学院 shenlanxueyuan.com

●算法流程

1、创建访问记录矩阵;

2、循环直到所有点标记为visited;

2.1 在未标记的点中随机选择初始点，并修改状态;

2.2 获取初始点r半径范围内的近邻;

2.3 若近邻数量小于min_samples，则标记为noise; 大于等于min_samples，标记为核心点;

2.4 从初始点创建新的聚类;

3、遍历其近邻;

```python
while len(unvisit_list) > 0:
    p = random.choice(unvisit_list)
    unvisit_list.remove(p)
    visit_list.append(p)
    #N为搜索过程中动态管理所有核心点及其radius领域点索引的合集，最终所有N内点将被访问且标记为该类簇
    N = sci_kdtree.query_ball_point(X[p],self.eps)
    #判断p是否为核心点
    if len(N) >= self.minpts:
        k += 1#初始化一个新类簇
        labels[p] = k
        for pi in N:
            if pi in unvisit_list:
                unvisit_list.remove(pi)
                visit_list.append(pi)#N是动态的，所以最终N内点即该类簇点都会标记为访问过了
                M = sci_kdtree.query_ball_point(X[pi],self.eps)
                #判断pi是否为新的核心点
                if len(M) >= self.minpts:
                    #把pi的领域点都加入N
                    for pt in M:
                        if pt not in N:
                            N.append(pt)
            if labels[pi] == -1:
                labels[pi] = k
    else:
        #噪声
        labels[p] = -1
return labels
```

深蓝学院
shenlanxueyuan.com

感谢各位聆听！

Thanks for Listening