深蓝学院
shenlanxueyuan.com

第四章作业分享

主讲人 张点堃

地面点去除——思路分析

- 地面点云去除主要需要用到模型拟合的方法。课上一共介绍了三种方法，分别是最小二乘法、Hough变换和RANSAC。
- 其中最小二乘法在这里是完全不适用的，因为对于本任务来说，所有非地面点都是outliers，这对最小二乘法的解影响很大。
- Hough 变换也太适用，因为在空间中确定一个平面需要4 个参数(其中三个是独立的)。3D空间内的Hough变换搜索范围较大，效率不高。
- 最终选择使用RANSAC方法来实现地面点的去除。

地面点去除——思路分析

- 因此最终适用的是RANSAC 方法：其具体步骤(实现思路) 如下，基本与课上内容一致：
  - 1、随机选择3 个点（确定平面法向量三分量需要三个方程），求解其法向量，进而得到平面方程参数；
  - 2、计算所有点到平面的距离，并根据阈值 δ 计算inliers 数量；
  - 3、重复1、2，并选取具有最多inliers 的参数。

地面点去除——具体实现

- 平面方程参数有4个，其中三个独立。

  随机选用三个点，求解平面方程参数。

- 三点共线无法求解得到唯一平面，需要进行一次判别

地面点去除——具体实现

深蓝学院 shenlanxueyuan.com

- 求解平面方程需要对欠定线性方程组求解，本实现使用了sympy来完成这个任务。

- 判断是否共线

```
choose_points=random.choices(data,k=n)
C=np.zeros([n,n])
for jj in range(n):
    C[jj,:]=choose_points[jj]
if np.linalg.matrix_rank(C)==0:    #如果选出的三点共线，则
    continue
```

```
#计算各点之间的位置关系，方便求法向量
D=np.zeros([n,n])
D[1,:]=C[0,:]-C[n-1,:]
for jj in range(1,n):
    D[jj,:]=C[jj,:]-C[jj-1,:]
x = sympy.symbols('x')
y = sympy.symbols('y')
z = sympy.symbols('z')
#建立方程
eqs=D@[x,y,z]
s=sympy.solve(eqs,[x,y,z])
#这个解一定有一个基础解系，根据解的情况构建基础解系
if x not in s:
    x_v=1
    normal=np.array([x_v,s[y].subs(x,x_v),s[z].subs(x,x_v)]).astype(np.float)
    normal=normal/np.linalg.norm(normal)
elif y not in s:
    y_v=1
    normal = np.array([s[x].subs(y,y_v), y_v, s[z].subs(y,y_v)]).astype(np.float)
    normal = normal / np.linalg.norm(normal)
elif z not in s:
    z_v=1
    normal = np.array([s[x].subs(z,z_v), s[y].subs(z,z_v), z_v]).astype(np.float)
    normal = normal / np.linalg.norm(normal)
#计算常数D
D=-C[1,:]@normal.T
#计算所有点到面的距离
dis=np.abs(data@normal.T+D)
```

点云聚类——思路分析

- 点云聚类存在的问题有：
  - 1、一个场景中存在的物体数量（聚类类别）未知；
  - 2、点云数据一般分布在物体表面，其分布并不服从高斯分布。

点云聚类——思路分析

- 根据上述两个问题：

  - 1、由于聚类类别未知，因此一种需要类别数先验知识的方法均不可用，比如:Kmeans,Gmm;

  - 2、由于其并不服从高斯分布，则潜在基于高斯分布假设、或适用于高斯分布数据的聚类方法不再可用，比如Kmeans，GMM，Meanshift等等。

点云聚类——思路分析

- 综上，可以使用的方法主要有两种：
  - 1、谱聚类，复杂度O(n^3)
  - 2、DBSCAN，复杂度O(nlogn)，在使用KD树情况下。

- 谱聚类的复杂度较高，并且虽然谱聚类可以动态的判别数据的类别数，但是效果并不稳定。所以最后选择了DBSCAN作为最终的聚类方法。

点云聚类——具体实现

深蓝学院
shenlanxueyuan.com

- 随机选择一个点，如果是核心点，则认为是一个新类别的开始，之后遍历其所有近邻。以此类推。

- 上述思路很容易想到递归实现。

```
def fit_recursive(self,data): #递归对点进行分类
    #data:点云输入，N*3
    N=data.shape[0]
    #修改最大递归次数，否则很容易超过最大递归次数(3000)
    sys.setrecursionlimit(N)
    cls=np.zeros(N)
    access=np.zeros(N)
    radius_result=result_set.RadiusNNResultSet(self.radius)
    KD_root=kdtree.kdtree_construction(data,leaf_size=self.leaf_size)
    label=0
    for ii in range(N):
        if access[ii]==1:
            continue
        query = data[ii, :]
        radius_result = result_set.RadiusNNResultSet(self.radius)
        kdtree.kdtree_radius_search(KD_root, data, radius_result, query)
        indices = [radius_result.dist_index_list[jj].index for jj in range(radius_result.size())]
        if len(indices) < self.Min_Pts:
            cls[ii] = -1
            access[ii] = 1
            continue
        else:
            access[ii] = 1
            cls[ii] = label
            for ind in indices:
                self.recursive_scan(KD_root,data,cls,access,label,ind)
            label+=1
    # cls[cls==-1]=label

    self.cls=cls
    self.access=access
```

点云聚类——具体实现

- 递归实现子程序。

- 问题：这个方法实际上跑步起来，因为一个Sample的每个点都要用递归函数遍历一遍，一共上万个点，很轻松就会发生栈溢出错误。

```
def recursive_scan(self,root,data,cls,access,label,index):
    #cls:记录分类
    #access：记录访问
    #label:当前分类
    if access[index]==1:
        return
    else:
        access[index]=1
        cls[index]=label
    query=data[index,:]
    radius_result = result_set.RadiusNNResultSet(self.radius)
    kdtree.kdtree_radius_search(root,data,radius_result,query)
    indices=[radius_result.dist_index_list[jj].index for jj in range(radius_result.size())]
    if len(indices)<self.Min_Pts:
        return
    for ind in indices:
        self.recursive_scan(root,data, cls, access, label, ind)
```

点云聚类——具体实现

- 递归和循环一般都是可以替换的。将上述程序改为循环。

- 为了保持递归的点遍历顺序，维护两个列表。分别存储待遍历的点和其中优先遍历的点。

点云聚类——具体实现

深蓝学院 shenlanxueyuan.com

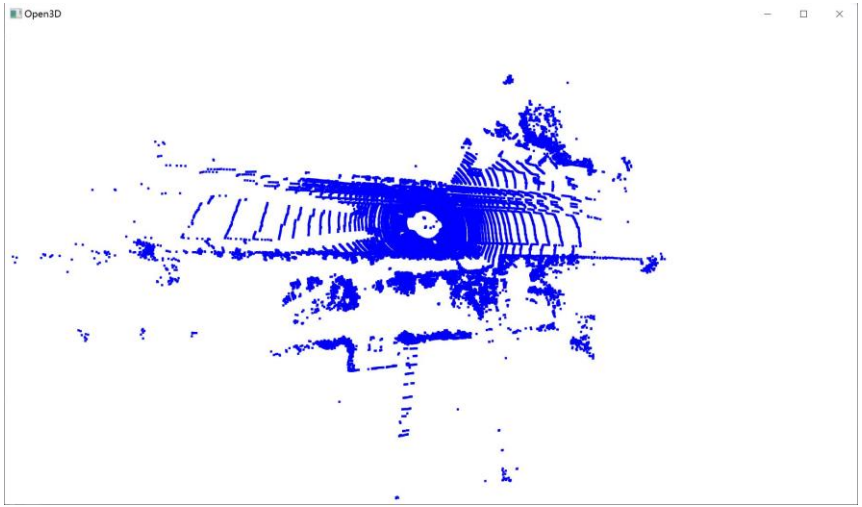
- 具体实现

```
def fit(self,data):
    # data:点云输入，N*3
    N = data.shape[0]
    # 修改最大递归次数，否则很容易超过最大递归次数(3000)
    sys.setrecursionlimit(N)
    cls = -1*np.ones(N)
    access = np.zeros(N)
    #总搜索队列
    search_index=list(range(N))
    #优先搜索队列
    Priority_search=[]
    #当前label
    label=-1
    #建立kdtree用于邻近搜索
    KD_root = kdtree.kdtree_construction(data, leaf_size=self.leaf_size)
    while search_index:
        if not Priority_search:
            ind=search_index.pop()
            #修改访问状态
            if access[ind]==1:
                continue
            else:
                access[ind]=1
            #kdtree邻近搜索
            query = data[ind, :]
            radius_result = result_set.RadiusNNResultSet(self.radius)
            kdtree.kdtree_radius_search(KD_root, data, radius_result, query)
            nn_indices = [radius_result.dist_index_list[jj].index for jj in range(radius_result.size())]
            #判断是否为核心点
            if len(nn_indices)-1 < self.Min_Pts:
                cls[ind] = -1
                continue
            else:
                label+=1
                cls[ind] = label
                Priority_search.extend(nn_indices)
        else:
            ind=Priority_search.pop()
            # 修改访问状态
            if access[ind] == 1:
                continue
            else:
                access[ind] = 1
                cls[ind]=label
            # kdtree邻近搜索
            query = data[ind, :]
            radius_result = result_set.RadiusNNResultSet(self.radius)
            kdtree.kdtree_radius_search(KD_root, data, radius_result, query)
            nn_indices = [radius_result.dist_index_list[jj].index for jj in range(radius_result.size())]
            # 判断是否为核心点
            if len(nn_indices) - 1 < self.Min_Pts:
                continue
            else:
                Priority_search.extend(nn_indices)
                Priority_search=list(np.unique(Priority_search))
    self.cls = cls
    self.access = access
```

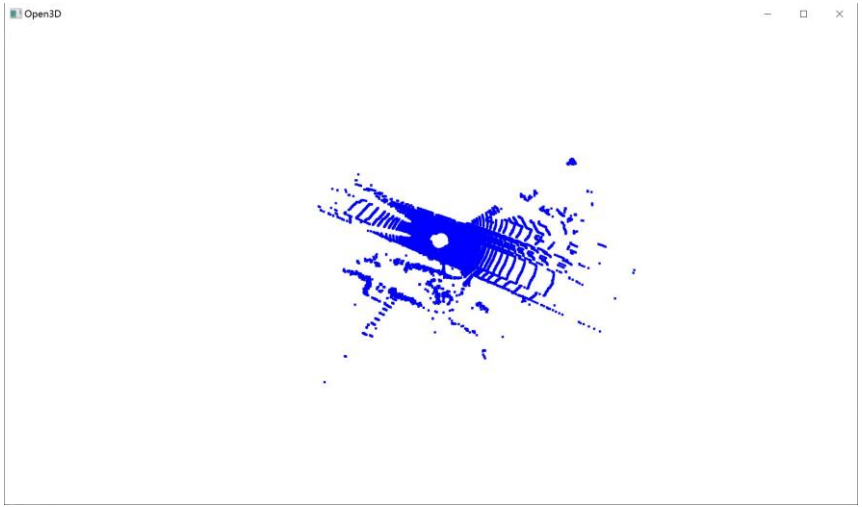
作业结果

深蓝学院
shenlanxueyuan.com

- 地面分割：原始点云（13W+个点）

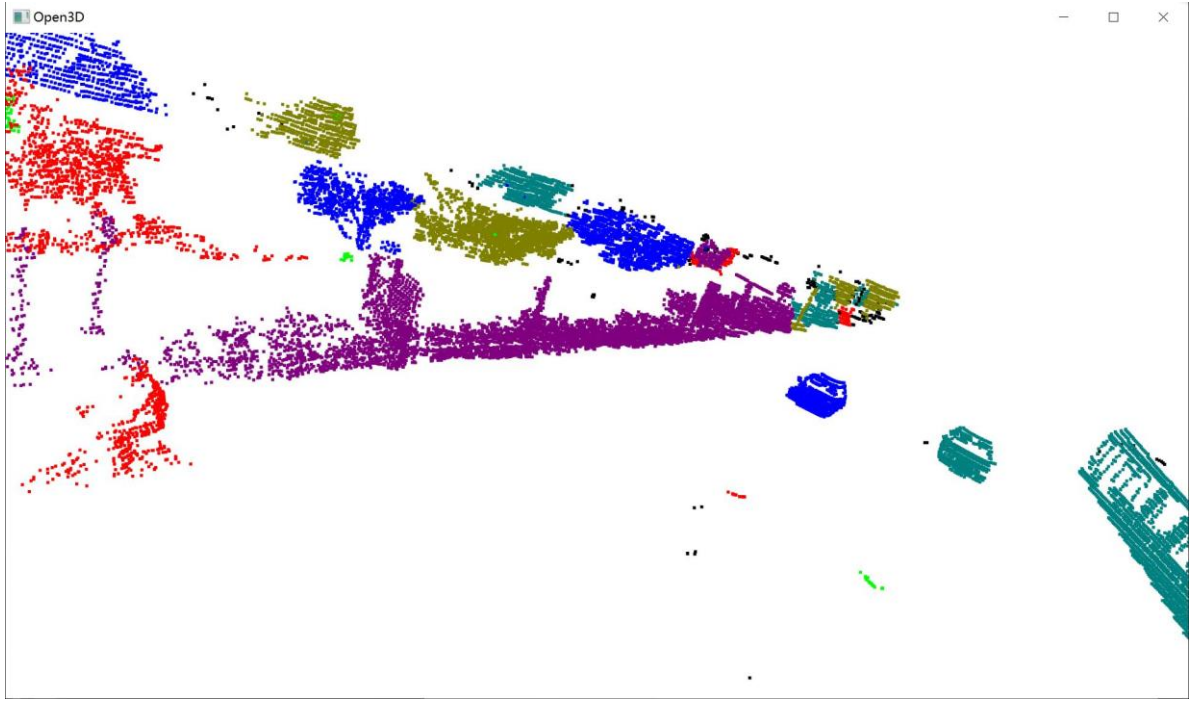


- 地面分割：地面点云（11W+个点）



作业结果

- 聚类结果：
- 比较显著的物体有骑行者，汽车，树木等等，说明聚类效果还不错。



在线问答

深蓝学院
shenlanxueyuan.com

Q&A

深蓝学院
shenlanxueyuan.com

感谢各位聆听！

Thanks for Listening