

三维点云第一次作业

一. 实现PCA分析和法向量计算


实现步骤:

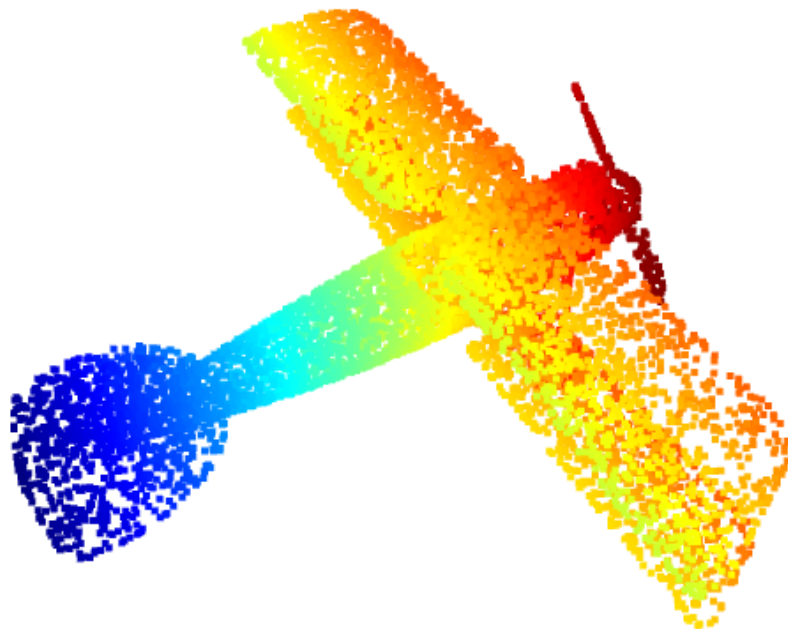
1. 加载点云文件并且显示

```
import open3d
import numpy as np

init_points = np.loadtxt('./data/airplane_0001.txt',
delimiter=',').astype(np.float32)[: , 0:3] # 读取点云数据
pointcloud = open3d.PointCloud() # 创建点云对象
pointcloud.points = open3d.utility.Vector3dVector(init_points) # 添加点云
open3d.visualization.draw_geometries([pointcloud]) # 可视化点云
```

可视化点云数据如下:

 init_pointcloud



2. 点云PCA分析

- 点云去中心化
- 计算协方差矩阵得到特征值和特征向量


```
def PCA(data, correlation=False, sort=True):

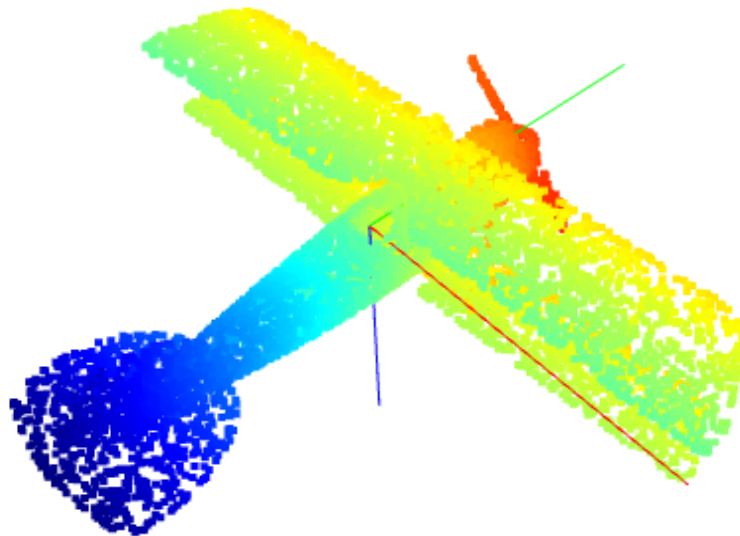
    # 去中心化
    data_mean = data - np.mean(data,axis=0) # n*3

    # 计算协方差矩阵
    eigenvalues, eigenvectors = np.linalg.eig(np.dot(data_mean.T,data_mean)) #
    (3,n)*(n,3)

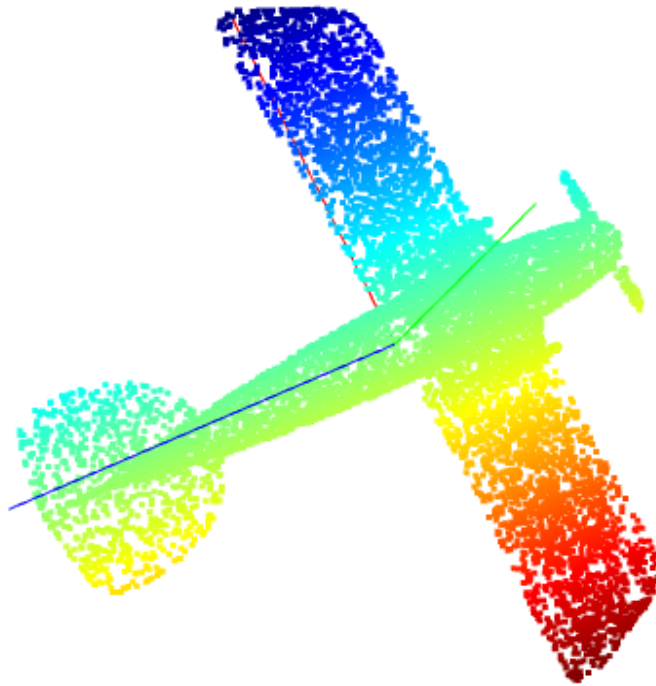
    if sort:
        sort = eigenvalues.argsort()[::-1]
        eigenvalues = eigenvalues[sort]
        eigenvectors = eigenvectors[:, sort]
    return eigenvalues, eigenvectors
```

显示PCA如下，其中红色是最大的特征向量方向，绿色次之，蓝色最小

 show_pca



将原始点云投影到主成分方向如下



3. 法向量估计

- 使用KDTree查找每个点邻域内的所有点
- 根据PCA计算每个点的法向量(特征值最小的向量)
- 显示法向量

```
pointcloud_tree = open3d.geometry.KDTreeFlann(pointcloud) # 创建一个KDTree对象
points = np.array(pointcloud.points)
normals = []

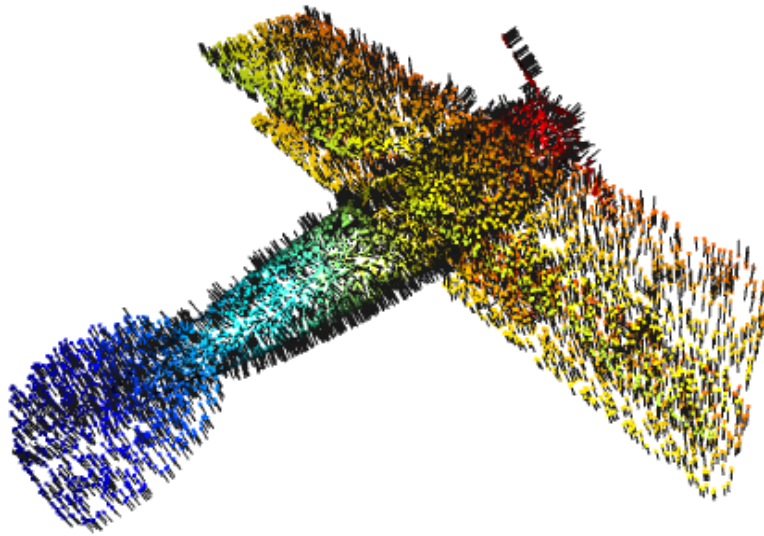
search_radius = 0.1 # 领域半径
search_num = 100 # 球域内最大点的数量
start = time.time()

for i in range(points.shape[0]):
    _, index, _ = pointcloud_tree.search_hybrid_vector_3d(points[i,:],
search_radius, search_num)
    set = points[index,:]
    _, set_v = PCA(set)
    normals.append(set_v[:, 2])

print('use time:',time.time()-start)
normals = np.array(normals, dtype=np.float64)

pointcloud.normals = open3d.utility.Vector3dVector(normals)
open3d.visualization.draw_geometries([pointcloud])
```

显示法向量如下



二. 实现体素滤波

1. 体素内随机选择点

主要代码:

```
def random_voxel_filter(point_cloud, leaf_size):

    x_max, y_max, z_max = np.max(point_cloud, axis=0)
    x_min, y_min, z_min = np.min(point_cloud, axis=0)

    D_x = np.floor((x_max - x_min) / leaf_size)
    D_y = np.floor((y_max - y_min) / leaf_size)
    D_z = np.floor((z_max - z_min) / leaf_size)

    # 计算每个点所在的voxel
    h = np.floor((point_cloud - np.array([x_min, y_min, z_min])) / leaf_size)

    # base formula calc
    h = h[:, 0] + h[:, 1] * D_x + h[:, 2] * D_x * D_y

    # 从小到大对h排序, 输出索引
    index = np.argsort(h).tolist()
    h = h[index].tolist()

    # 构建dict保存key和value
    dict = {}
    for i, j in zip(h, index):
```

```

        if i not in dict.keys():
            dict[i] = j
        else:
            if isinstance(dict[i], list) == False:
                dict[i] = [dict[i]]
                dict[i].append(j)
            else:
                dict[i].append(j)

filtered_points_index = []

# 随机选择的index
for item in dict.items():
    if isinstance(item[1], list) == True:
        filtered_points_index.append(random.choice(item[1]))
    else:
        filtered_points_index.append(item[1])

filtered_points = point_cloud[filtered_points_index]
filtered_points = np.array(filtered_points, dtype=np.float64) # numpy to
array

return filtered_points

```

2. 体素内取平均值

主要代码:

```

def centroid_voxel_filter(point_cloud, leaf_size):

    x_max, y_max, z_max = np.max(point_cloud, axis=0)
    x_min, y_min, z_min = np.min(point_cloud, axis=0)

    D_x = np.floor((x_max - x_min) / leaf_size)
    D_y = np.floor((y_max - y_min) / leaf_size)
    D_z = np.floor((z_max - z_min) / leaf_size)

    # 计算每个点所在的voxel
    h = np.floor((point_cloud - np.array([x_min, y_min, z_min])) / leaf_size)

    # base formula calc
    h = h[:, 0] + h[:, 1] * D_x + h[:, 2] * D_x * D_y

    # 从小到大对h排序, 输出索引
    index = np.argsort(h).tolist()
    h = h[index].tolist()

    # 构建dict保存key和value, 一个h对应多个index
    dict = {}
    for i, j in zip(h, index):
        if i not in dict.keys():
            dict[i] = j
        else:
            if isinstance(dict[i], list) == False:
                dict[i] = [dict[i]]
                dict[i].append(j)
            else:

```

```
dict[i].append(j)

filtered_points = []

# 取voxel中心点
for item in dict.items():
    if isinstance(item[1], list) == True:
        avg_points_ = np.mean(point_cloud[item[1]], axis=0)
        filtered_points.append(avg_points_.tolist())
    else:
        filtered_points.append(point_cloud[item[1]])

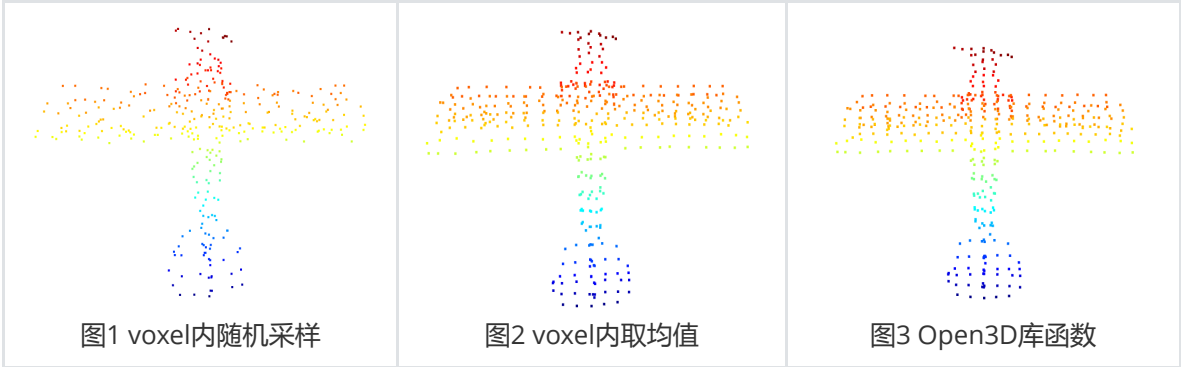
return filtered_points
```

3. voxel的不同分辨率结果对比

为了验证算法的准确性,同时测试实时性,利用Open3D自带的体素采样库函数,进行对比

3.1 voxel分辨率为0.1时

- 结果比较



- 运行时间比较

方法	随机采样	取平均值	库函数
运行时间 (s)	0.011934280395507812	0.01797652244567871	0.001009225845336914

3.2 voxel分辨率为0.01时

- 结果比较



- 运行时间比较

方法	随机采样	取平均值	库函数
运行时间 (s)	0.013996601104736328	0.03388857841491699	0.004175901412963867

3.3 结果分析

通过上图展示的结果，可以发现,自己编写程序实现的两种方法都实现了体素降采样，但是明显第二种方法的效果更好，接近于库函数的实现效果，但是两种方法的实时性和库函数相比还有较大的差距，仍然有优化代码，提高的空间。