深蓝学院
shenlanxueyuan.com

点云作业第四讲——聚类&模型拟合

主讲人 陆一帆

算法概述

●RANSAC地面去除

前提：一般情况下，在车辆采集的点云数据中，拥有点云数量最多的那个平面一定是地面。

算法流程：

初始化相关参数
while iter < 最大循环次数：
1. 随机取三个点确定一个平面并计算平面法向量
2. 根据平面法向量求其他点到该平面的距离并统计投票

取投票最多的平面为最终的地面
将距离地面较近的点云删除

算法概述

深蓝学院 shenlanxueyuan.com

●RANSAC地面去除

初始化相关参数

```
outlier_p = 0.7
N = 70
iter_num = 0
tau = 0.05
```

外点占比，第一次跑的时候瞎猜了一个值。求出地面后知道删掉了30%的点，后面就改成0.7了。这个值不影响地面拟合精度，只影响算法速度，设置的好可以提前退出循环。

最多循环70次，根据课堂上PPT中给出的经验表格设置的。

距离平面小于该值则记一次投票。
在拟合阶段，这个值设小一点，可以让拟合精度高一些。值越小表示稍微偏离这个平面，就不算内点了，所以最终得票最多的那个平面一定非常贴合地面；但是也不能特别小，因为地面也不是绝对平面。

算法概述

深蓝学院 shenlanxueyuan.com

- RANSAC地面去除

三点确定平面

```
# 取三个随机点
idxs = [np.random.randint(0, data.shape[0]) for _ in range(3)]
pts = data[idxs]
# 计算平面法向量
p0p1 = pts[1] - pts[0]
p0p2 = pts[2] - pts[0]
# 三点共线
diff = p0p2 - (p0p2[0] / p0p1[0]) * p0p1
if np.linalg.norm(diff) < 0.001:
    continue
nor_vec = np.cross(p0p2, p0p1)
norm = np.linalg.norm(nor_vec)
nor_vec /= norm      You, a day ago • hw5
```

从一个点指向其他两个点，可以形成两个三维向量

三点共线不能确定一个平面，舍弃

两个向量的叉乘垂直于这两个向量，也就是这两个向量组成平面的法向量。最后别忘了归一化。

算法概述

深蓝学院 shenlanxueyuan.com

●RANSAC地面去除

根据法向量确定距离后投票

```
# 根据点到平面的距离投票
inlier_vote = 0
for i in range(data.shape[0]):
    pt = data[i]
    dist = math.fabs(np.matmul(nor_vec, (pt - pts[0]).T) / np.linalg.norm(nor_vec))
    if dist < tau:
        inlier_vote += 1
if inlier_vote > max_inlier_vote:
    nor_vec_final = nor_vec
    max_inlier_vote = inlier_vote
    pt_final = pts[0]
if max_inlier_vote > (1 - outlier_p) * data.shape[0]:
    print("stop at[" + str(iter_num) + "].")
    break
```

根据平面的法向量和平面上任意一个点，求空间中一个点到该平面的距离。并确定是否投票。

记录下投票最高的平面的法向量和该平面上任意一点的坐标。

如果某个平面已经得到了足够多投票，提前退出循环。

算法概述

深蓝学院 shenlanxueyuan.com

●RANSAC地面去除

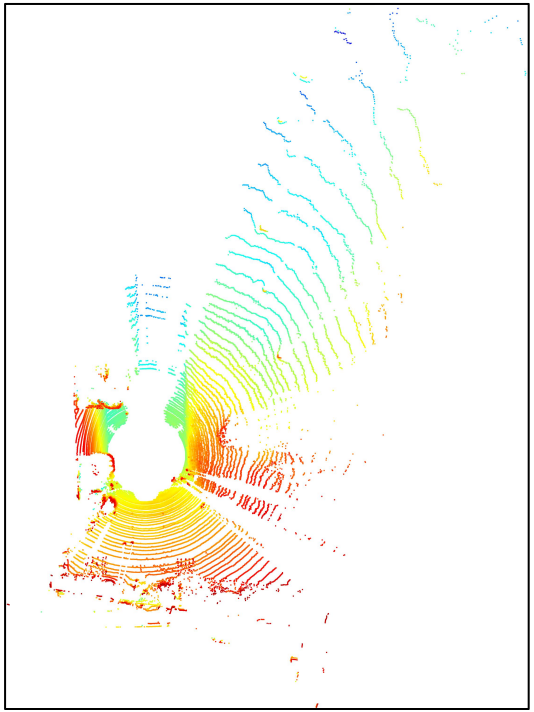
地面点云去除

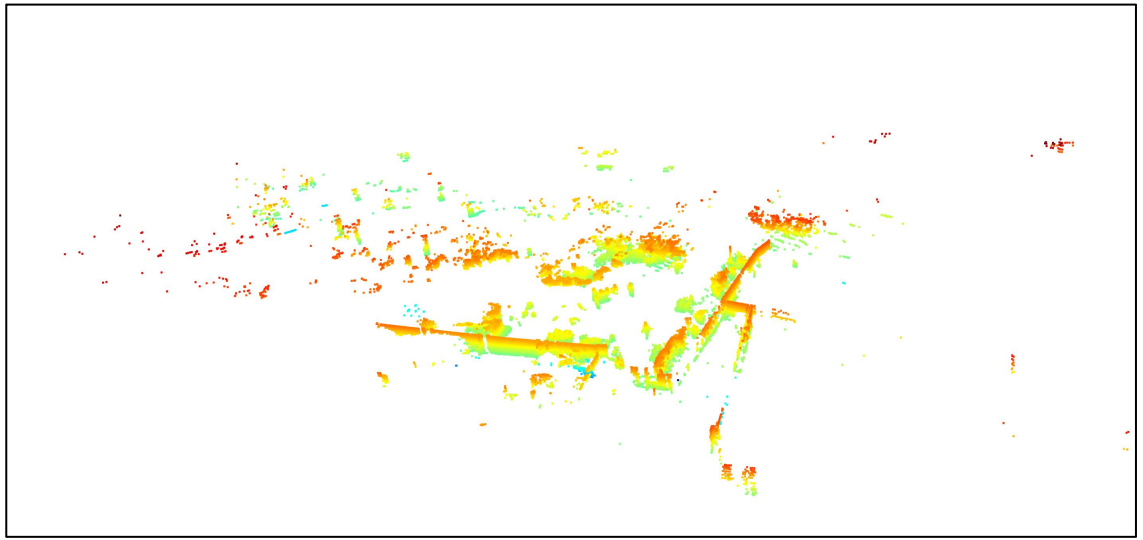
```
left_idx = []
rm_idx = []
for i in range(data.shape[0]):
    pt = data[i]
    dist = math.fabs(np.matmul(nor_vec_final, (pt - pt_final).T) / np.linalg.norm(nor_vec_final))
    if dist > 0.3:
        left_idx.append(i)
    else:
        rm_idx.append(i)
segmengted_cloud = data[left_idx]
rm_cloud = data[rm_idx]
```

取投票最高的平面为地面，重新计算所有点到地面的距离，距离<0.3则滤除。
注意，前面拟合的时候tau = 0.05，是为了拟合精度较高；这里tau = 0.3，是为了把地面滤除干净，因为地面不是绝对平面，也有高低起伏。但这样做，地面的砖块啥的也就被过滤了。为了能够精准过滤地面，又保留低矮障碍物点云，最好能够将空间栅格化，每个栅格独立做地面拟合，但我也没尝试，只是个思路。

算法概述

●RANSAC地面去除





左图：地面点云
右图：滤除掉地面点云后剩下的点云

算法概述

●空间点云分类

前提：无人驾驶领域，决策规划其实大部分还是在二维平面进行。感知模块输出的障碍物信息为一个棱柱状的凸包（俯视为polygon，前视侧视都是长方形）。因此，三维空间点云聚类可以直接压缩到二维图像上处理。

算法流程：

1. 将三维点云投影到地面所在的二维平面
2. 将二维点云构建为图像
3. 利用图像相关算法聚类
4. 根据图像上的聚类结果给三维空间的点云聚类。

注意：本方法只是一个补充，实现过程中借用了opencv的函数，学习的时候还是以课堂上讲解的dbscan等方法为主。

算法概述

深蓝学院 shenlanxueyuan.com

- 空间点云聚类

  三维点云二维化

```
w, v, mean_removed = PCA(data)
point_cloud_vector = v[:, :2]  # 主方向向量
low_dim_data = mean_removed * point_cloud_vector
```

合理的做法是根据上面得到的地面信息，将点云投影到地面进行二维化。

我这边直接调用第一课的PCA，投影到方差最大方向（其实也就是地面方向）。

算法概述

深蓝学院 shenlanxueyuan.com

●空间点云聚类

二维点云转图像

```python
# 获取图像的高和宽
x_min = float("inf")
x_max = -float("inf")
y_min = float("inf")
y_max = -float("inf")
for i in range(data.shape[0]):
    pt = data[i]
    x_min = min(x_min, pt[0, 0])
    x_max = max(x_max, pt[0, 0])
    y_min = min(y_min, pt[0, 1])
    y_max = max(y_max, pt[0, 1])
width = math.ceil((y_max - y_min) / 0.2)
height = math.ceil((x_max - x_min) / 0.2)
# 点云转图像
mat = np.zeros((height, width, 1), np.uint8)
for i in range(data.shape[0]):
    pt = data[i]
    x = pt[0, 0]
    y = pt[0, 1]
    p_x = math.floor((x - x_min) / 0.2)
    p_y = math.floor((y - y_min) / 0.2)
    mat[p_x, p_y, 0] = 255
# cv2.imwrite("pointclouds.png", mat)
```

根据二维点云的坐标范围确定图像尺寸

遍历每个点云，根据其二维坐标，将其画到图像中，图像分辨率为一个像素0.2米（这一步其实也有采样的用处，多个点云被采样成一个像素，数据量大幅下降。）

算法概述

深蓝学院 shenlanxueyuan.com

●空间点云聚类

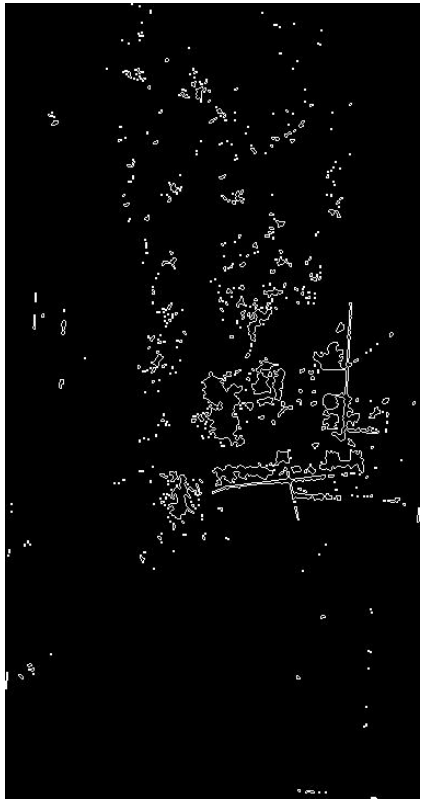
点云类别提取

```
kernel = np.ones((2, 2), np.uint8)
mat = cv2.dilate(mat, kernel)
h = cv2.findContours(mat, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
# 提取轮廓
contours = h[0]
background = np.zeros((mat.shape[0], mat.shape[1], 1), np.uint8)
cv2.drawContours(background, contours, -1, 255, 1)
```

首先用opencv的dilate函数对原始图像进行膨胀，kernel=(2,2)，意味着距离小于0.4米的点会连在一起。

然后用findContours函数找出图像中的所有轮廓，每一组轮廓就是一个类。

右图为所有轮廓。



算法概述

- 空间点云聚类

  点云分类

```
clusters_index = []
all_kinds = []
for i in range(data.shape[0]):
    pt = data[i]
    x = pt[0, 0]
    y = pt[0, 1]
    p_x = math.floor((x - x_min) / 0.2)
    p_y = math.floor((y - y_min) / 0.2)
    clas = 0
    final_clas = 0
    max_dist = -float('inf')
    find = False
    for contour in contours:
        # hull = cv2.convexHull(contour, returnPoints = False)
        dist = cv2.pointPolygonTest(contour, (p_y, p_x), True)
        if dist > 0.0:
            find = True
            clusters_index.append(clas)
            break
        elif dist > max_dist:
            max_dist = dist
            final_clas = clas
        clas += 1
    if not find:
        # print("pt is classifered to " + str(final_clas))
        clusters_index.append(final_clas)
```

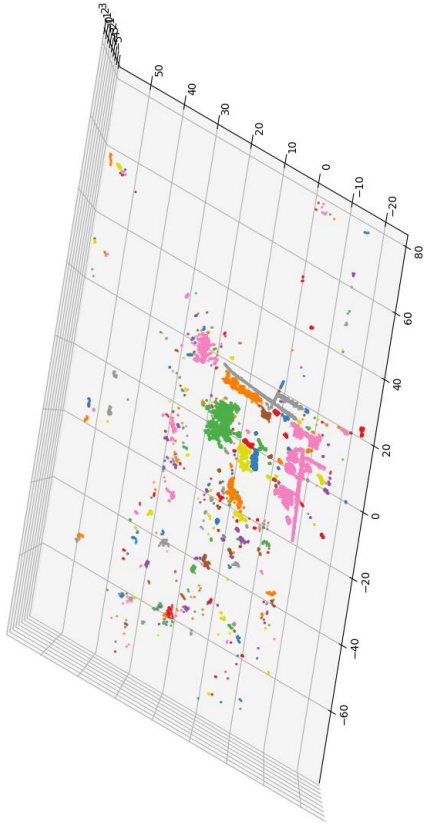
最后用pointPolygonTest函数（判断点是否在轮廓内）给每个点分类。

分类结果可视化如右图

可以看到其实就是把连在一起的点云分为一类，所以大致和dbscan效果差不多。

优势在于这样的聚类方法用C++实现的话速度很快，可以在10-20ms左右处理完。所以在硬件资源有限且实时性要求较高的场景下效果还不错。

另外在完成大作业的时候，可以在去除地面后，再把高于无人车的点云过滤掉（一般是树冠啥的）。再投影到图像上聚类，这样不会因为行人或车在树冠下导致被聚类在一起。



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