



北京航空航天大学
BEIHANG UNIVERSITY

机器人导航python实践(入门)

Lecture3-建图

北航 国新院 实验实践课
智能系统与人形机器人国际研究中心



教师： 欧 阳 老 师



邮 箱： ouyangkid@buaa.edu.cn



学 期： 2025年秋季

目录

Contents

01 课程内容安排

02 建图算法

03 目标轮廓识别

Part 1 | 课程内容安排

课程内容

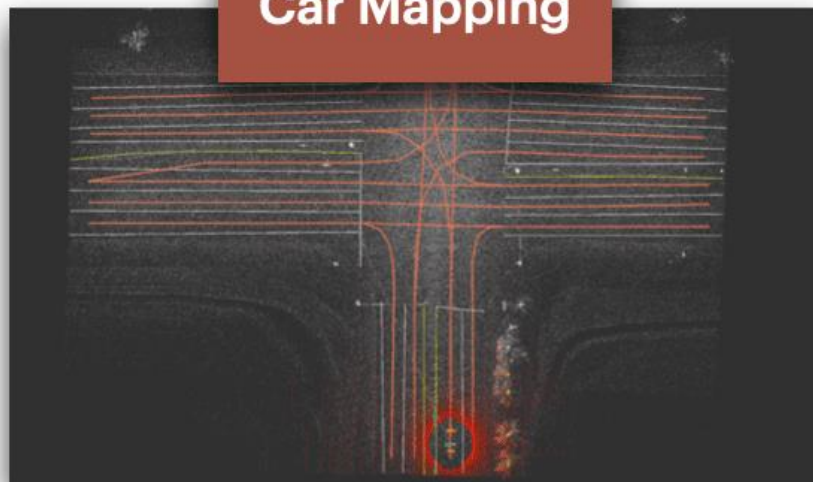
» 内容安排

机器人需要依赖环境感知传感器获取外部周围信息，识别可行驶区域、以及障碍物信息。常见的传感器包括模拟人视觉的相机、双目立体视觉、激光雷达和毫米波等。

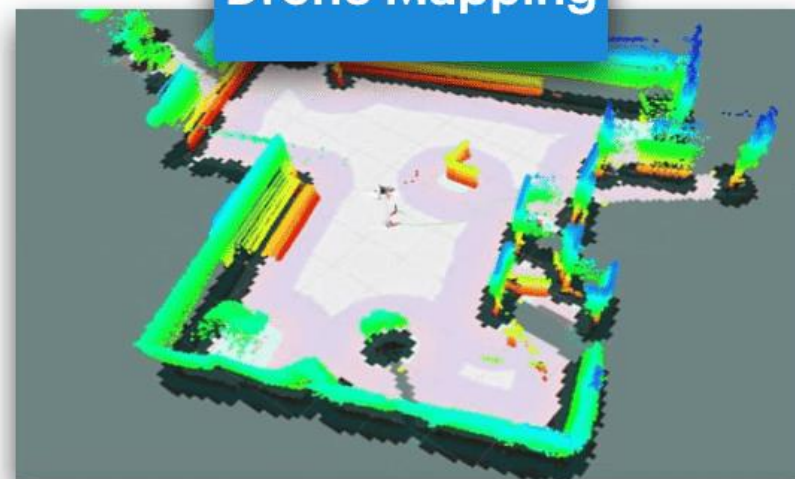
Robot Mapping



Car Mapping



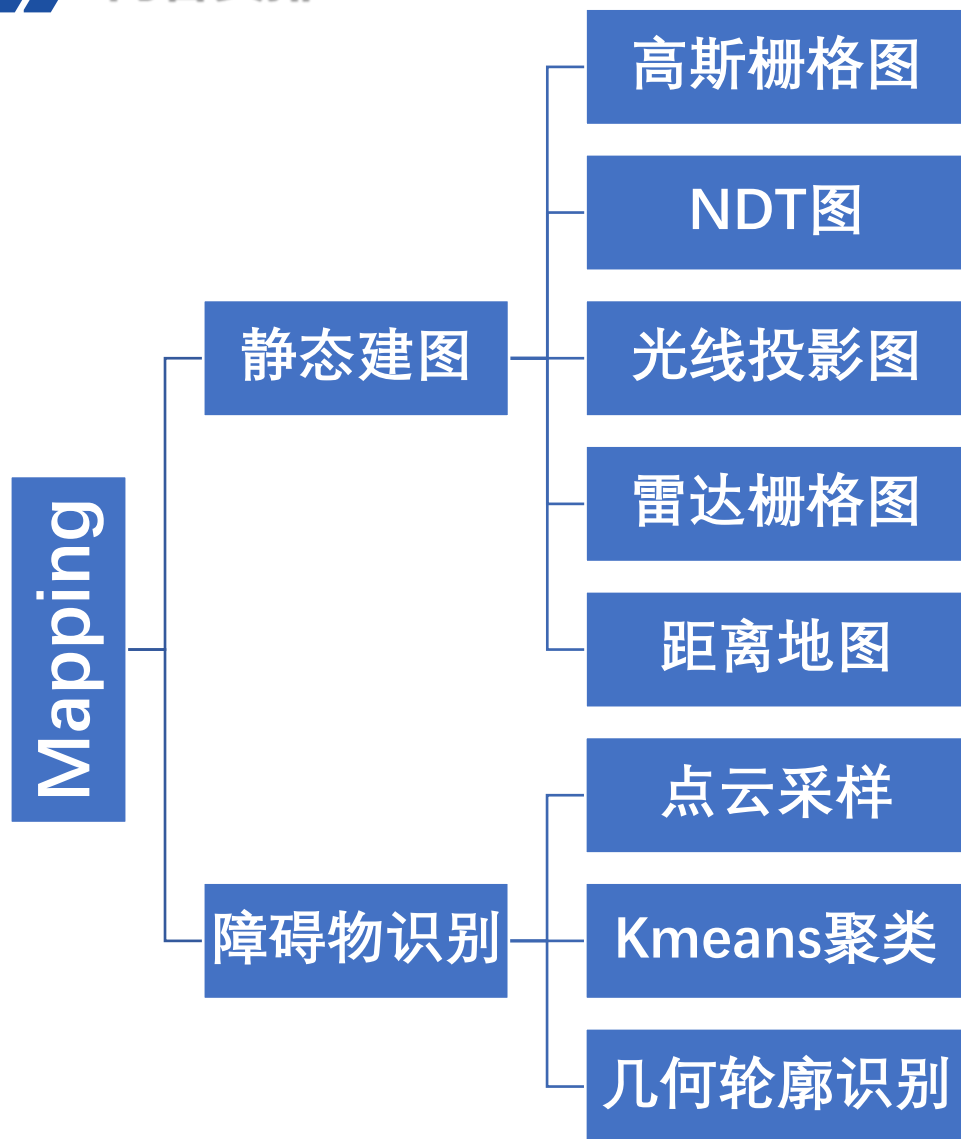
Drone Mapping



Part 1 | 课程内容安排

课程内容

» 内容安排



目录

Contents

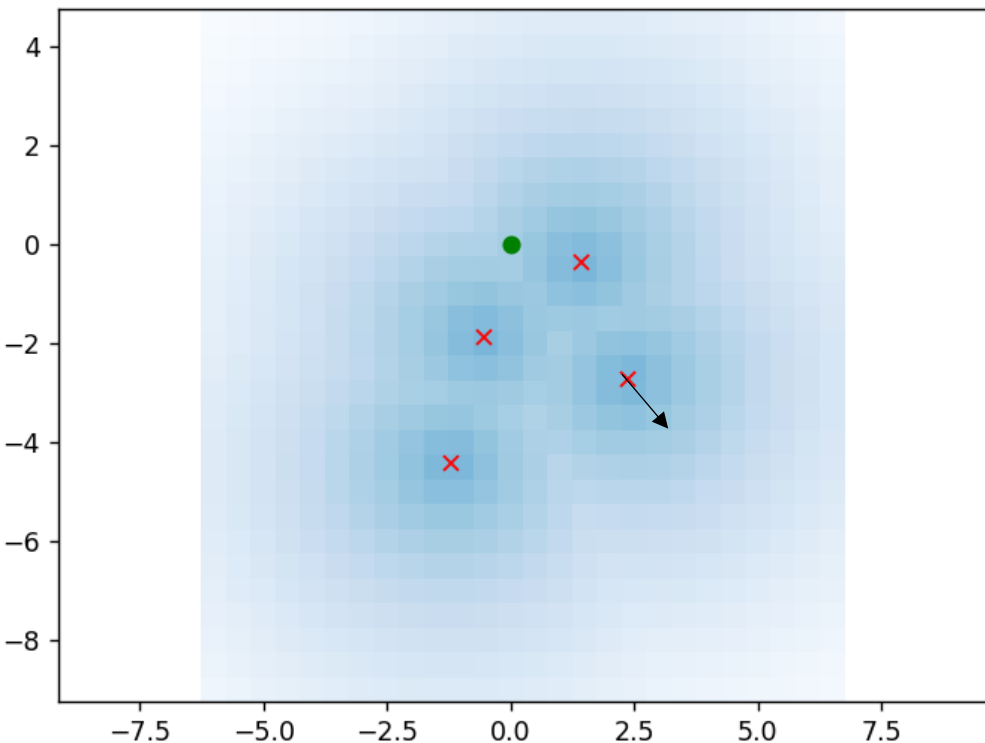
01 课程内容安排

02 建图算法

03 目标轮廓识别

Part 2 | Mapping (static)

» 建图算法 (静态) --高斯栅格图

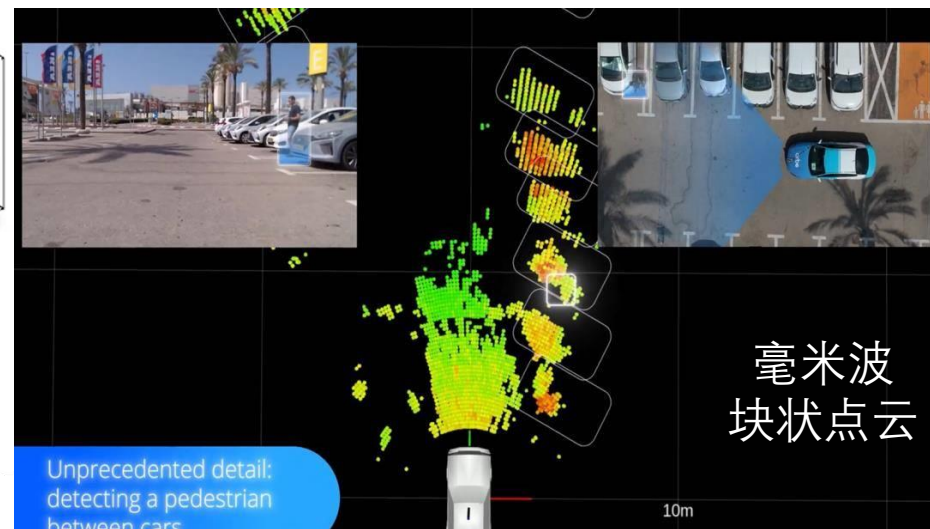
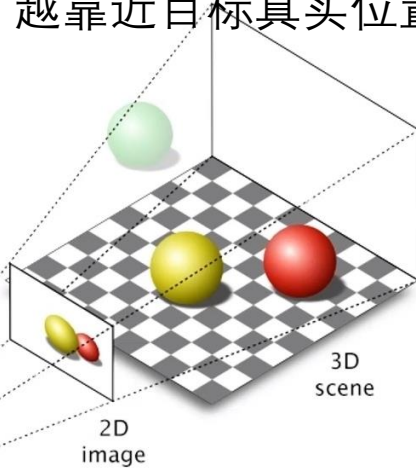


- 绿色点：传感器位置。
- 红色x：障碍物坐标。
- 蓝色热力图：估计的障碍物在栅格地图的高斯占位

高斯栅格图 (Gaussian Grid Map) 是一种基于栅格的环境表示方法，结合了高斯分布 (正态分布) 的特性来描述空间中障碍物或目标的概率分布。

理想情况下，传感器能够获得目标精准信息，映射到栅格地图是具体的某个栅格。

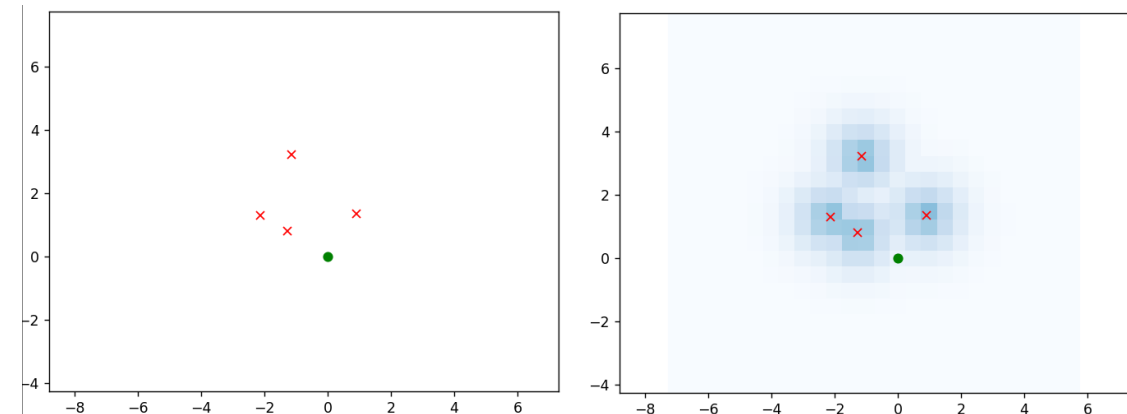
实际情况下，比如相机透视坐标无法实现精准测距，映射到栅格地图上，越靠近目标真实位置的栅格概率越高，随距离下降。



Part 2 | Mapping (static)

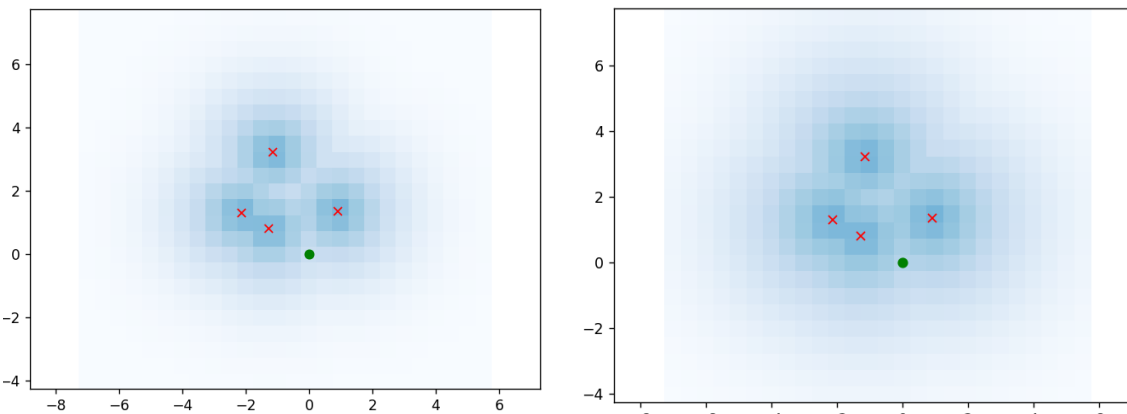
建图算法 (静态) --高斯栅格图

- 定义栅格地图分辨率0.5m，仿真模拟场景[-10, 10]m
- 传感器测量高斯分布的标准差(STD)=5m，理解为传感器性能，性能越好，误差越小，在栅格图上的干扰越小。



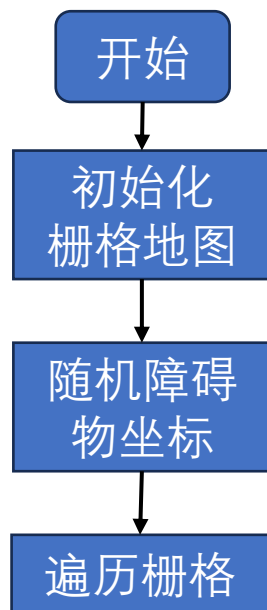
STD=0

STD=1



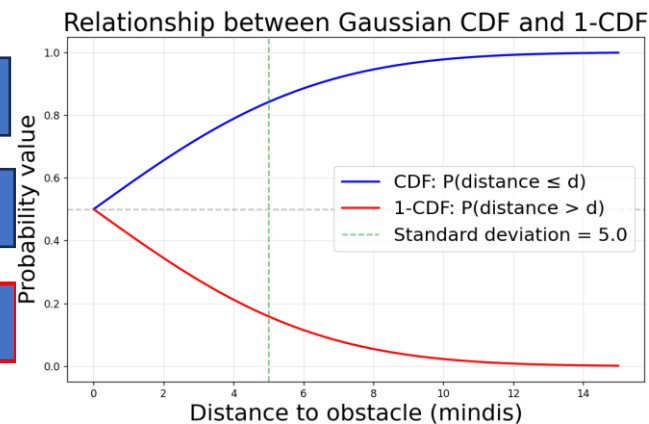
STD=2

STD=3



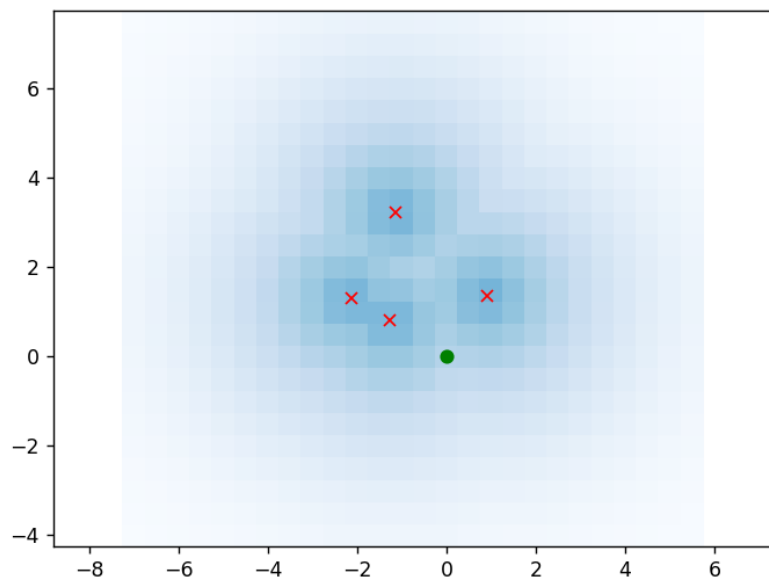
通过调整 高斯分布的标准差，能够模拟不同的感知占位效果。

```
def generate_gaussian_grid_map(ox, oy, xyreso, std):  
    minx, miny, maxx, maxy, xw, yw = calc_grid_map_config(ox, oy, xyreso)  
    gmap = [[0.0 for i in range(yw)] for i in range(xw)]  
    for ix in range(xw):  
        for iy in range(yw):  
            x = ix * xyreso + minx  
            y = iy * xyreso + miny  
            # Search minimum distance  
            mindis = float("inf")  
            for (ioy, ioy) in zip(ox, oy):  
                d = math.hypot(ioy - x, ioy - y)  
                if mindis >= d:  
                    mindis = d  
            pdf = (1.0 - norm.cdf(mindis, 0.0, std))  
            gmap[ix][iy] = pdf  
    return gmap, minx, maxx, miny, maxy
```

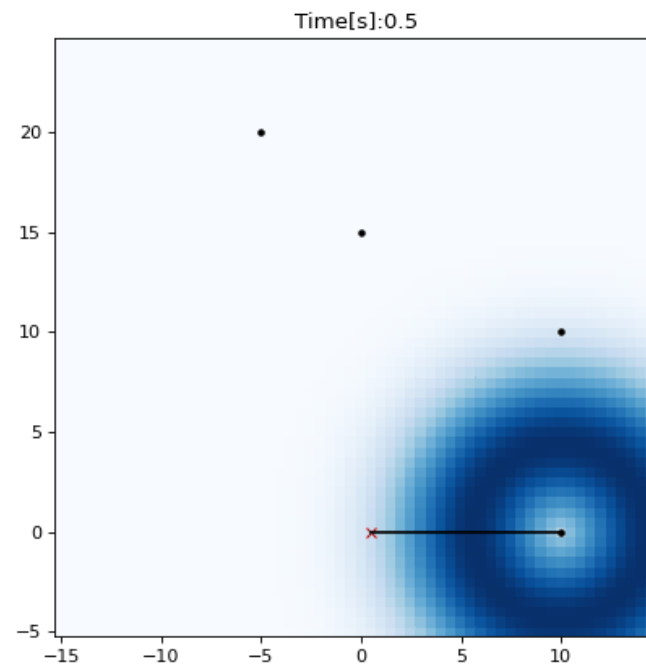


Part 2 | Mapping (static)

» 建图算法（静态） -- 高斯栅格图



VS.







高斯栅格地图基于高斯分布对目标障碍物进行场景表征

直方图滤波定位是利用测距信息的不精准，对自身位置进行建模

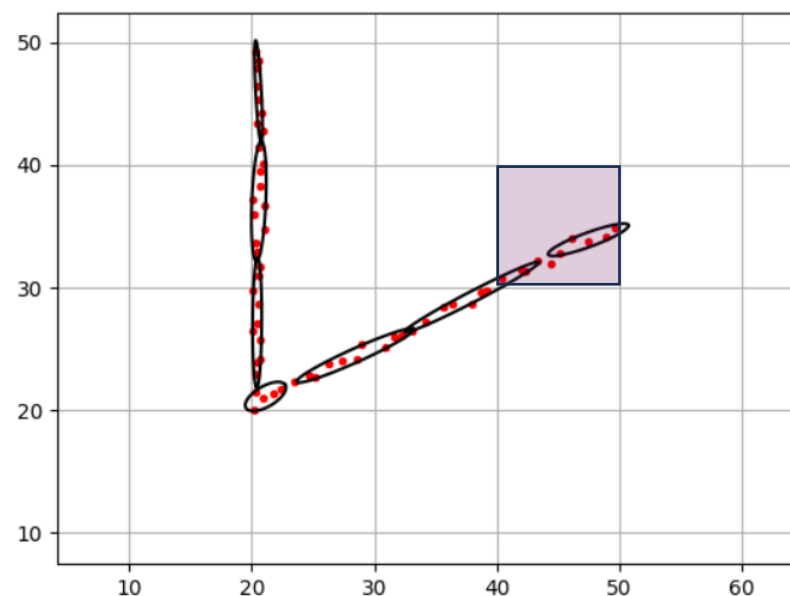
Part 2 | Mapping (static)

建图算法 (静态) --NDT map

Normal Distribution Transform (NDT) Map, 正态分布变换地图, 利用高斯分布拟合栅格内的点云数据描述环境特征。针对**激光雷达**稀疏点云数据。即便激光点云比图像像素稀疏, 但是在空间采样时依然具有较大的规模。特别的, 将三维点云投影到BEV鸟瞰图上时, 同一目标在不同高度上的点云采样, 以及点云本身的抖动噪声会导致数据规模巨大。通过NDT map有利于极大的加速计算, 如: NDT配准算法。

				
Range	Up to 120m	Up to 100m	100m	100m
Range Accuracy	Up to ± 2 cm (Typical) ⁴	Up to ± 2 cm (Typical) ¹	Up to ± 3 cm (Typical) ¹	Up to ± 3 cm (Typical) ¹
# of Lines	64	32	16	16
Horizontal FoV	360°	360°	360°	360°
Vertical FoV	26.9°	41.33°	30°	30°
Horizontal Resolution	0.08° - 0.35°	0.08° - 0.33°	0.1° - 0.4°	0.1° - 0.4°
Vertical Resolution	0.4°	1.33°	2.0°	2.0°
Points Per Second (Single Return Mode)	~ 1,300,000	~ 695,000	~ 300,000	~ 300,000
Points Per Second (Dual Return mode)	~ 2,200,000 ⁵	~ 1,390,000	~ 600,000	~ 600,000
Refresh Rate	5-20 Hz	5-20 Hz	5-20 Hz	5-20 Hz

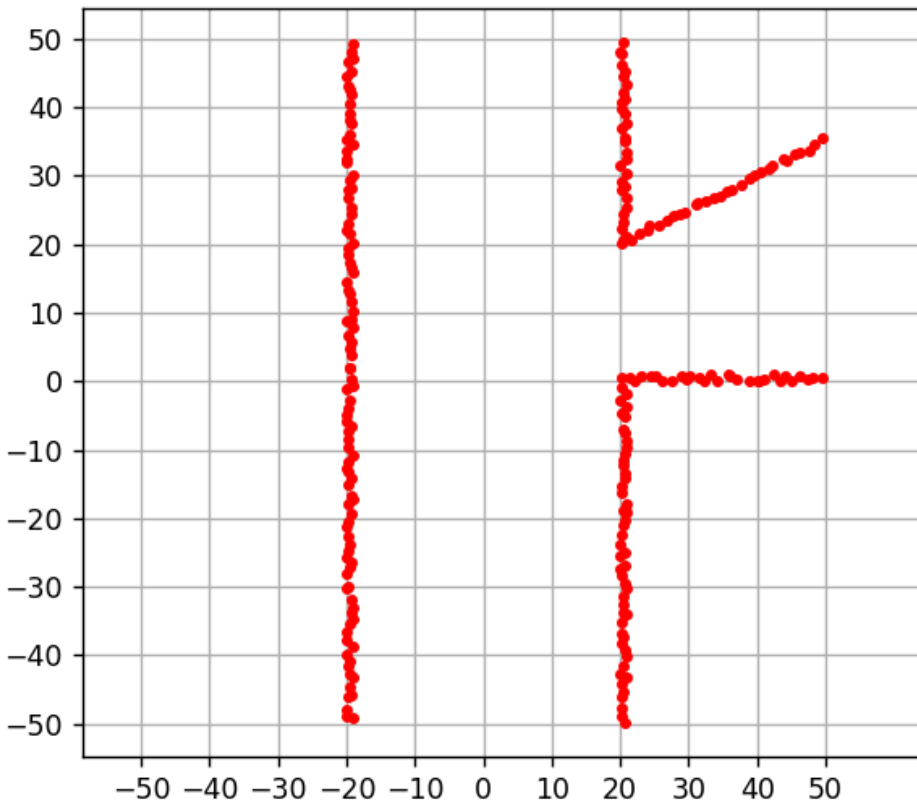
当单位栅格内的点数量大于1时, 如何对栅格进行描述, 也就是选取**何种特征**表示当前栅格。



Part 2 | Mapping (static)



建图算法 (静态) --NDT Map

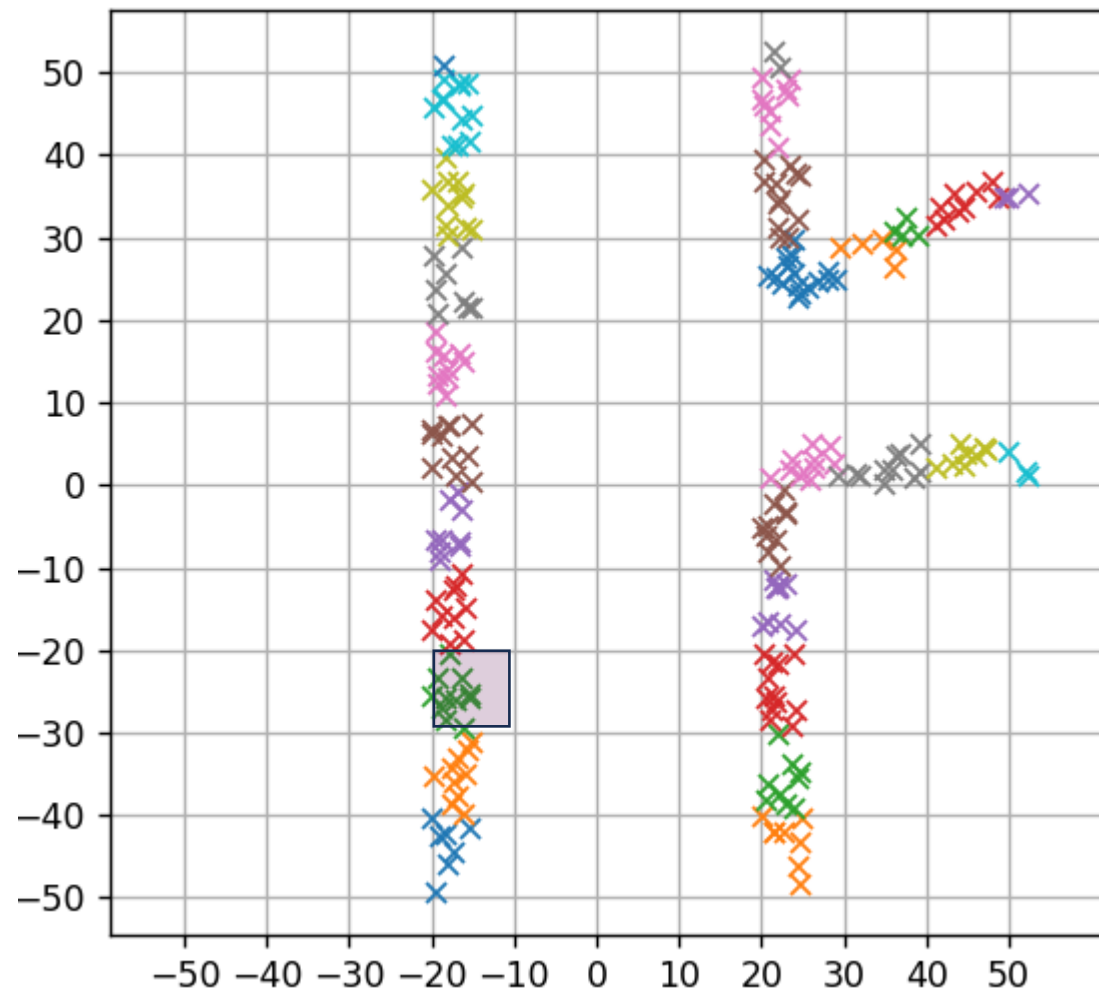
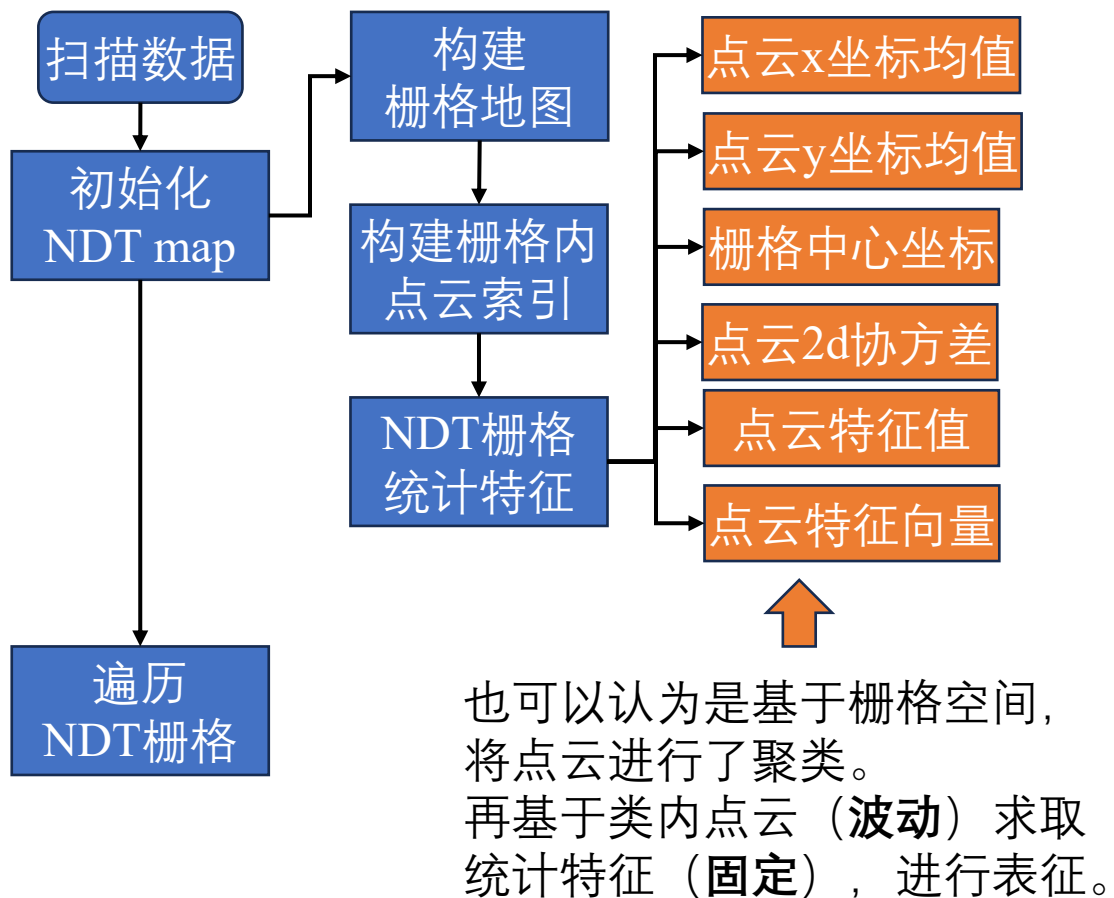


模拟单线激光雷达扫描的
三岔路口数据；
并对点云施加轻微扫描噪声。

```
def create_dummy_observation_data():  
    ox = []  
    oy = []  
    # left corridor  
    for y in range(-50, 50):  
        ox.append(-20.0)  
        oy.append(y)  
    # right corridor 1  
    for y in range(-50, 0):  
        ox.append(20.0)  
        oy.append(y)  
    # right corridor 2  
    for x in range(20, 50):  
        ox.append(x)  
        oy.append(0)  
    # right corridor 3  
    for x in range(20, 50):  
        ox.append(x)  
        oy.append(x/2.0+10)  
    # right corridor 4  
    for y in range(20, 50):  
        ox.append(20)  
        oy.append(y)  
    ox = np.array(ox)  
    oy = np.array(oy)  
    # Adding random noise  
    ox += np.random.rand(len(ox)) * 1.0  
    oy += np.random.rand(len(oy)) * 1.0  
    return ox, oy
```

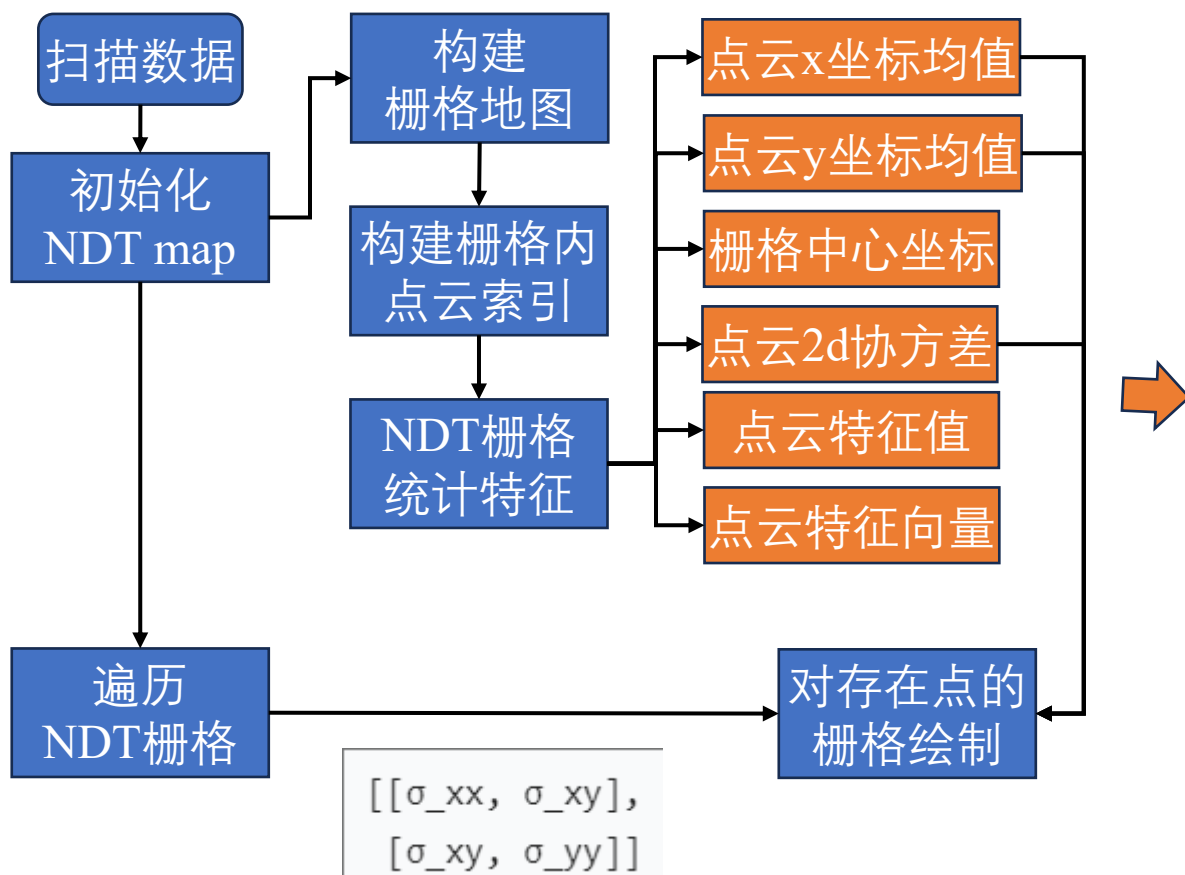
Part 2 | Mapping (static)

建图算法 (静态) --NDT Map



Part 2 | Mapping (static)

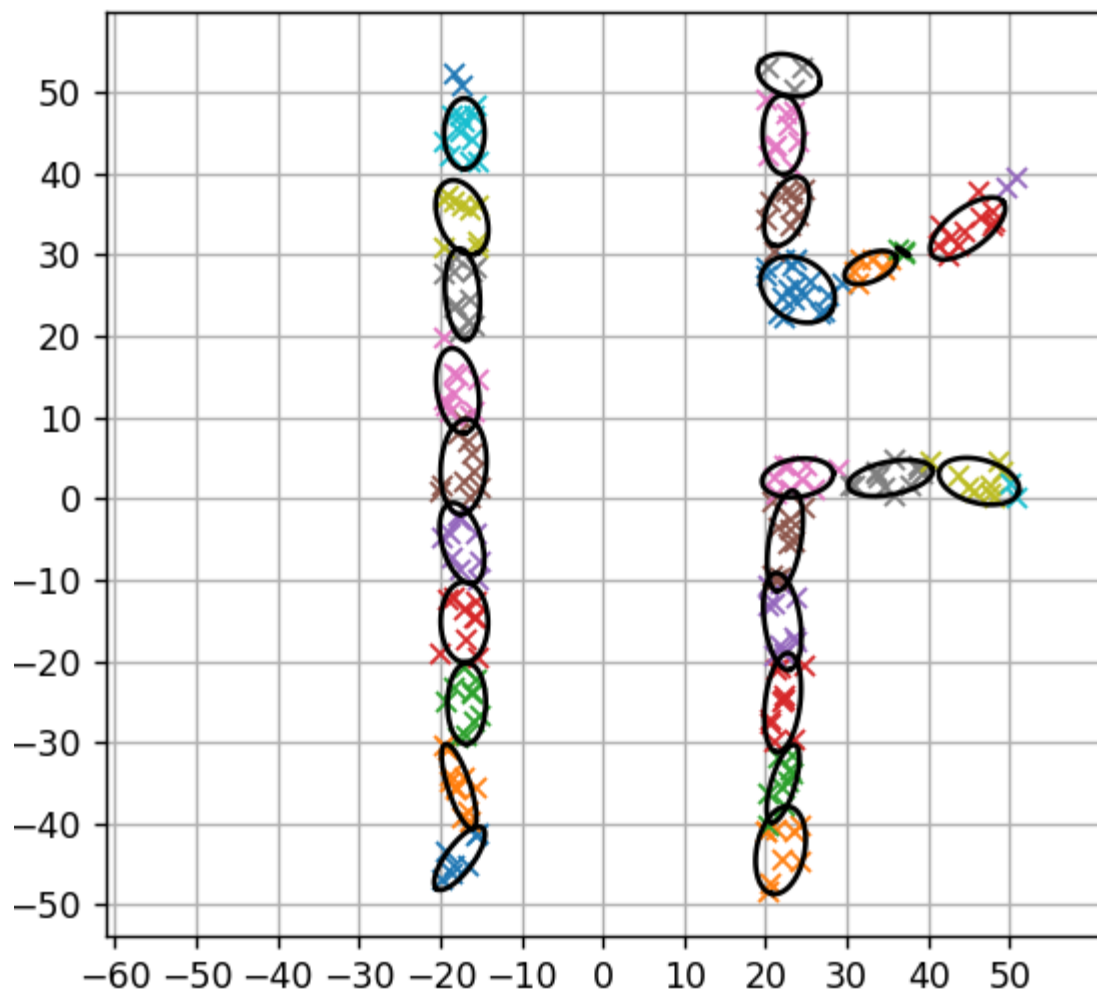
建图算法 (静态) --NDT Map



$$X \sim \mathcal{N}(\mu, \Sigma) \Rightarrow$$

$$X = \frac{1}{\sqrt{(2\pi)^2|\Sigma|}} \exp \left\{ -\frac{1}{2} {}^t(x - \mu) \Sigma^{-1} (x - \mu) \right\}$$

从一维 扩展到 二维 的概率密度。

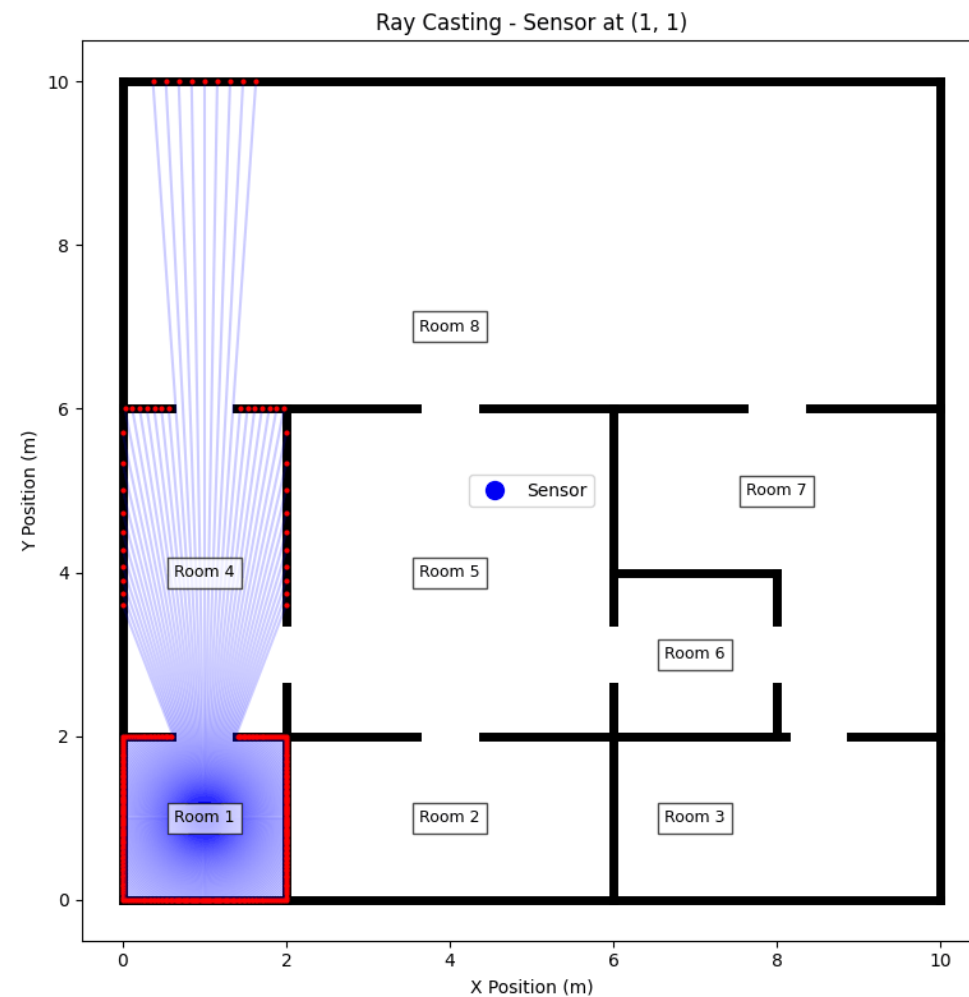
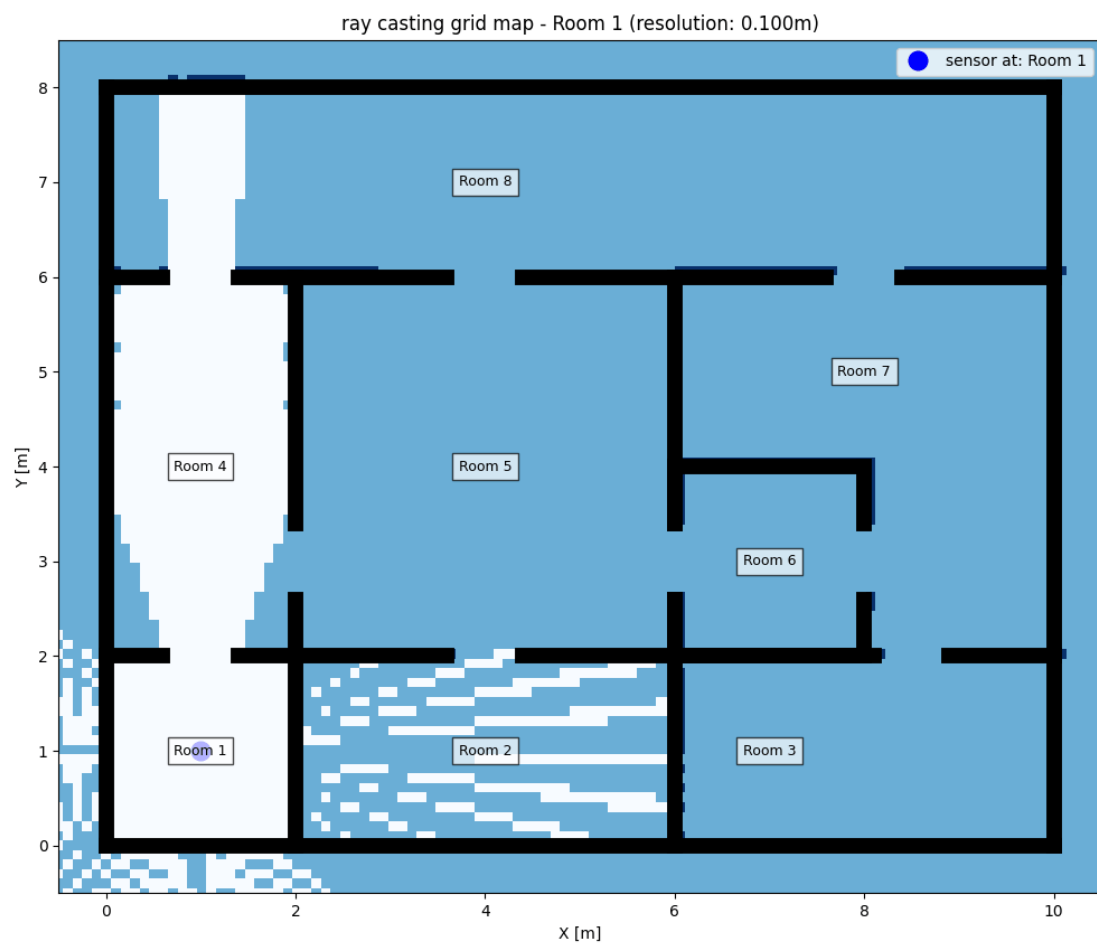


因为空间剖分，不同栅格内的点规模是动态的，如果以栅格为单位将难以描述单位栅格的属性。

当前的绘制虽然仅使用坐标均值和协方差矩阵，但特征值（椭圆长轴长短）和特征向量（椭圆长轴方向）实际包含其中。

Part 2 | Mapping (static)

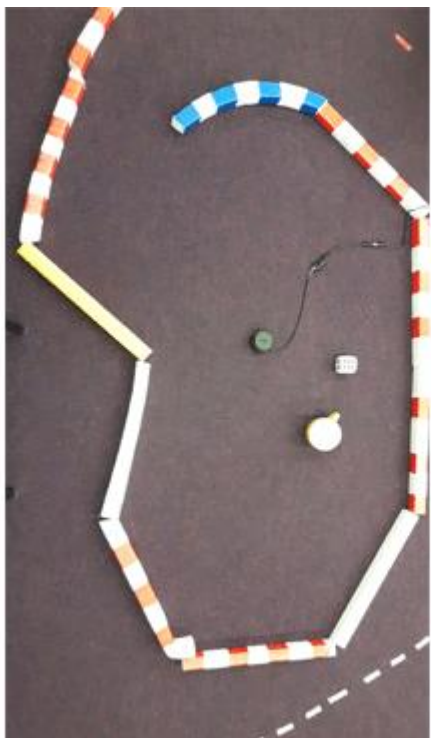
» 建图算法 (静态) --光线投影栅格地图(略)



Part 2 | Mapping (static)

» 建图算法 (静态) -- 雷达占位图

雷达占位图可以理解为分辨率较高的 栅格图。
由一线激光雷达数据生成。



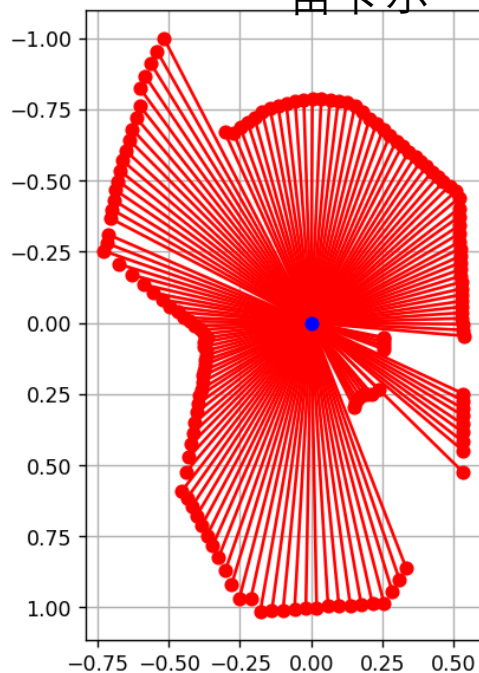
一线激光雷达数据: CSV文件

角度, 距离

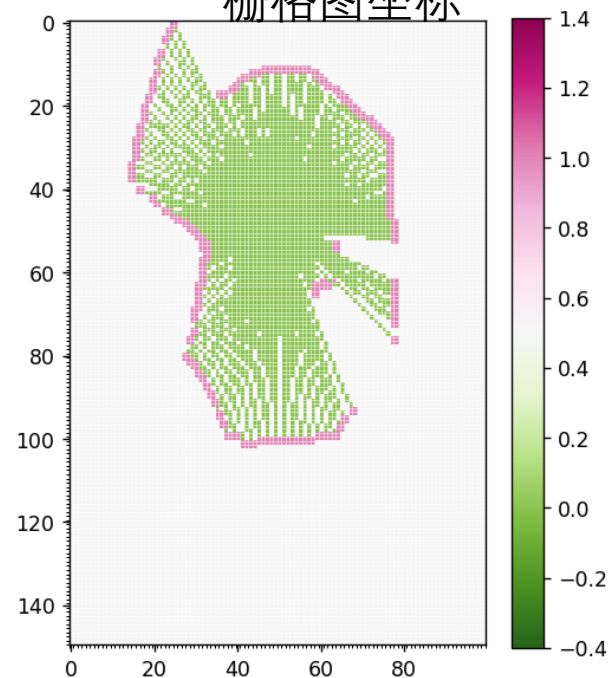
1	0.008450416037156572, 0.5335
2	0.046902201120156306, 0.5345
3	0.08508127850753233, 0.537
4	0.1979822644959155, 0.2605
5	0.31180075107071505, 0.0405

栅格分辨率=0.02 # 提升25倍, 适用于室内近距离环境。

笛卡尔

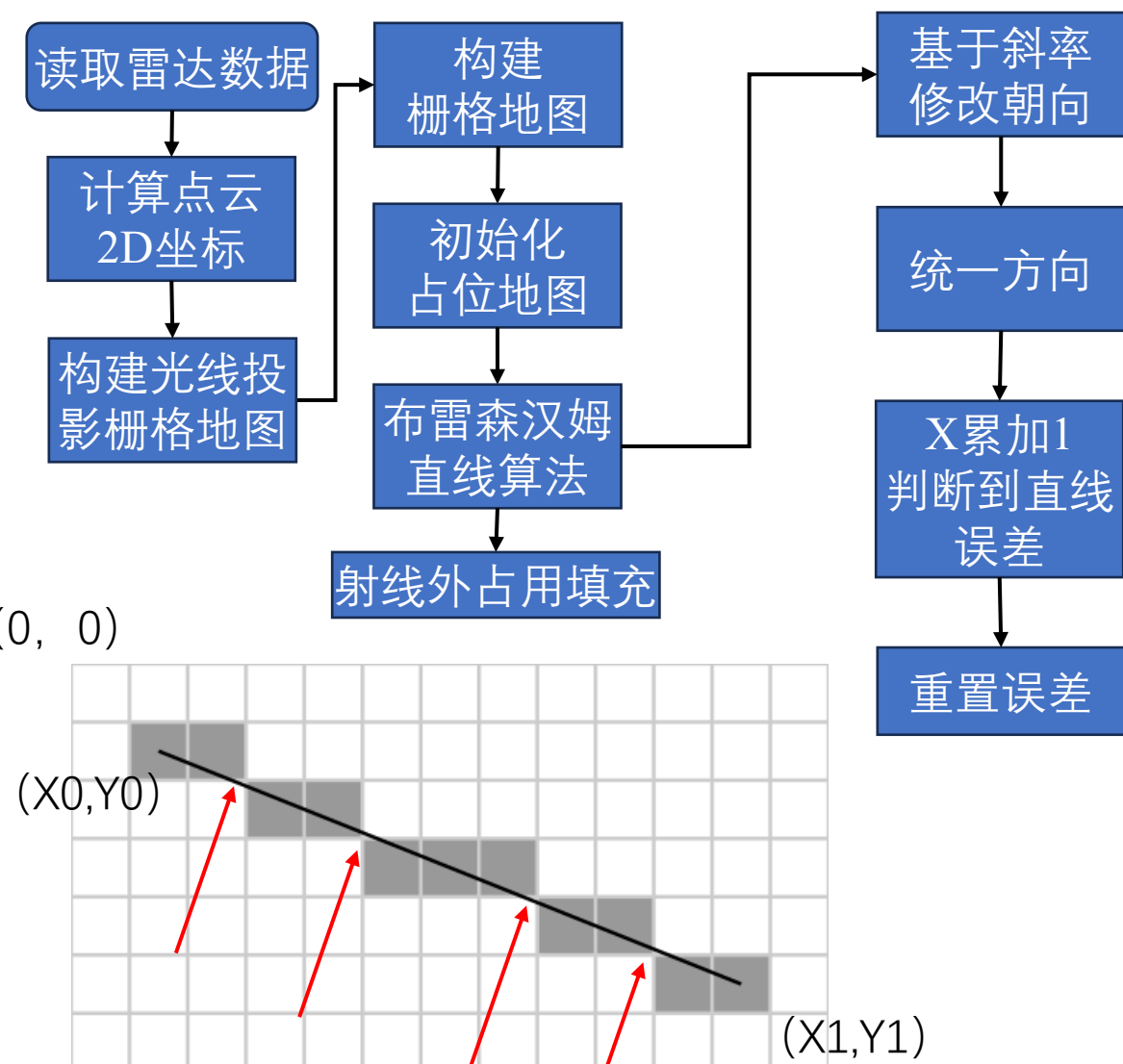


栅格图坐标



Part 2 | Mapping (static)

建图算法 (静态) -- 雷达占位图



然后利用布雷森汉姆直线算法，计算传感器中心 (0, 0) 到每束激光终点（障碍物）所对应的栅格之间的光线。

初始化：算法首先通过沿x和y轴的坐标差（保证斜率在0-1之间），以及起始点大小，将“起点-终点”顺序调整到统一情况，使得直线沿y轴向下递增；x轴向右递增。

初始误差为x坐标间隔的一半： $error = \text{int}(dx/2)$
x每增加1，直线y增加 dy/dx , 误差减 $abs(dy)$, 也就是y方向的总偏移。
error < 0 时修正误差， $y+1$

```
for x in range(x1, x2 + 1): #计算速度快。  
    coord = [y, x] if is_steep else (x, y)  
    points.append(coord)  
    error -= abs(dy)  
    if error < 0:  
        y += y_step  
        error += dx
```

Part 2 | Mapping (static)

建图算法（静态）--雷达占位图

布雷森汉姆 直线算法

参数设置

起点坐标 (x1, y1)

5

5

终点坐标 (x2, y2)

30

20

绘制直线

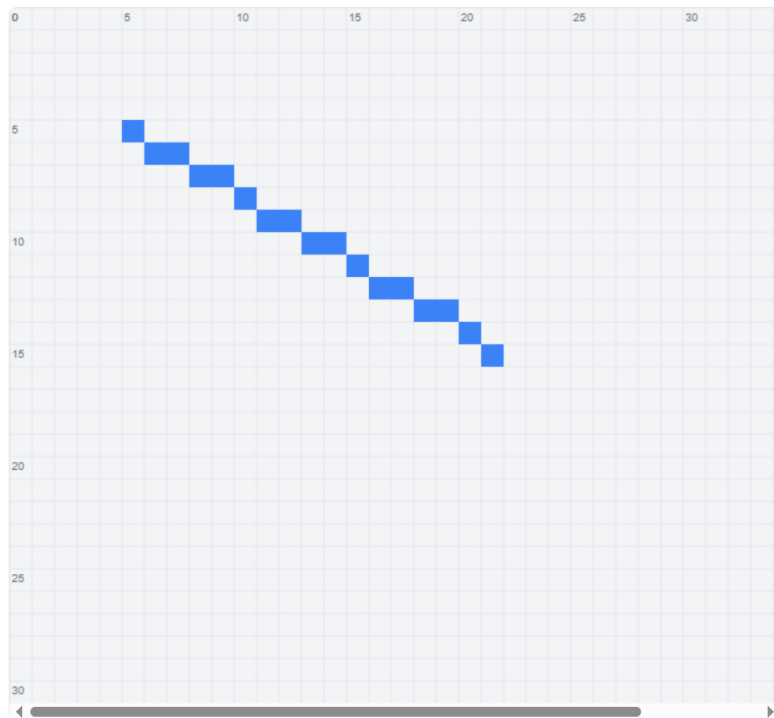
下一步

重置

算法步骤

1. 起点(5,5) → 终点(30,20)

可视化结果



生成的点集

(5,5), (6,6), (7,6), (8,7), (9,7), (10,8), (11,9), (12,9), (13,10), (14,10), (15,11), (16,12), (17,12), (18,13), (19,13), (20,14), (21,15)

函数 布雷森汉姆直线算法(起点, 终点):

从起点获取(x1, y1), 从终点获取(x2, y2)

计算x方向差值 $dx = x2 - x1$

计算y方向差值 $dy = y2 - y1$

判断线段是否陡峭 (y方向变化大于x方向):

如果是, 交换x和y坐标 (旋转线段)

如果起点x大于终点x:

交换起点和终点坐标

标记为已交换

重新计算dx和dy

初始化误差值为dx的一半 (整数除法)

确定y方向步长 (上移为1, 下移为-1)

初始化y为起点y坐标

创建空列表存储线段点

从起点x到终点x循环:

如果线段陡峭, 存储坐标为(y, x), 否则存储(x, y)

将坐标添加到列表

误差值减去dy的绝对值

如果误差值小于0:

y按步长移动

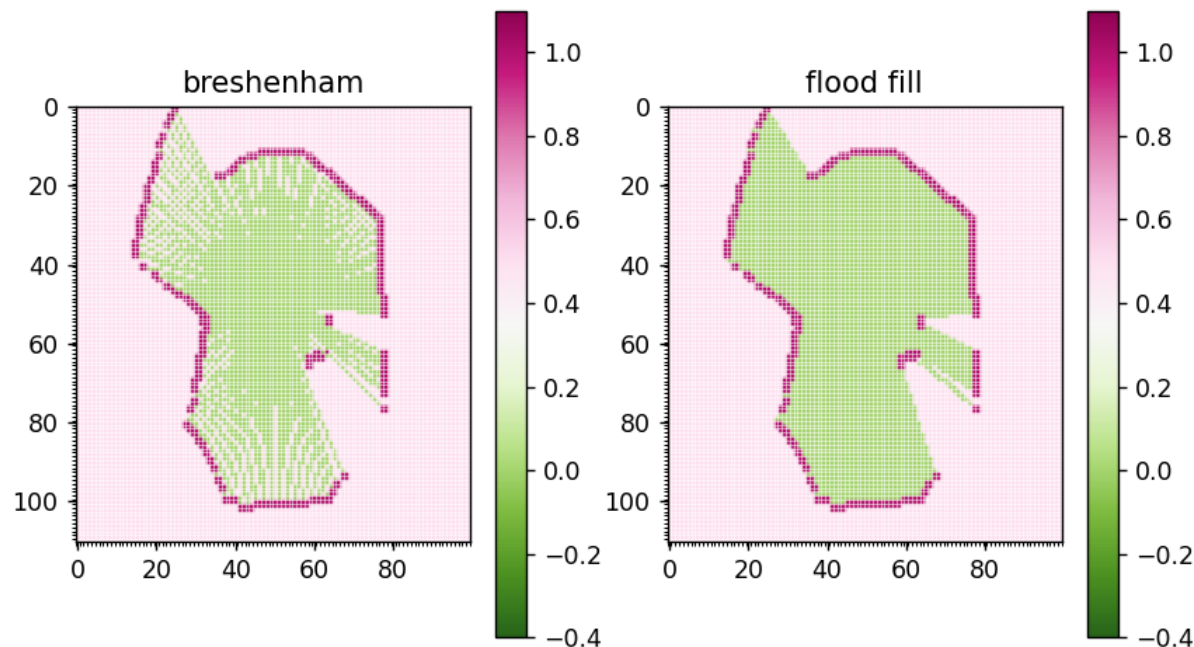
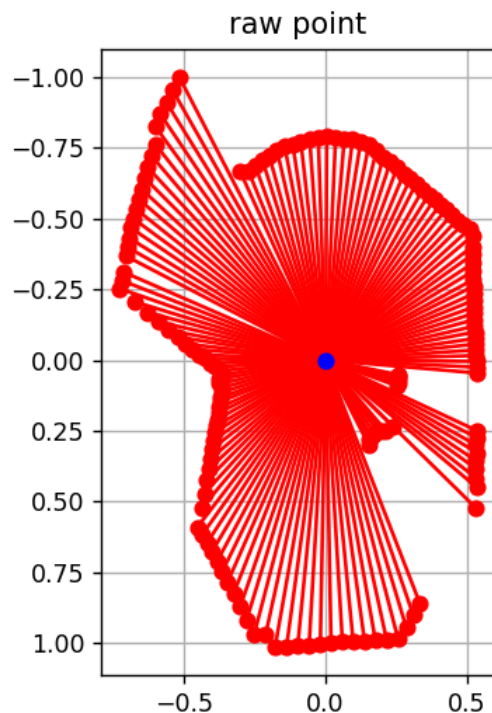
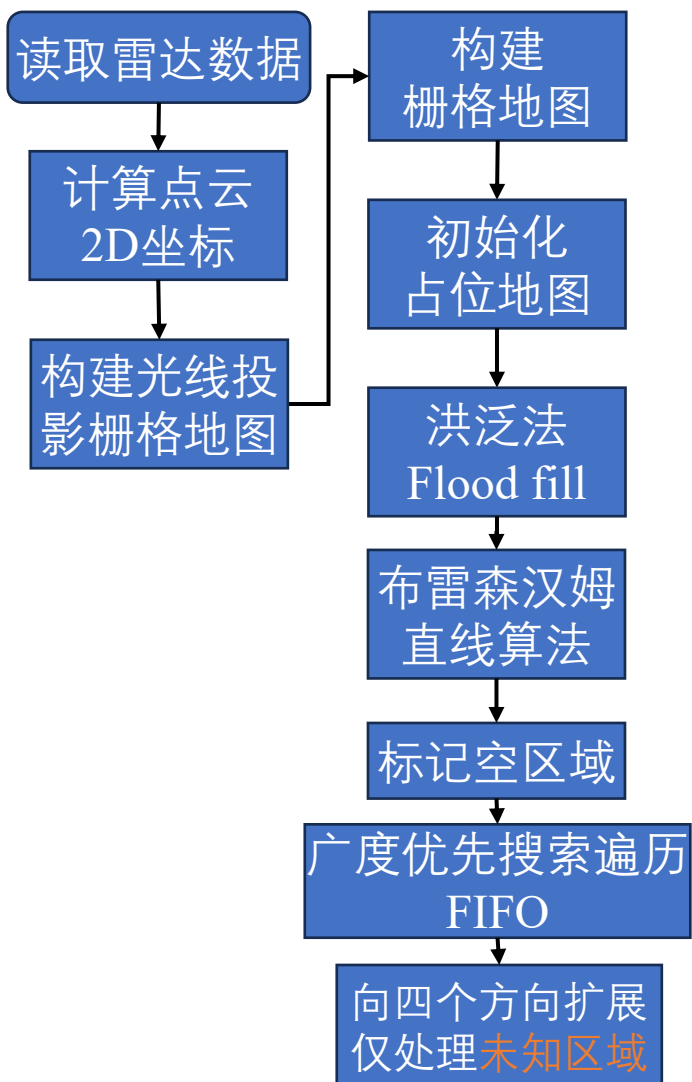
误差值加上dx

如果之前交换过起点终点, 反转点列表

将列表转换为数组并返回

Part 2 | Mapping (static)

建图算法 (静态) -- 雷达占位图



```
bresenham time: 0.004182536602020264  
flood fill time: 0.004353456497192383
```

洪范相当于在直线检测的基础上，对空白区域进行填充。注意观测射线差异。这也是之前**射散**的原因和优化思路之一。

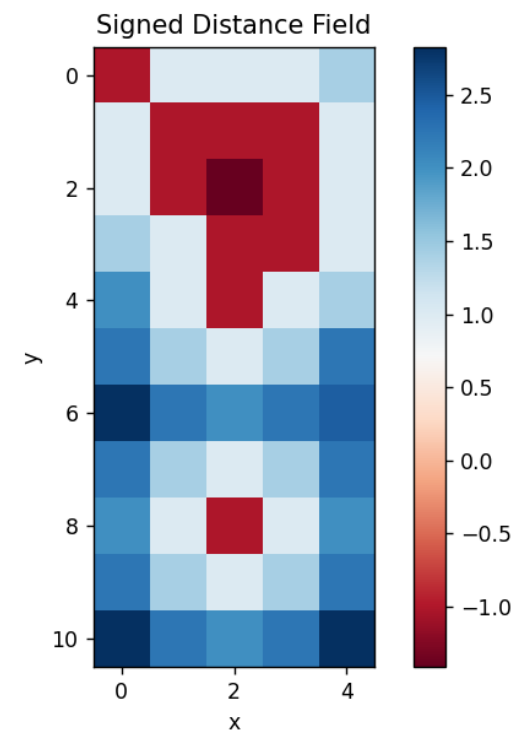
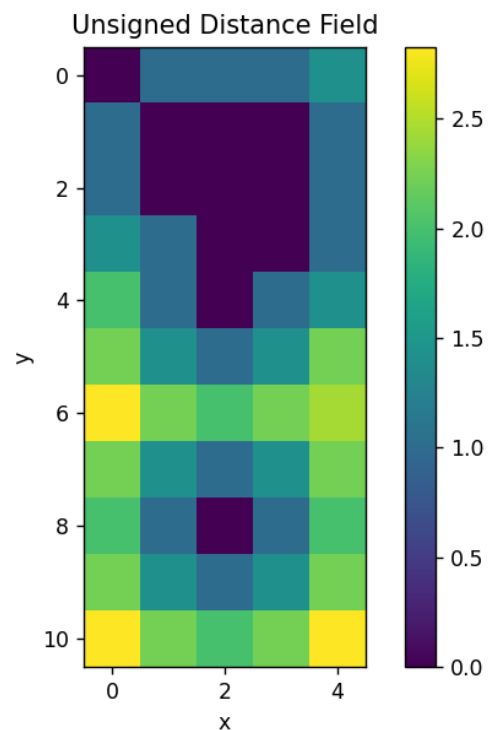
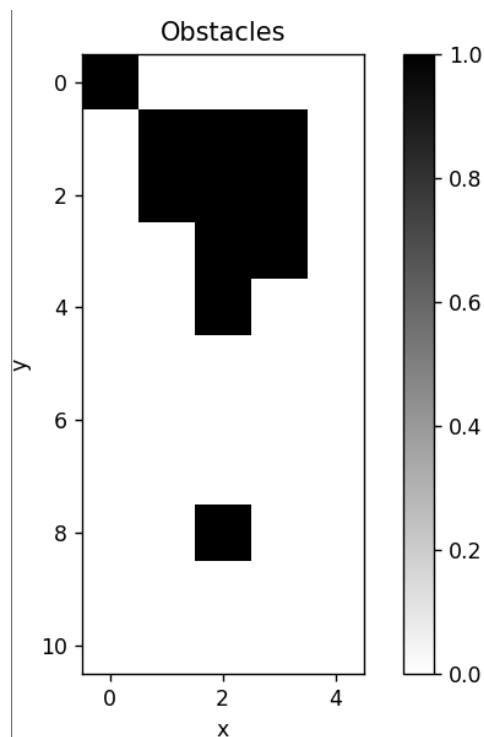
Part 2 | Mapping (static)

建图算法 (静态) --Distance Map 距离地图

距离地图主要对象是二值图像，但应用并不局限于机器人建图。
感知结果输入为一个二值矩阵，1=占位，0=空。

数组

```
obstacles = np.array(  
[  
    [1, 0, 0, 0, 0],  
    [0, 1, 1, 1, 0],  
    [0, 1, 1, 1, 0],  
    [0, 0, 1, 1, 0],  
    [0, 0, 1, 0, 0],  
    [0, 0, 0, 0, 0],  
    [0, 0, 0, 0, 0],  
    [0, 0, 0, 0, 0],  
    [0, 0, 1, 0, 0],  
    [0, 0, 0, 0, 0],  
    [0, 0, 0, 0, 0],  
])
```



Part 2 | Mapping (static)

建图算法 (静态) --Distance Map 距离地图

无符号距离场(UDF): 到障碍物的距离 (≥ 0)

矩阵输入

数据合法验证

0->Inf
1->0

按行计算

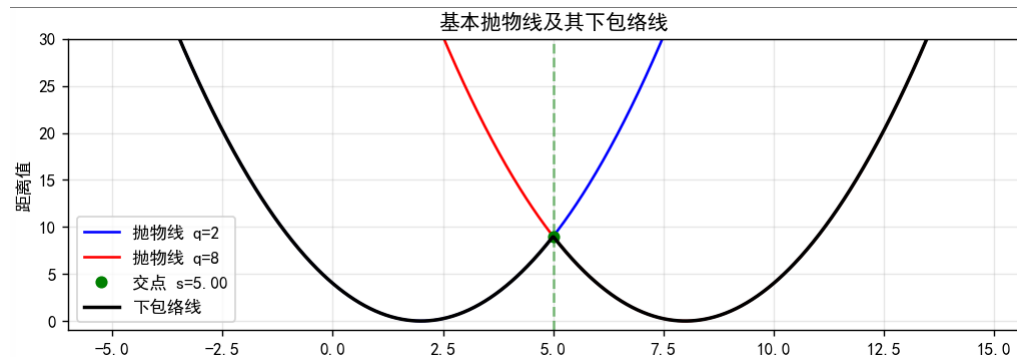
1D欧式距离变换

按列计算

1D欧式距离变换

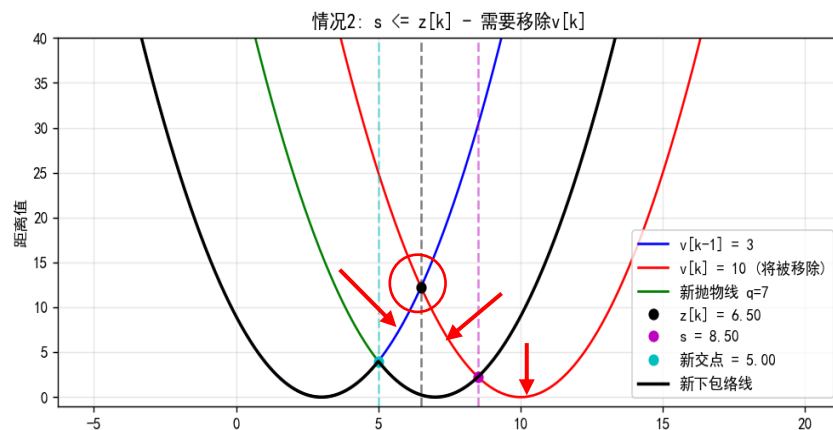
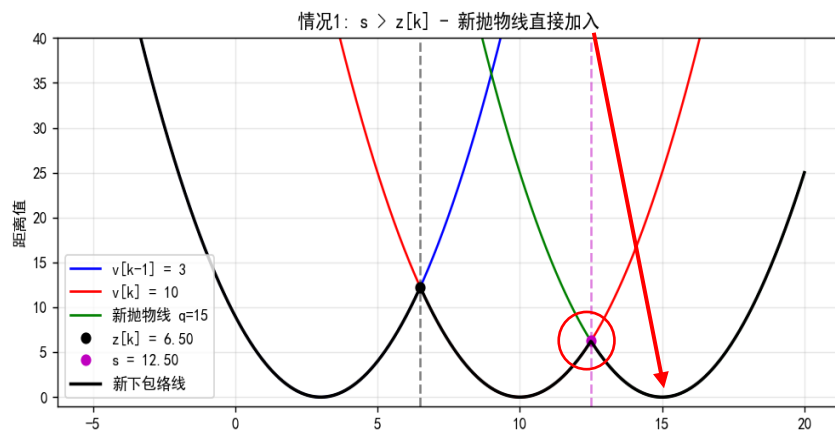
一维欧氏距离变换用于计算一维数组中每个位置到最近“障碍物”(初始距离为0的点)的平方距离。

两条抛物线交点横坐标 $x=s$



$$s = ((d[q] + q * q) - (d[\text{int}(v[k])] + v[k] * v[k])) / (2 * q - 2 * v[k])$$

原相交的抛物线 红-蓝, 新抛物线 绿



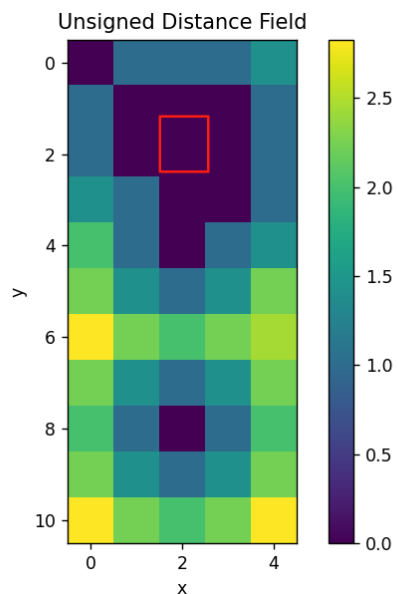
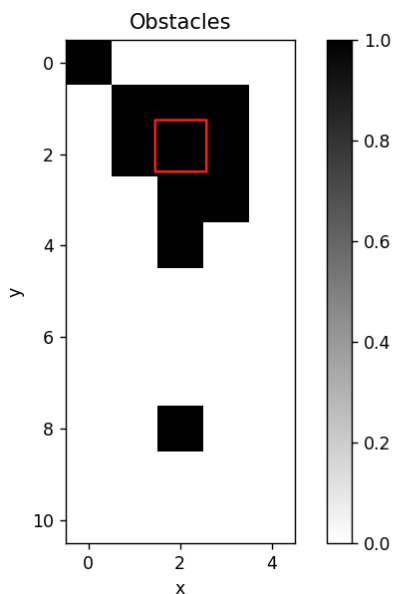
Part 2 | Mapping (static)

建图算法 (静态) --Distance Map 距离地图

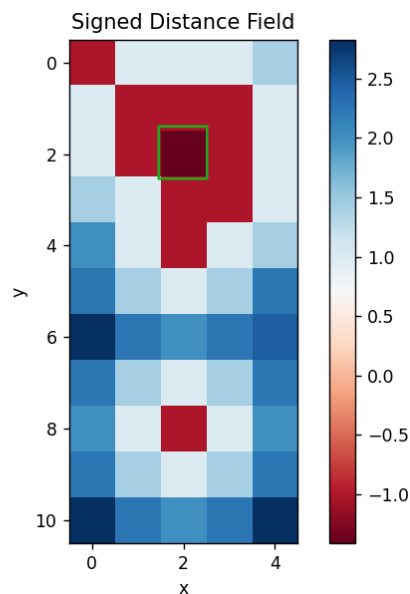
有符号距离场(SDF):

用**正值**表示到最近障碍物栅格的距离，用**负值**表示到最近空栅格的距离。

实际通过求：1) 无符号距离场；2) 输入取反后，求无符号距离场；3) 做差。



内外无区分



```
a = compute_udf(obstacles)
b = compute_udf(obstacles == 0)
return a - b
```

通过互反的两个UDF，将所有点到最近障碍物的距离进一步拓展，并且能够量化障碍物**内部**的“安全程度”，有助于生成连续、平滑且安全的轨迹。

*也提供了封装好的函数直接计算，效率更高。

`scipy.ndimage.distance_transform_edt()`

目录

Contents

01 课程内容安排

02 建图算法

03 目标轮廓识别

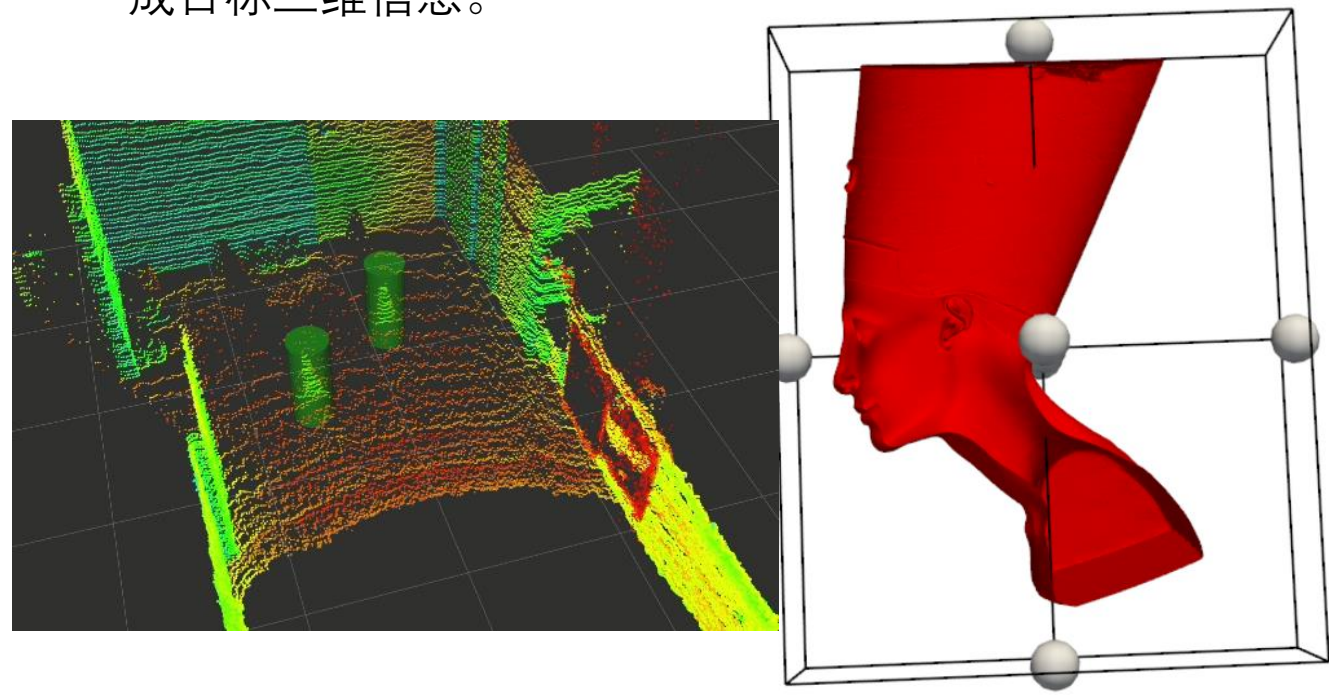
Part 2 | Mapping (static)

» 建图算法 (静态) --点云采样

三维扫描结果按扫描结果可以分为：表面扫描和穿透扫描两类。

表面扫描：在机器人领域，所使用的传感器通过飞秒测距（Time of Flight, ToF），主要是利用波从传感器到目标的往返时间进行测距；进而利用多束光波实现多点采样（一般为宏观并行，微观串行），构建目标轮廓，实际上仅能扫描**目标表面**。

断层扫描：在医疗图像领域，通过大功率的扫描射线，如CT、核磁共振，能够穿透目标；利用连续的断层组成目标三维信息。



数据维度的增加，带来计算效率的下降；特别针对室内和户外大规模场景，如100坪的房间；KITTI户外一般处理 $x=[-40,40]$; $y=[-40,40]$; $z=[-2,4]$ ，即便点云稀疏，遍历如此大规模的范围，所需要消耗的时间太长，不利于**实时性**。

Part 2 | Mapping (static)

建图算法 (静态) --点云采样

点云采样：如何找出少量具有代表性的关键特征点，代替原始数据，满足计算需求。（在不用应用中，选择条件不一）下面介绍几种典型的统计采样。

- 1) 体素化 (voxelization)：将三维空间划分为等大小的体素网格（定义体素边长），每个网格内的点通过求均值合并为一个采样点（计算方式不唯一）。
- 2) 最远点采样 (Farthest point sampling)：迭代选择离已选点集最远的点（定义采样规模），确保采样点分布均匀。
- 3) 泊松圆盘采样 (poisson disk sampling)：随机选择点，但需要保证与已选择点的距离不小于设定值（定义最小距离阈值和迭代次数）。

维度	体素采样	最远点采样	泊松圆盘采样
原理	将空间划分为等大小体素网格，每个网格内的点取均值作为采样点。	迭代选择离已选点集最远的点，优先保证采样点在空间中均匀分布。	随机选择点，但强制新点与已有点的距离不小于设定阈值，确保点间最小间距。
采样特性	采样点数量取决于体素大小和点云分布（体素越密，采样点越多），结果是网格均值，可能丢失细节。	严格保证输出指定数量的点，且点在空间中分布均匀，能覆盖原始点云的“边缘”区域。	输出点数量可能小于目标值（若难以找到满足距离条件的点），点间距离严格大于等于阈值，分布较均匀。
计算效率	较高（通过字典分组和均值计算，时间复杂度与点数量线性相关）。	中等（每次迭代需计算所有点到已选点的距离，时间复杂度为 $O(N^2)$ ， N 为原始点数量）。	较低（随机尝试 + 距离校验，可能需要大量迭代，尤其当最小距离较大或点云密集时）。
适用场景	点云简化（如降低数据量）、保留整体分布趋势（忽略局部细节）。	需均匀覆盖点云空间的场景（如点云特征提取、关键点选择）。	要求点间间距可控的场景（如避免采样点过于密集，用于渲染、重建等）。
参数	依赖体素大小：过小则采样点多（接近原始点），过大则过度简化（丢失结构）。	依赖目标采样点数：点数越多，结果越接近原始分布，但计算成本越高。	依赖最小距离：距离越大，越难达到目标点数；距离过小则与随机采样差异不大。
随机性	无随机性。	初始点随机选择，但后续选择是确定性的（基于最远原则）。	完全随机选择（但受距离约束），结果可能因随机种子不同而有差异。

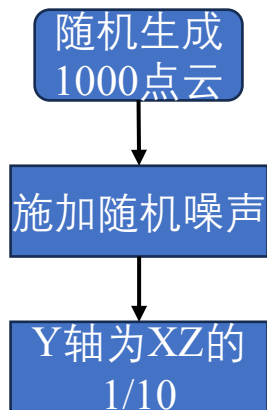
Part 2 | Mapping (static)

建图算法 (静态) --点云采样

点云采样：如何找出少量具有代表性的关键特征点，代替原始数据，满足计算需求。

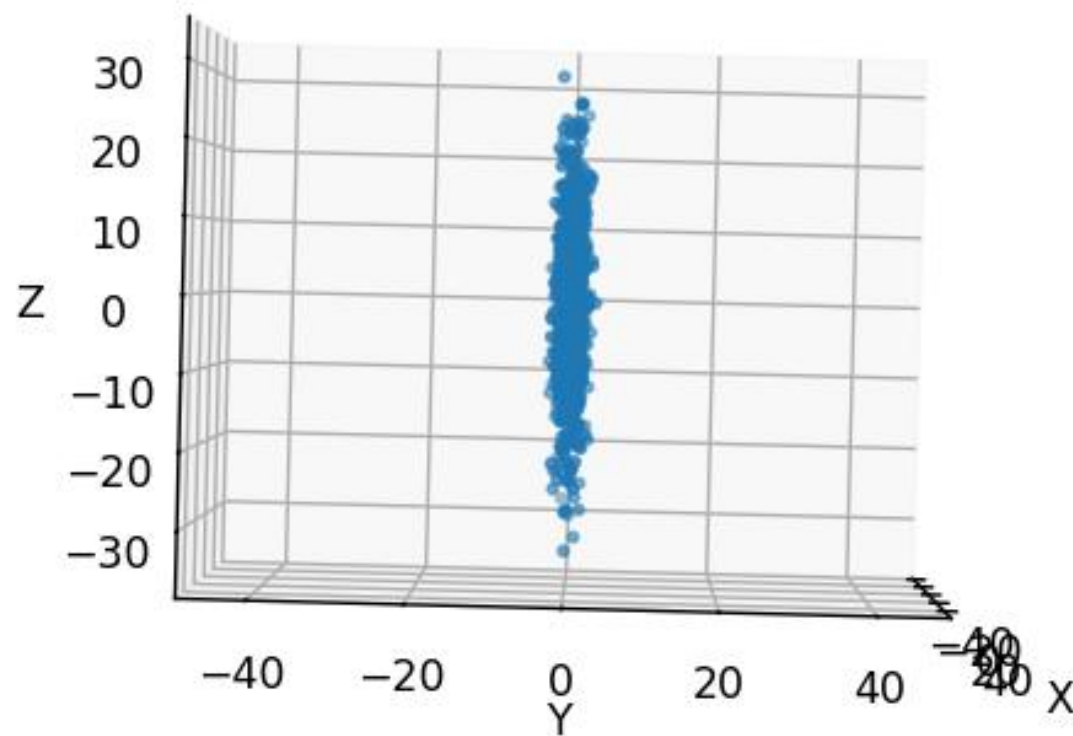
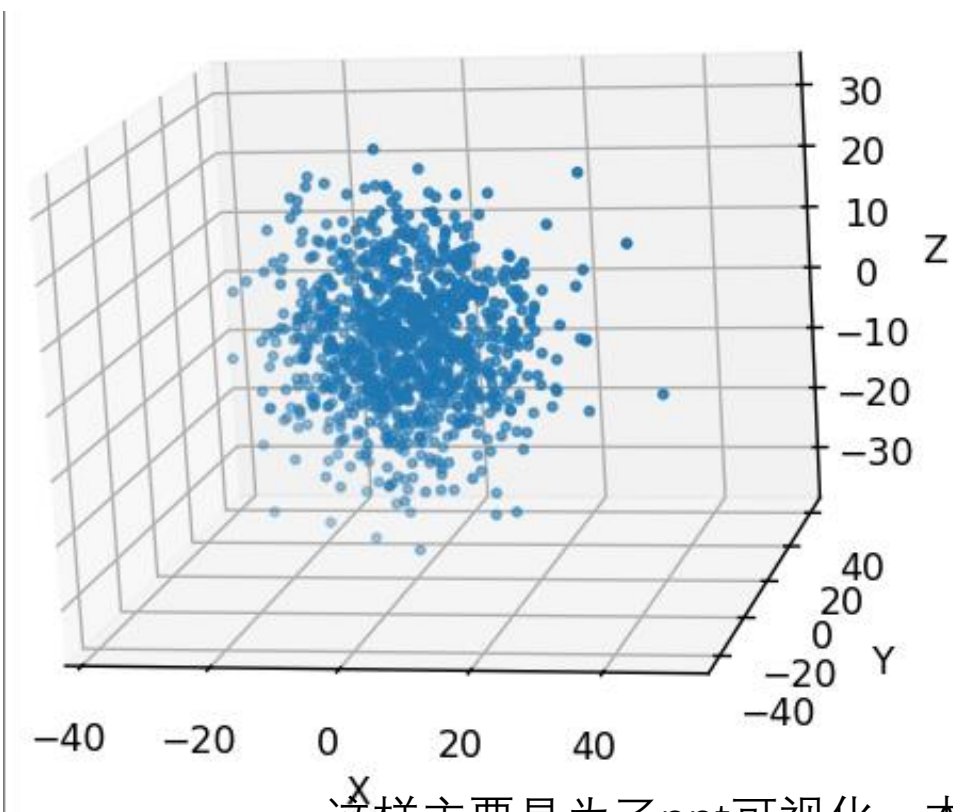
(在不用应用中，选择条件不一)

下面介绍几种典型的统计采样。



```
n_points = 1000
seed = 1234
# 创建随机数生成器
rng = np.random.default_rng(seed)

# 生成三个维度的正态分布随机数，x和z的均值为0，标准差为10，y的均值为0，标准差为1
x = rng.normal(0.0, 10.0, n_points)
y = rng.normal(0.0, 1.0, n_points)
z = rng.normal(0.0, 10.0, n_points)
# 将三个维度的数据堆叠成点的坐标数组
original_points = np.vstack((x, y, z)).T
```



这样主要是为了ppt可视化，本地可以交互拖拽，改成均匀分布测试

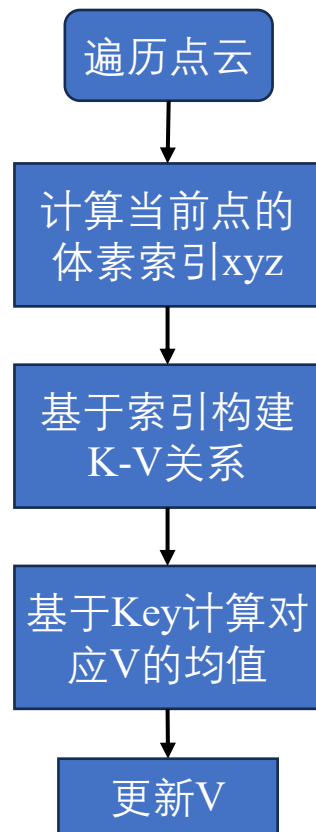
Part 2 | Mapping (static)

建图算法 (静态) --点云采样

点云采样：如何找出少量具有代表性的关键特征点，代替原始数据，满足计算需求。（在不用应用中，选择条件不一）

下面介绍几种典型的统计采样。

1) **体素化 (voxelization)**：将三维空间划分为等大小的体素网格（**定义体素边长=20**），每个网格内的点通过**求均值**合并为一个采样点（计算方式不唯一）。



函数 体素点采样(原始点云, 体素大小):

创建一个空的体素字典 (键为体素索引, 值为该体素内的点列表)

对于原始点云中的每个点:

计算该点在每个维度上的体素索引 (坐标值除以体素大小后取整数)

将体素索引转换为元组 (作为字典的键)

将该点添加到体素字典中对应应的列表里

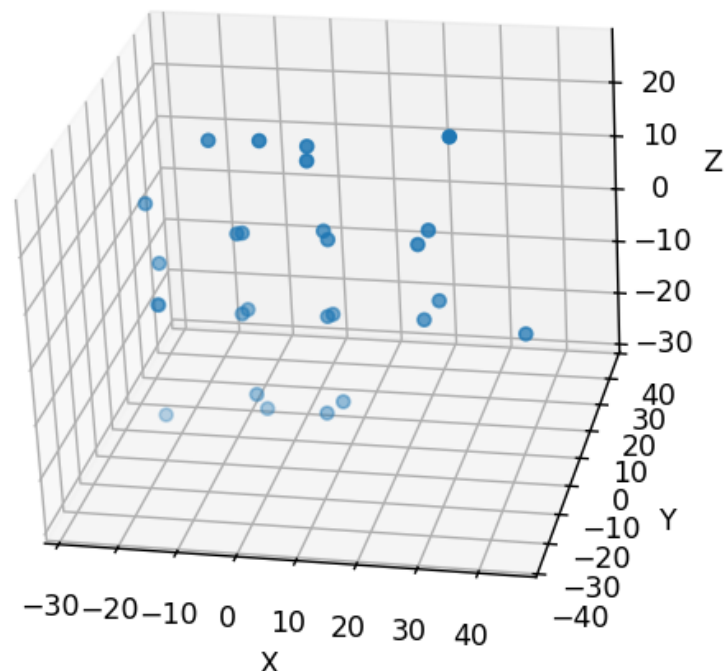
初始化一个空的采样点列表

对于体素字典中的每个体素:

计算该体素内所有点的均值 (每个维度分别求平均)

将均值点添加到采样点列表

将采样点列表转换为数组并返回



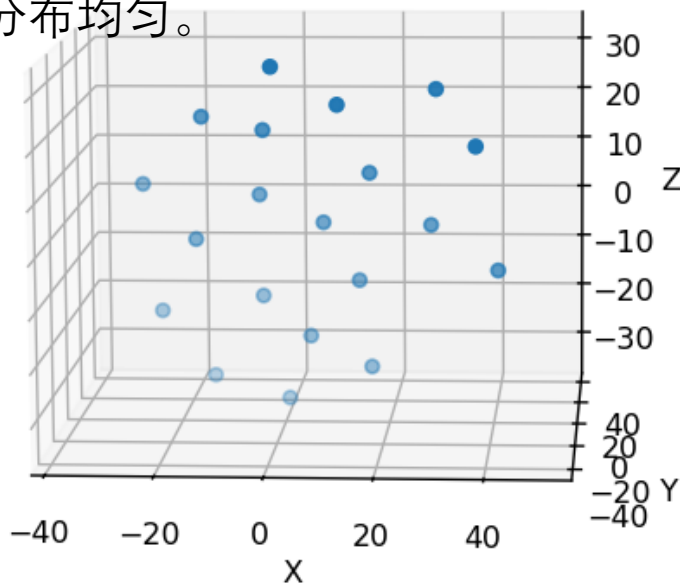
对于密集的点云而言，相邻的voxel会被采样到
通过均值反应到最终的点上，能够体现原点云的分布

Part 2 | Mapping (static)

建图算法 (静态) --点云采样

点云采样：如何找出少量具有代表性的关键特征点，代替原始数据，满足计算需求。（在不用应用中，选择条件不一）下面介绍几种典型的统计采样。

2) 最远点采样 (Farthest point sampling)：迭代选择离已选点集最远的点（定义采样规模=20），确保采样点分布均匀。



最远点采样保证了采样的均匀性

函数 最远点采样(点云, 目标点数):

随机选1个初始点, 加入已选集合

初始化所有点到已选集合的最小距离为 ∞

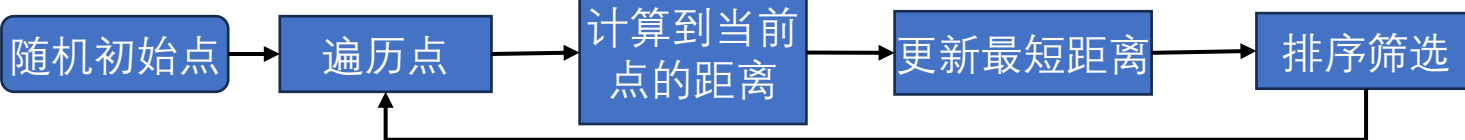
循环至选满目标点数:

计算所有点到最新选点的距离

更新每个点到已选集合的最小距离 (取更小值)

选最小距离最大的未选点, 加入已选集合

返回已选点集合

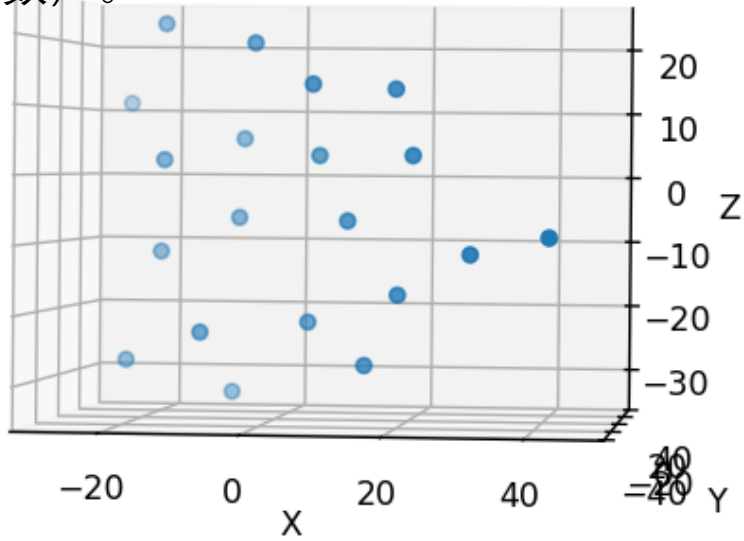


Part 2 | Mapping (static)

建图算法 (静态) --点云采样

点云采样：如何找出少量具有代表性的关键特征点，代替原始数据，满足计算需求。（在不用应用中，选择条件不一）下面介绍几种典型的统计采样。

3) 泊松圆盘采样 (poisson disk sampling)：随机选择点，但需要保证与已选择点的距离不小于设定值（定义最小距离阈值=10，采样规模=20和迭代次数）。



函数 泊松圆盘采样(原始点云, 目标采样点数, 最小距离, 随机种子, 最大迭代次数=1000):

初始化随机数生成器(使用随机种子)

从原始点云中随机选择一个点的索引, 作为第一个选中点

已选点索引列表 = [第一个选中点的索引]

迭代次数 = 0

当已选点索引列表的长度小于目标采样点数 且 迭代次数 \leq 最大迭代次数时:

从原始点云中随机选择一个点的索引

当前点 = 原始点云中该索引对应的点

计算当前点与已选点索引列表中所有点的欧氏距离

如果这些距离中的最小值 \geq 最小距离:

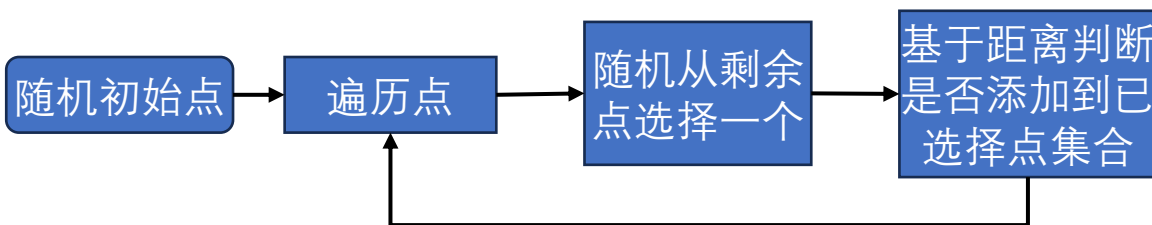
将当前点的索引添加到已选点索引列表

迭代次数 $+= 1$

如果已选点索引列表的长度不等于目标采样点数:

打印提示信息"无法找到指定数量的点..."

返回原始点云中, 已选点索引列表对应的点



Part 2 | Mapping (static)

建图算法 (静态) --K-means聚类

K-means 是一种经典的**聚类算法**，属于**无监督学习**方法，用来将数据分成 k 个簇 (clusters)，让同一簇内的数据点尽量相似，而不同簇之间的差异尽量大。需要已知目标类数量 k 。

假设数据集为 $X = \{x_1, x_2, \dots, x_n\}$ ，希望分成 k 个簇：

1. **初始化**：随机选择 k 个数据点作为初始簇中心 $\mu_1, \mu_2, \dots, \mu_k$ 。

2. **分配步骤 (Assignment)**

对每个点 x_i ，找距离最近的簇中心：

$$c_i = \arg \min_j \|x_i - \mu_j\|^2$$

3. **更新步骤 (Update)**

对每个簇 j ，更新簇中心为该簇内所有点的均值：

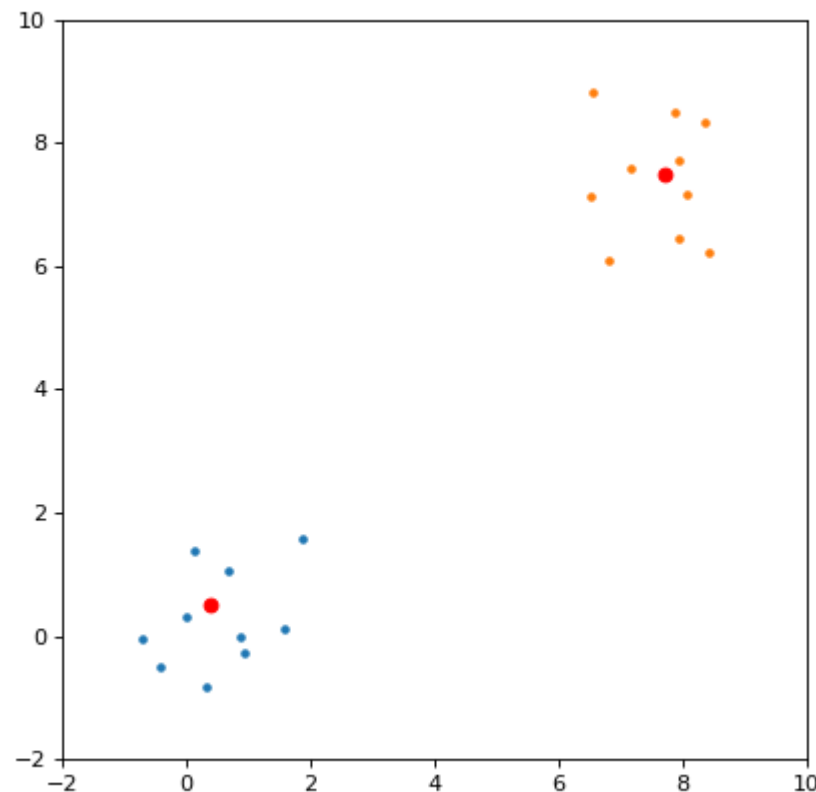
$$\mu_j = \frac{1}{|C_j|} \sum_{x_i \in C_j} x_i$$

4. 重复 **分配** 和 **更新**，直到簇中心不再变化或达到迭代上限。

模拟两个匀速对向行驶的车辆，观测各为一簇点云。

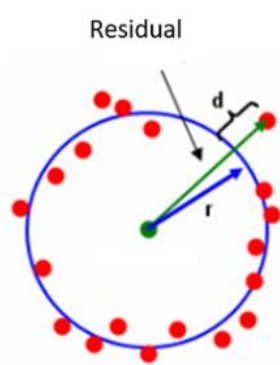
基于K-means对两簇点进行持续聚类，保证属于同一目标的点的观测点云被正确分类。

1. 初始两个目标位置分别为(0,0)和(8,8)；
2. 采用大致相反的方向向量(0.4, 0.5)和(-0.4, -0.5)进行周期迭代；
3. 以目标为中心，随机生成各10个点，模拟传感器观测点云。
4. 针对每轮运动更新后的点云进行Kmeans聚类。



Part 2 | Mapping (static)

建图算法 (静态) --几何轮廓识别



通过每个采样点计算到拟合圆心的距离（测量半径） - 拟合半径，构建残差，进行最小二乘拟合。

几何轮廓识别主要利用先验信息明确待检测和拟合的目标采样点的具体几何形状，如：圆形、矩形、椭圆等。基于一系列采样输入点集，进行基于最小二乘的目标函数拟合。

以二维平面的圆形为例：

Kåsa 法 (Kåsa, 1976) 代数最小二乘拟合圆, Algebraic Circle Fitting

圆的一般方程为：

$$x^2 + y^2 + Ax + By + C = 0$$

其中 A 、 B 、 C 是未知参数。

圆心和半径为：

$$c_x = -\frac{A}{2}, \quad c_y = -\frac{B}{2}, \quad r = \sqrt{c_x^2 + c_y^2 - C}$$

给定一组点 (x_i, y_i) ，代入方程：

$$x_i^2 + y_i^2 + Ax_i + By_i + C = 0$$

我们希望找到 A, B, C 使得：

$$\sum_{i=1}^n (x_i^2 + y_i^2 + Ax_i + By_i + C)^2 \quad \text{最小}$$

对 A, B, C 求偏导，令导数为0，得法向方程组

$$F = \begin{bmatrix} \sum x_i^2 & \sum x_i y_i & \sum x_i \\ \sum x_i y_i & \sum y_i^2 & \sum y_i \\ \sum x_i & \sum y_i & n \end{bmatrix} \begin{bmatrix} A \\ B \\ C \end{bmatrix} = - \begin{bmatrix} \sum (x_i^3 + x_i y_i^2) \\ \sum (x_i^2 y_i + y_i^3) \\ \sum (x_i^2 + y_i^2) \end{bmatrix} = G$$

$$\begin{bmatrix} A \\ B \\ C \end{bmatrix} = F^{-1}G$$

Part 2 | Mapping (static)

» 建图算法 (静态) --几何轮廓识别

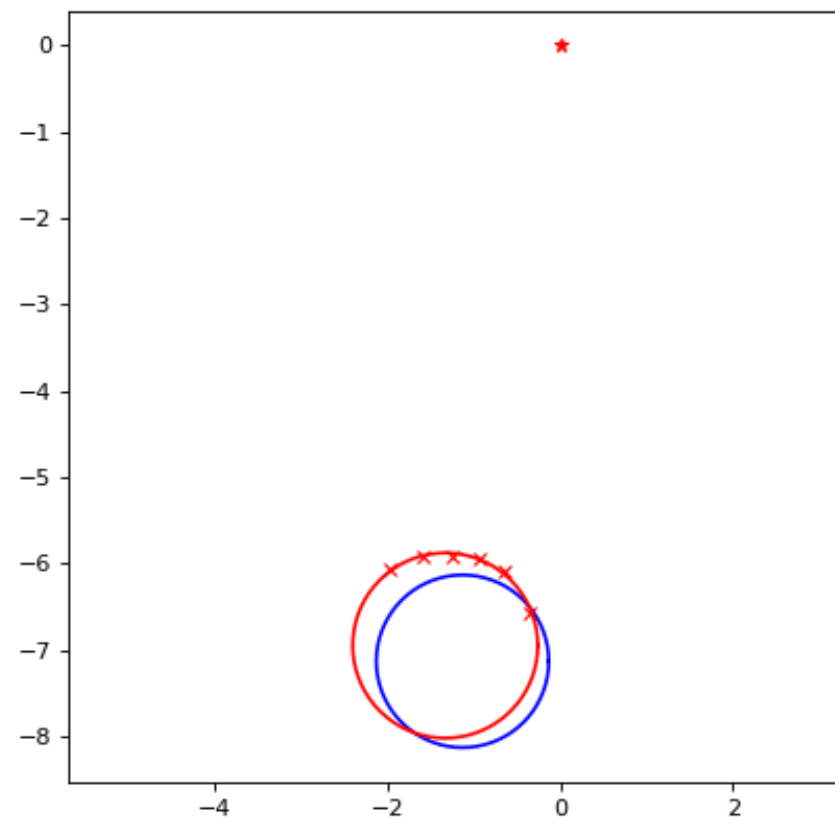
Kåsa 法 (Kåsa, 1976) 代数最小二乘拟合圆, Algebraic Circle Fitting

$$F = \begin{bmatrix} \sum x_i^2 & \sum x_i y_i & \sum x_i \\ \sum x_i y_i & \sum y_i^2 & \sum y_i \\ \sum x_i & \sum y_i & n \end{bmatrix} \begin{bmatrix} A \\ B \\ C \end{bmatrix} = - \begin{bmatrix} \sum (x_i^3 + x_i y_i^2) \\ \sum (x_i^2 y_i + y_i^3) \\ \sum (x_i^2 + y_i^2) \end{bmatrix} = G$$

```
F = np.array([[sumx2, sumxy, sumx],  
              [sumxy, sumy2, sumy],  
              [sumx, sumy, len(x)]])
```

```
G = np.array([[-sum([ix ** 3 + ix * iy ** 2 for (ix, iy) in zip(x, y)]),  
              [-sum([ix ** 2 * iy + iy ** 3 for (ix, iy) in zip(x, y)]),  
              [-sum([ix ** 2 + iy ** 2 for (ix, iy) in zip(x, y)]))])
```

$$\begin{bmatrix} A \\ B \\ C \end{bmatrix} = F^{-1}G \quad T = \text{np.linalg.inv}(F).dot(G)$$



Part 2 | Mapping (static)

» 建图算法 (静态) --几何轮廓识别

无人驾驶中，激光雷达扫描的鸟瞰图下，车辆能够呈现出不完整的矩形（视距内测量）。

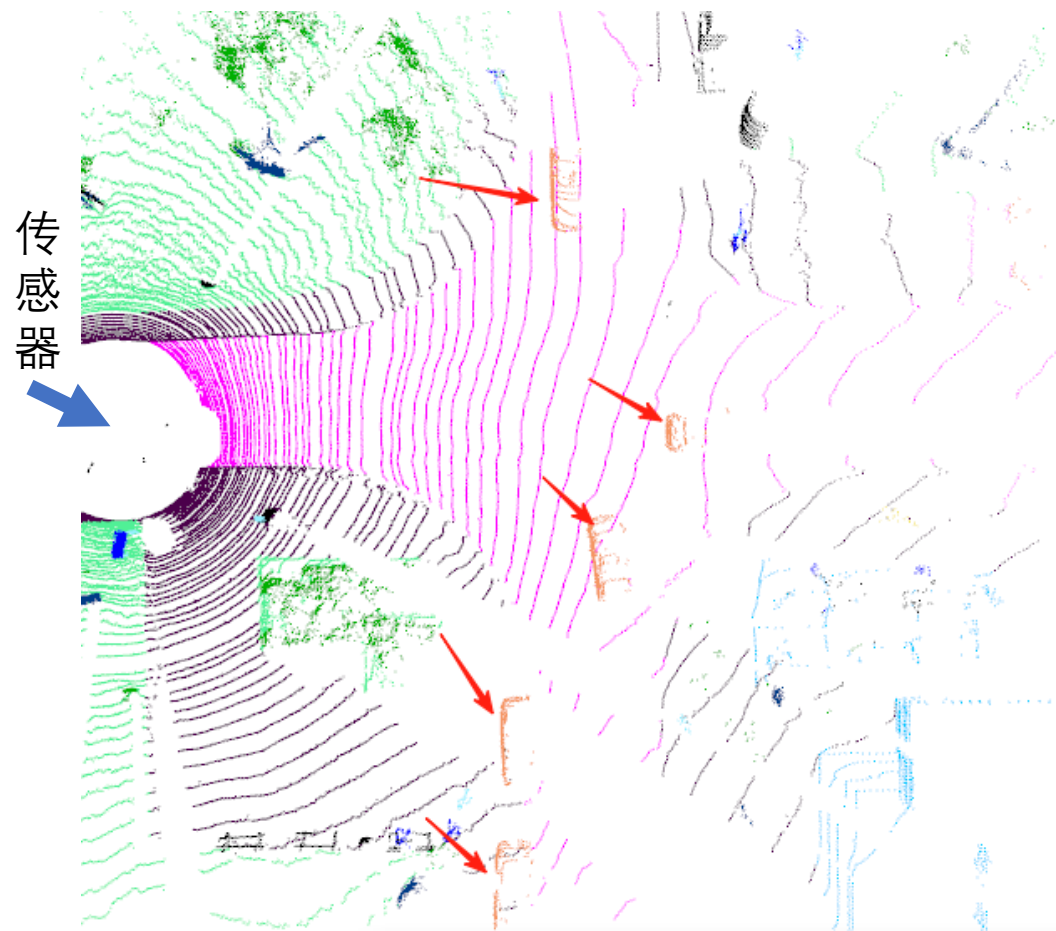
通过扫描点进行**矩形拟合**，可以估计目标的实际位、中心和朝向。

在图像检测任务中，可以利用2D Bounding Box (BBox) 和 旋转BBox 任务来解决相关问题。

当然，最新的技术方案主要采用数据驱动的深度学习建模，因为大多数车辆并非完美矩形，因而会导致误差，加上复杂环境噪声（如地面）。当然，传统方法仍然在一些低算力环境具有应用价值。

也可以基于两类方法相结合，初步用深度学习获取ROI，再在局部进行矩形拟合，进行调优。

如果已知目标长宽，等**先验信息**也可以进一步在拟合过程利用。



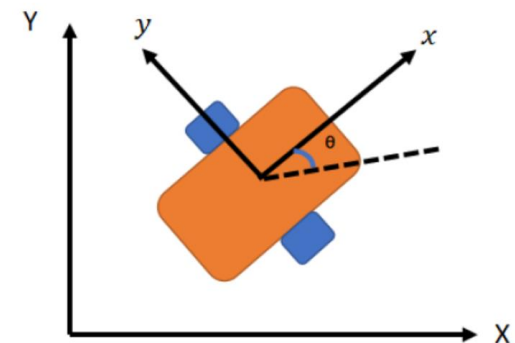
Part 2 | Mapping (static)

» 建图算法（静态） --几何轮廓识别

构建一个基于单线激光雷达扫描的载体（位置为[0,0]）；
场景中存在两个移动车辆（以矩形进行模拟）。

车辆运动学采用lesson1介绍的刚体运动的4维状态向量：

$$\text{state} = [x, y, \theta, \mathbf{v}]^T$$

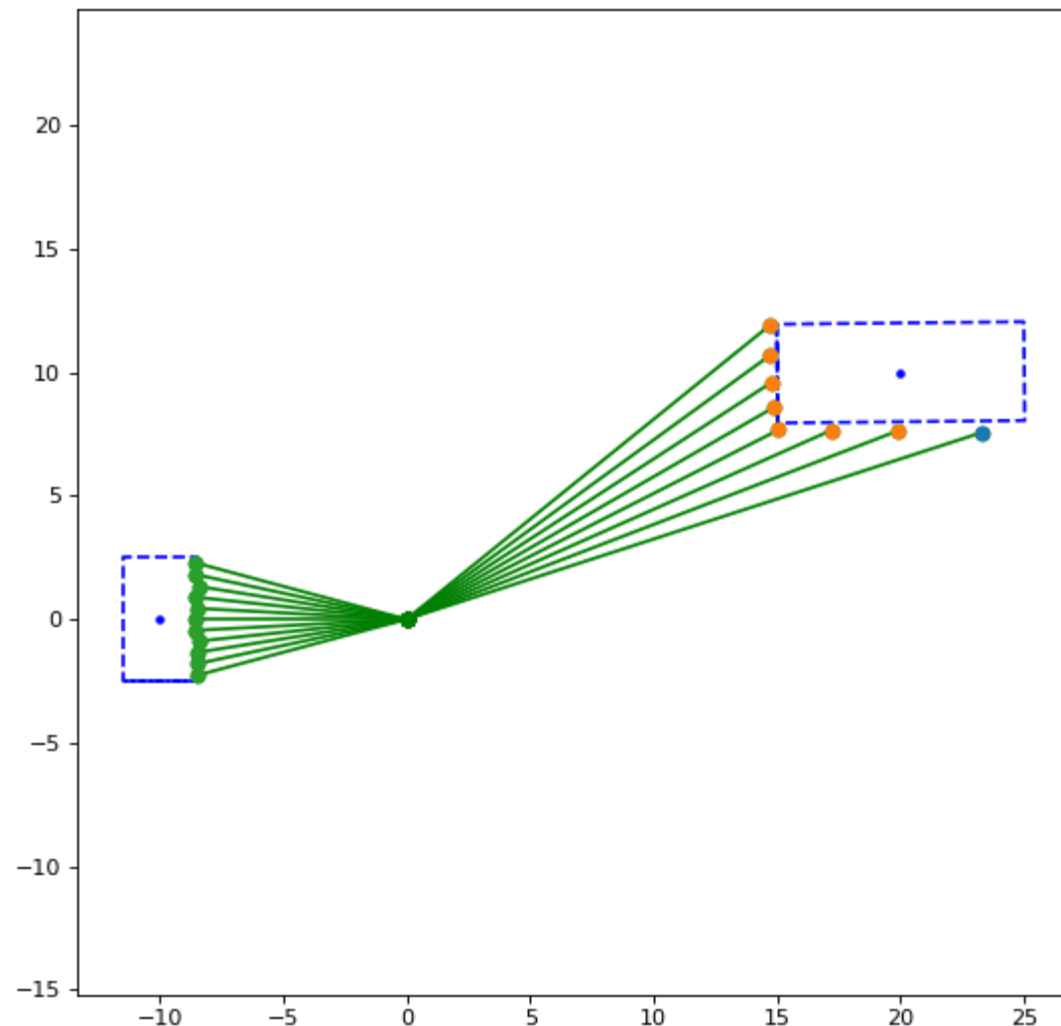


此外，定义车辆的长宽，以及最大速度限制。
基于车辆state和长宽，确定车身在地图上的2D轮廓。

通过**加速度**和**角速度**
控制目标车辆运动
(蓝色虚线框)。

```
def update(self, dt, a, omega):  
    self.x += self.v * np.cos(self.yaw) * dt  
    self.y += self.v * np.sin(self.yaw) * dt  
    self.yaw += omega * dt  
    self.v += a * dt  
    if self.v >= self.max_v:  
        self.v = self.max_v
```

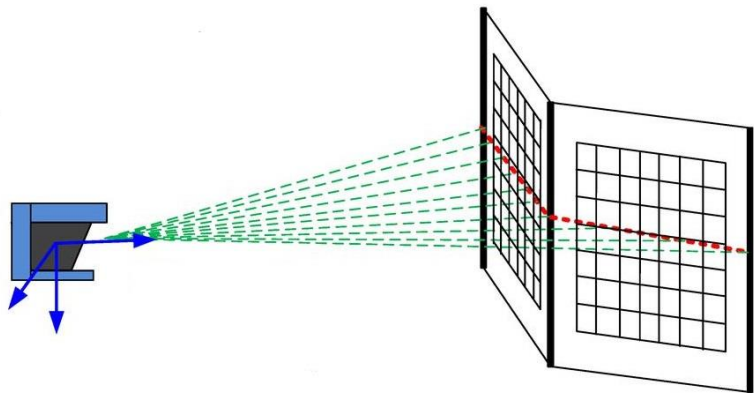
位置 → 角度 → 速度



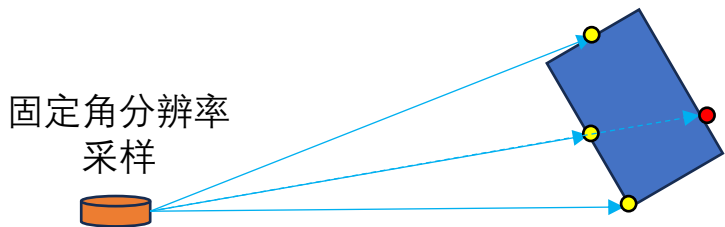
Part 2 | Mapping (static)

» 建图算法 (静态) --几何轮廓识别

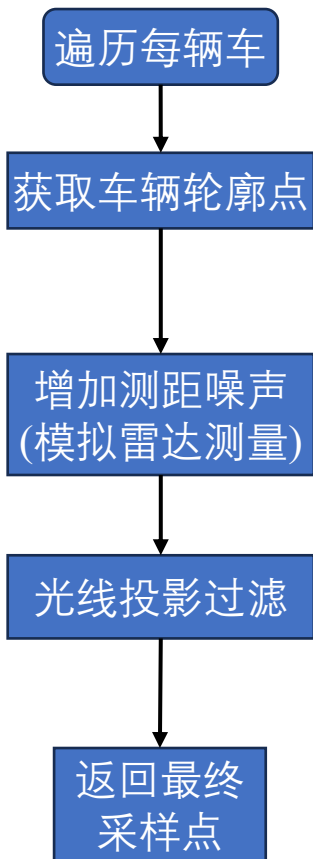
定义观测模型，2D激光雷达属性。



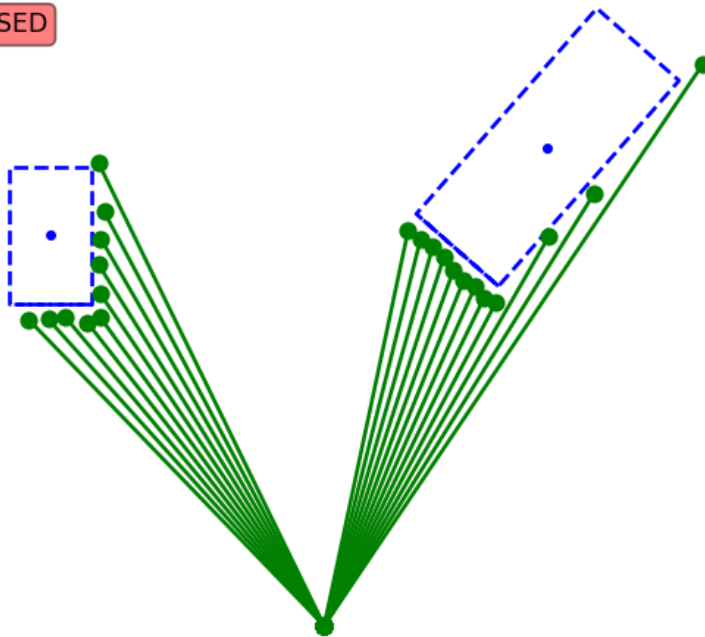
水平角分辨率 (3) 度，测距噪声 (0.01) m。



角分辨率越小，采样点越多；
噪声越大，抖动越强。



PAUSED



尝试修改雷达参数观测估计的结果。

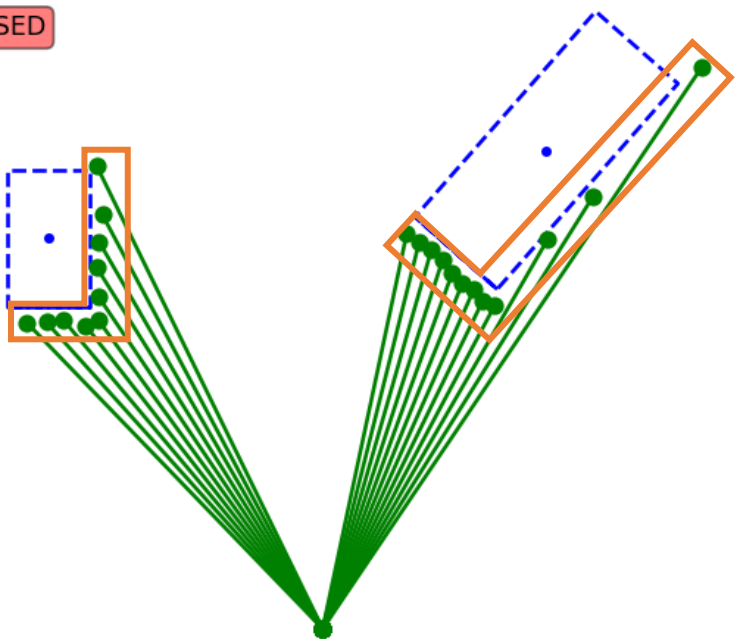
- 但雷达角分辨率过小，而车辆的轮廓插值密度相对的大；则会出现 lesson1 中的穿透现象。是否能够采用线段焦点/其它的方式修正？
- 测距噪声则会导致估计错误。是否可以通过引入粒子/Kalman 滤波进行压制？

Part 2 | Mapping (static)

建图算法 (静态) --几何轮廓识别

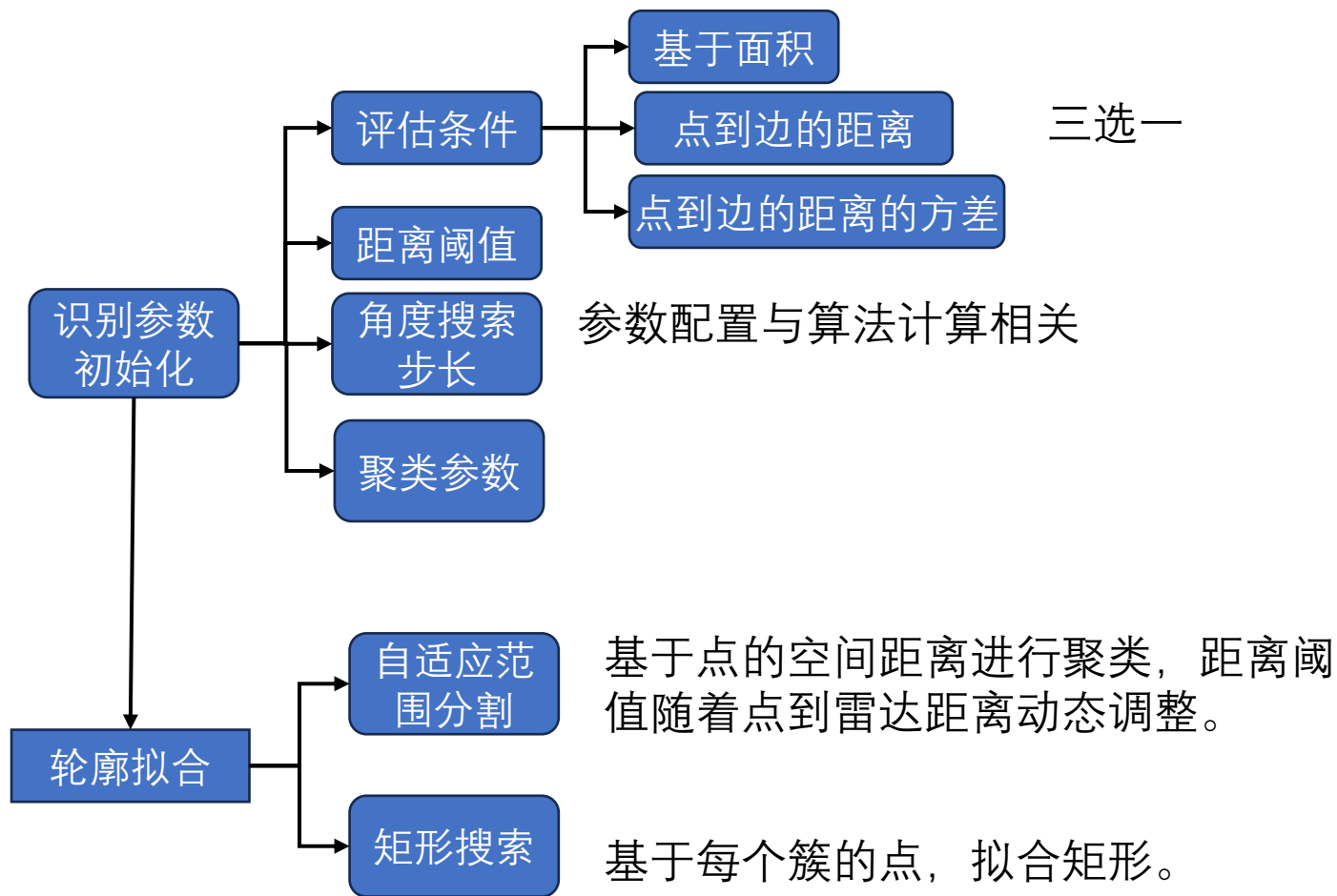
矩形拟合，在本任务中实际上是L-shape拟合。

PAUSED



对于每辆车，考虑它的相邻边是正交的，基于光线追踪原理，在车辆本体无法被光线穿透的情况下，扫描结果一般为“一”或“L”形。

- ①希望拟合得到的矩形能够覆盖扫描点的情况下，尽可能的小
- ②扫描点到矩形边的距离尽可能小
- ③扫描点到矩形边的波动（方差）尽可能下

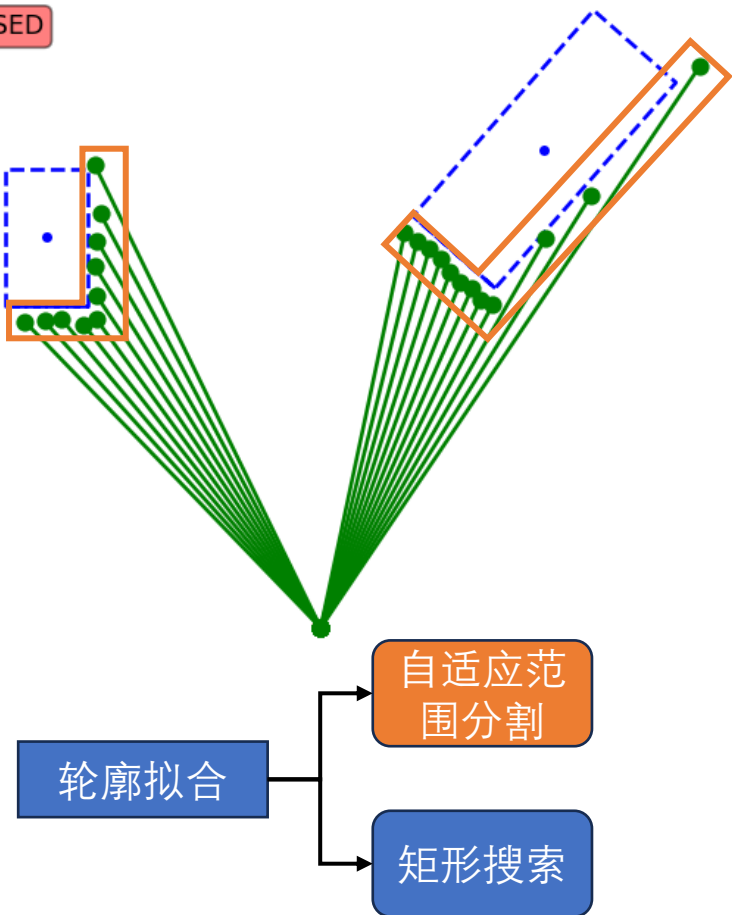


Part 2 | Mapping (static)

建图算法 (静态) --几何轮廓识别

自适距离分割聚类

PAUSED



自底向上的层次聚类

分割

分割阈值：随点到雷达中心距离增加

$$r = R_0 + R_d \cdot \sqrt{x^2 + y^2}$$

- R_0 : 基础半径 (离原点很近的点也有这个最小半径)

- R_d : 半径随离原点距离增加的增长系数

分割阈值：随点到雷达中心距离增加；
基于该方法分割，每个类会包含重复点

合并

- 遍历候选分割列表，每次选择两个子集
- 判断子集之间是否有交集
- 若有，合并两个集合 (被合并的删除)
- 重新开始遍历，直到候选列表不再变化

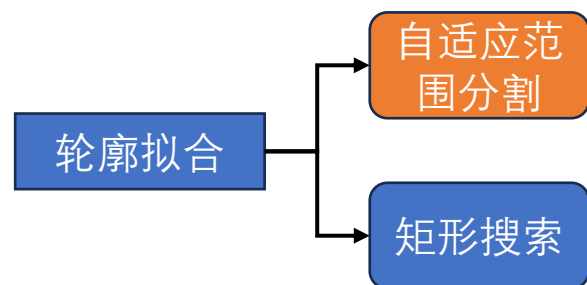
Part 2 | Mapping (static)

建图算法 (静态) --几何轮廓识别

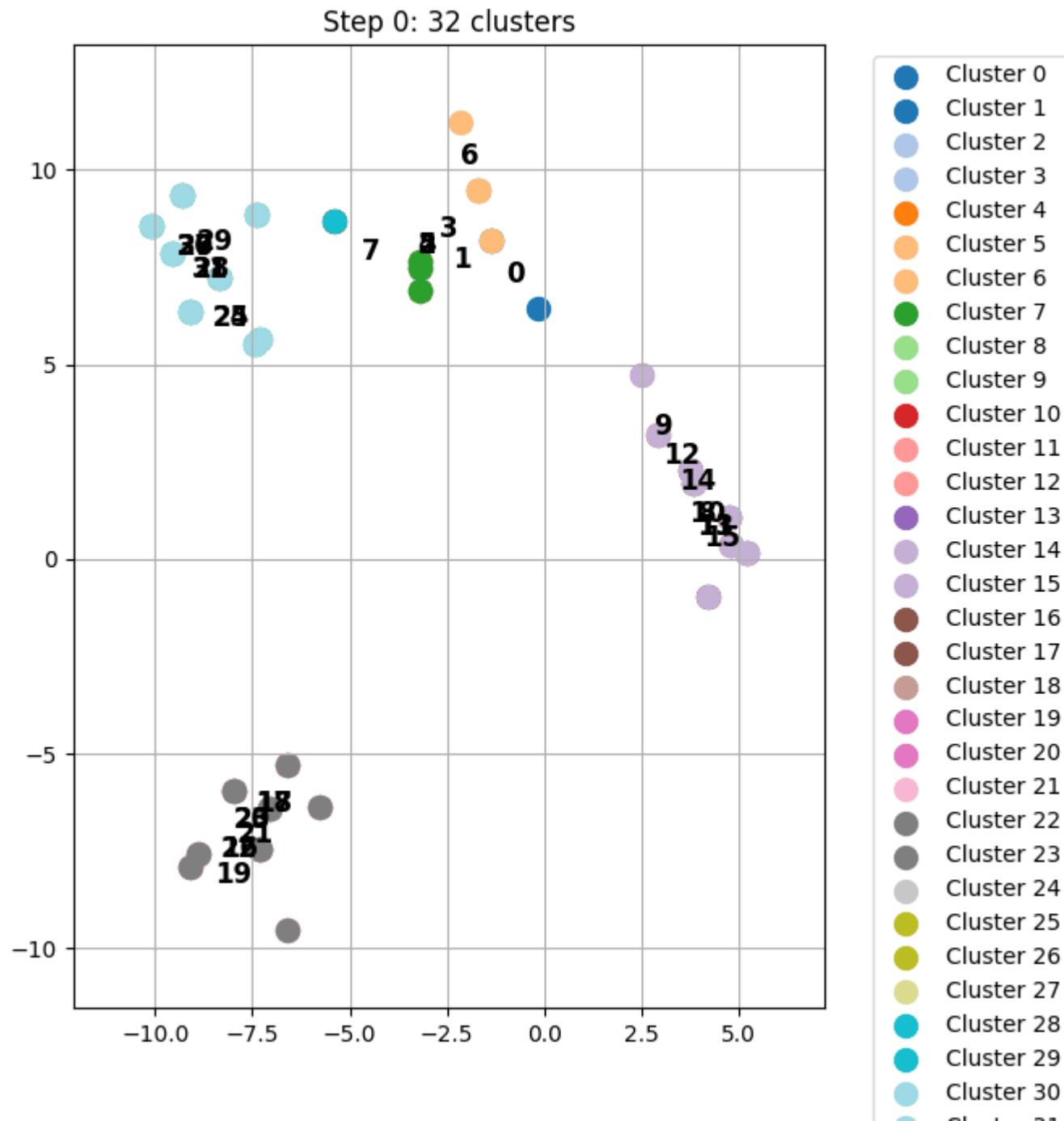
- 合并
- 遍历候选分割列表，每次选择两个子集
 - 判断子集之间是否有交集
 - 若有，合并两个集合（被合并的删除）
 - 重新开始遍历，直到候选列表不再变化

初始设置4个簇，每个簇8个点，随机分布。
最终聚类形成3个簇（存在误差）

使用K-means聚类？进行改进



自底向上的层次聚类

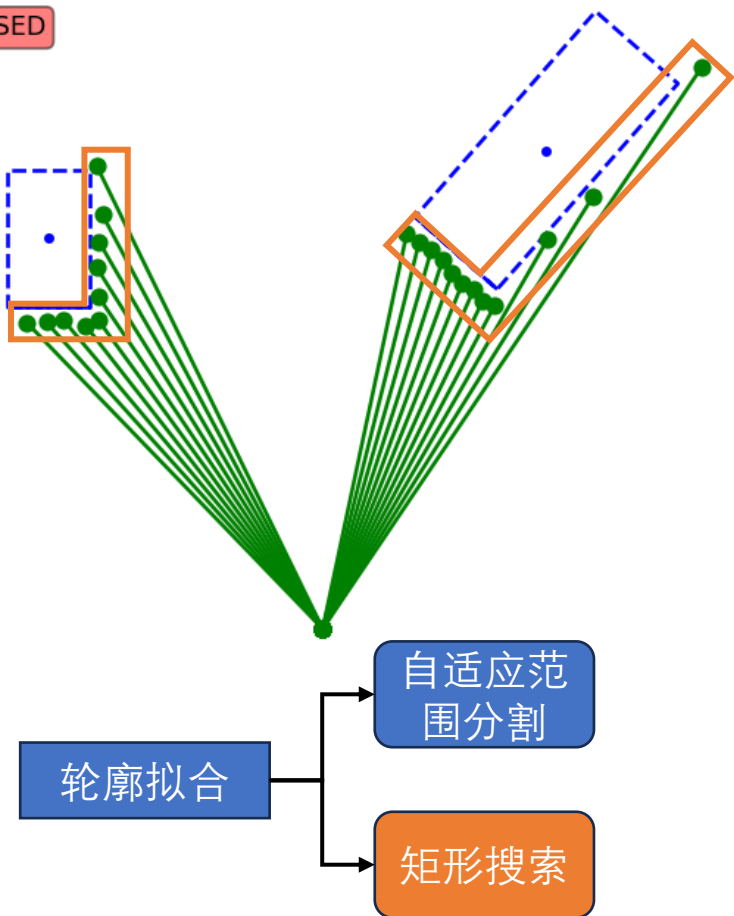


Part 2 | Mapping (static)

» 建图算法 (静态) --几何轮廓识别

矩形搜索

PAUSED



自底向上的层次聚类

基于每个聚类簇的观测点云，拟合矩形。

- 通过固定角度，依次旋转点云到 $\pi/2$ ，寻找与坐标轴平行的情况。
- 对于每次旋转，基于评估条件估计拟合误差cost。
- 更新最小拟合误差cost，确认最佳旋转角度。
- 计算最佳旋转角度下的矩形。

矩形类：不同于传统保存四个顶点

```
class RectangleData:
```

```
def __init__(self):  
    self.a = [None] * 4  
    self.b = [None] * 4  
    self.c = [None] * 4
```

4条边的参数, $a_i x + b_i y = c_i$

```
self.rect_c_x = [None] * 5  
self.rect_c_y = [None] * 5
```

4个顶点，闭合性是，最后一个重复第一个，形成闭环，包含顺序信息。

是一个典型的 用空间复杂度 换取 计算复杂度 降低的方式

实验：尝试在真实数据上 000004.bin 进行车辆检测。

提示：如何消除地面点云干扰（问问gpt）？

开始实验



北京航空航天大学
BEIHANG UNIVERSITY