总结：

第一章

- PCA：
  - 去中心化数据向主方向上投影，希望投影坐标的方差尽可能大，区分好
  - 瑞利熵证明，等价于求原数据协方差矩阵的，最大特征值和特征向量
  - 减掉第一个主方向后，得到的主方向就是原来第二个特征向量（主方向）

  - encoder：得到的是点在各个主方向的投影（坐标/系数）
  - decoder：降维后的数据重构回原空间，需要再次用投影矩阵对当前数据做线性组合变化

  - 拓展：
    - KPAC：
      - 低维空间数据不能线性划分，所以通过升维之后，再进行PCA降维；
      - 升维之后的协方差矩阵H，表示为对原数据的一个核函数（隐式掉升维的函数），只要考虑核函数就行，实践中根据经验选取不同核函

- 计算normal：
  - 假设有个中心点，每个点到中心点都形成一个向量，希望法向量和这些点形成的向量的内积之和尽可能的小；
  - 另一角度，可以认为是将点向法向量做投影之和最小，**所以同样可以用PCA来实现，最小的主方向即作为法向量方向**
  - 降噪：
    - 选点集可以用KNN或者半径（radius）
    - 特征加权
    - RANSAC（第四章）
    - 深度学习

- 体素滤波：
  - 减少点的数量，深度学习中基于voxel的方法会从voxel中提取特征送入下一层

  - 方法
    - 降采样：用体素网格划分空间，每个空间输出一个点作为降采样结果
      - 输出方法：
        - centroid：平均；效果更好，平滑，但是慢
        - random：随机；效果一般，但是快
    - 哈希表近似：可能会导致两个很远的点被判断放到同一个体素网格容器，导致输出的点异常，且数量会比较多；
      - 提前设想好有几个voxel作为输出，因为实际很多voxel都是空的，

所以留够用数量就可，减少计算量

第二章

- 数据结构
  - KD-Tree：
    - 在任意维度都可以！原理同二叉树一致，所以在每个维度，都跑一次二叉树，就OK了！
    - ⚠️：比BST稍微复杂在于，每个节点可以包含很多内容！对于BST最后一个节点肯定只有一个值，但是KD-Tree不同，可能包含多个值，即足够细就不继续切了～

  - Octree：
    - 专门为3D数据设计的，是一个立方体～每一维度都切一刀成两半～
    - 1D二叉树，2D四叉树，3D八叉树

    - 好处：
      - 可以提前终止搜索！
      - KD-Tree每层实际上只考虑了一个维度，所以要回溯root遍历所有可能区间；
      - 而八叉树则对3个维度都做了限定～一旦限定在某个小立方体内，就不用考虑其他！

- 最邻近搜索：
  - KNN
  - Radius-NN

第三章 # 聚类

- EM算法：

  - 概念：
    - 先验概率：统计得到，大因，事情发生前的预判概率
    - 后验概率：事件发生后求的反向条件概率；或者说，基于先验概率求得的反向条件概率。概率形式与条件概率相同
    - 条件概率：一个事件发生后另一个事件发生的概率。一般的形式为P(x|y)表示y发生的条件下x发生的概率

    - 定义
      - 给定模型参数（分布）下，求数据点的概率，参数根据最大似然估计来解出；
        - 对GMM，参数就是【pi、均值、方差】；

    - 步骤：
      - 1. 初始化；

        - 2. 【E-step】：根据参数分类/求概率
            - 根据给定数据点、参数计算条件概率（GMM里是后验概率）；
        - 3. 【M-step】：迭代求参数MLE
            - 跟GMM不同，不是直接优化原始目标函数的最大似然，而是【构造一个Q函数】，是原始函数的对数值对z的期望，可以证明优化Q可以间接优化原始函数；
        - 4. 迭代参数；
    -

- K-means：
    - E-step：最小化类内距离，即min 代价函数J（点到聚类中心到距离）
    - M-step：对J求导，更新聚类中心

    - 改进算法：K-Medoids：
        - 中心点的选取不再采用均值，而是从原来数据点中取到类内所有点距离之和最小的点，减少噪声的干扰；
        - 另一方面，欧式距离不适用于类别标签；
            - 例如狗和猫平均的结果无法解析，这里用类似投票的方法找到最具代表的标签更合适；

        - E-step：与之前类似，只是不再用欧式距离，但是可以离散表示，比如猫与狗的距离是1，狗与狗距离是0；
        - M-step：因为不能求导，所以转换为一个O(N-k)的查询问题，记录K_th类中，每个点到其他点的距离之和作为损失函数；
    - 应用：
        - 图像压缩
        - 点云压缩（不能压缩xyz 坐标，只能压缩颜色而不是像素（分辨率））
    - 问题：
        - 主要问题是K不知道；
        - 对噪声很敏感；
        - 非0即1（hard assignment）：不能说一个点百分之几属于一个类，百分之几属于另一个类（最好是给出用一个置信度）；——>GMM

    - 优势：简单、快；
    - 劣势：
        - 1. 基于欧式距离，各个方向方差一致，可视化就是数据分布是一个圆，但实际中很多并不是这样子。
        - 2. K不可知 3. 对初始化、噪声都很敏感（可用K中心点缓解一下）

- GMM
    - 混合高斯模型——每个类都用高斯分布来描述！
        - 这样就可以给出每个点属于一个类的概率，克服K-means的问题之

一；
        - 更像是一种对数据表示的建模；
        - 数学上表达成K个高斯模型的线性组合，K也是人为定的
    - 算法：
        - 初始化【均值、方差矩阵、概率】：
            - 以2个高斯模型为例，可视化为两个圆，不是两个中心点，圆是因为假设每个方向点方差一致，权重初始化为0.5，每个类权重一样；
        - 2. E-step：
            - 计算点属于某个类的概率（后验概率），K-means里计算的是是否属于某一个点，一样只是这里取的概率而已；
        - 3. M-step：
            - 用MLE，算高斯模型的三个参数，固定两个，优化一个求解；

    - 优势：soft，知道属于每个类都概率，对噪声更加稳定；
    - 劣势：
        - 1. K不可知；
        - 2. 虽然没有假设各个方向方差均等，但是也是一个高斯模型，所以可视化在空间里数据分布是个椭圆，实际中很多数据也不能以椭圆方式建模；
        - 3. 有奇点问题，可能一个高斯模型会坍塌成一个点，只拟合了一个点，导致方差为0，MLE无限大，工程上加几个if判断一下是否掉入奇点就可以避免；

- 总结：EM > GMM >K-means：
    - EM是通用优化方法，优化p(x|theta)，同时引入z帮助优化MLE；
    - GMM是基于线性高斯模型组合的EM算法，可以认为是EM的一个应用，解出三个模型参数，进而计算后验概率r(z_nk)；
    - K-means是GMM的一个特殊情况：后验概率函数r(z_nk)退化成r_nk，GMM方差等于0，各个方向都一致；

- 谱聚类：不规则数据分布下聚类，K-means和GMM都假设在欧氏空间

- 工作在Graph上，关心的是点与点的【连接性】～而不是【距离】！
    - 距离很远的点，根据连接性可能会是一个类；
    - 距离很近的点，根据连接性可能反而不是一个类；

- 基于拉普拉斯矩阵处理的K-means，数学上基于图论，而不是GMM或EM；

- 步骤：
    - 建图&相似矩阵W：假设结点到自己的相似度为0，3中主要方法
        - 1. 每个点在半径范围内建立连线；

        - 2. KNN-Graph，只选最邻近K个点建立联系

        - a. 只要有一个是对方的KNN就建立连线；
        - b. 双方互为KNN才建立连线，权重是相似度；

    - 3. 每个节点都有连线，全连接
        - 权重 / 相似度：距离的导数 or（常数-距离）等，
        - 即只要越近相似度越大就OK；

- 对角矩阵D：W里对应行之和
    - 物理意义：第i个结点连接的权重之和；
- 拉普拉斯矩阵L = D- W

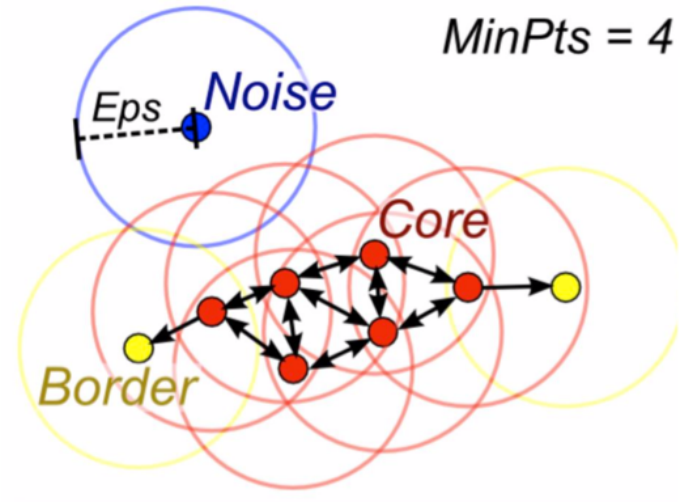
- 无归一化SP：
    - 1. 初始化W，L；
    - 2. 找L的前K个（最小的）特征值对应的特征向量
        - 有点类似PCA升维啦，K个轴，对每个点X重新投影到K个轴，得到升维后的点 Y（K维向量）
    - 3. 对Y做K-means（低维不足以区分，升维后区分），Y对应对类就是相应的X对应的类，一一对应；
    - ⚠️：K=聚类数量=Y维度=特征值/向量个数

- 归一化SP：区别在于拉普拉斯矩阵用的归一化的L'=L_rw

- 优点：
    - 不对类的形状假设，工作在图论/连接性而不是欧式空间；
    - 可以对任何维度数据操作
        - 因为X->Y，对数据进行升维或者降维处理，所以数据在什么维度就没关系啦～
        - 但是有些算法对维度是敏感的，例如下节课的DBSCAN、Mean-Shift；
    - 可以自动发觉有多少个类；

- 缺点：复杂度很高！

- DBSCAN：基于密度、连接



Red: Core points. point number within circle ≥ 4

Yellow: Border points. Still part of the cluster because it is within r of a core point, but does not meet the min_points criteria

Blue: Noise point. Not assigned to a cluster.

Advantage

- No assumption on cluster shape
- Automatically determines cluster numbers
- Robust to outliers

第四章 # 模型拟合

- 最小二乘法：
  - 只要有平方、min 就是；
  - 问题：若f(x)非线性，如何求解？优化方法。。。
  - 优点：简单快速，适用没有噪声或很少

- 霍夫变换：
  - 欧式空间的点->参数空间的线：
    - 在参数空间求线的交点，就可以得到精确的参数！
    - 实际上，线可能很复杂，很难求交点——对参数空间划分（分辨率自定），固定一个参数，求解另一个（floor），锁定一个格子投票，最后遍历每个格子，票数最多的视为交点；
  - 优点：
    - 1. 最噪声稳定，只要噪声数量低于inlier，一定能投出最高的；
    - 2. 图形缺少部分点，也OK（比如一个圆形的点，只需要部分点就可以确定3个参数）
    - 3. 通用的，只要能找到投票到方法，就可以对任何模型处理；

- 缺点：一般只用在2/3D，不能放缩，参数多空间很大；
  - RANSAC

- RANSAC:
  - 受限于inlier；

  - 直线拟合：
    - 1. 随机选一个sample，直线选2个点（两点确定一条直线）
    - 2. 解直线模型（这里使用参数模型，其他也OK），注意这里额外规定n = [a, b]T为单位法向量；
    - 3. 算consensus，即其他点是否支持这条线（与n作点乘，即投影得到该点到直线的距离，近支持远不支持）；（这里是点法式）

3. Compute error function for each point

$$p_i = (x_i, y_i)$$

$$d_i = \frac{n^T(\mathrm{p_i} - \mathrm{p_0})}{\|n\|_2}$$

    - 4. 设定一个阈值，小于阈值表示支持，统计支持的点数，包括sample的点；
    - 5. 重复步骤1～4，选支持点最多的那个模型；

  - 平面拟合：

  - 参数设定：
    - 1. 阈值：以经验/实验设定，或者用卡方分布（X^2）得到一个理论上的值，但是不靠谱，需要的信息太多了；
    - 2. 迭代次数N：选择N根据概率p，至少一次随机采用完全inline的概率是p = 0.99

Iteration number N is given by

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

Table for $p = 0.99$

	proportion of outliers e						
s	5%	10%	20%	25%	30%	40%	50%
2	2	3	5	6	7	11	17
3	3	4	7	9	11	19	35
4	3	5	9	13	17	34	72
5	4	6	12	17	26	57	146
6	4	7	16	24	37	97	293
7	4	8	20	33	54	163	588
8	5	9	26	44	78	272	1177

  - trick：
    - 1. 提前终止：当某一次的inline比例达到/超过预期的(1-e)，就证明已经拟合很好了；
    - 2. 最小二乘refine一下参数，选出模型和inlier点后，因为每次只选两个点肯定效果不是最优的；

- 优势：
  - 1. 简单通用；
  - 2. 实际工作好，即使inline率很低；
- 缺点：
  - 1. 需要定参数阈值；
  - 2. inline 率低的话，需要迭代次数多，时间多；

第五章 # 点云深度学习

- 方法分类：
  - 网络角度：
    - 3D卷积
    - 多个2D投影 + 2D卷积：工程上难以拓展，投影次数太多，计算速度很慢；
    - 化为一维向量：N*3 -> 3N 向量->投入MPL（很少使用）：
      - 没有数列不变性：输入分量顺序影响输出，pointCNN 提出打乱顺序输入让网络自适应，1000个点有1000！排列，不合理；

- VoxNet（IROS 2015 ）：ModelNet40
  - 思路：
    - 网格划分，每个voxel放的内容可以自己设定（本文是概率）
    - 3D卷积—>摊平—>MLP之类的输出分类；
  - 评价：分辨率影响大，工程难以拓展，特别是自动驾驶的场景带不动分辨率

- MVCNN（ICCV 2015 ）：ModelNet40
  - 多视角投影2D分别送入多个CNN，在MaxPooling坍塌成一个vector送入最后的CNN多分类
  - 工程难以拓展，投影次数太多，很慢！

- PointNet：物体识别/语义分割
  - 思路：share MLP + Max Pooling，容易拓展，效果也好
    - Max Pooling：
      - 数列不变性：对噪声不敏感！
      - 坍塌后的feature vector 只与critical points有关
        - deep dream 风格迁移、特征提取可以借鉴
  - 证明：PN可以模拟任何函数，当然也可以模拟识别；
    - 得益于voxel downsampling：只要分辨率（feature维度）够高，每个点都可以模拟到独一无二的feature
    - Max pooling：feature融合形成global feature
      - 当分辨率足够高，可以近似重构点的feature
    - MLP：已经证明可以模拟任何函数，则可以模拟点集映射识别的函数！

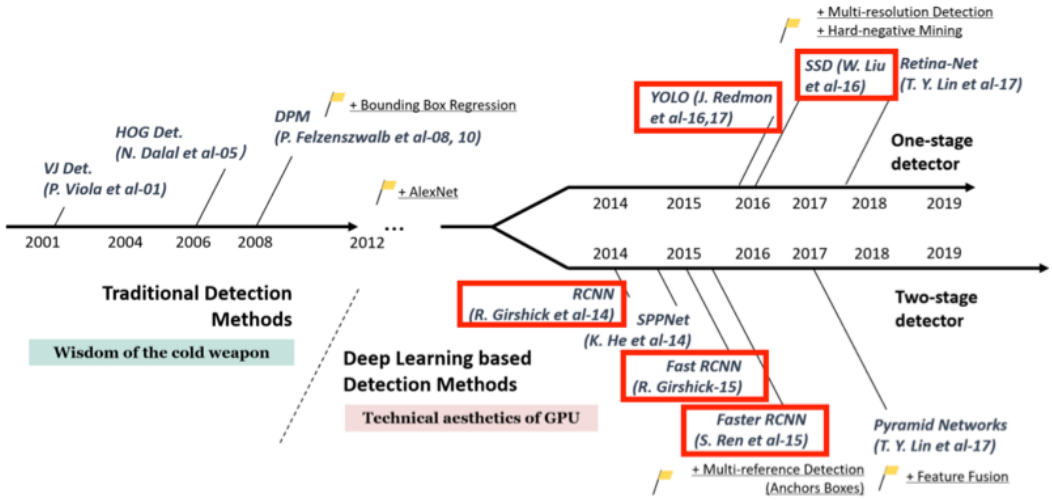
  - 为什么要搭建网络而不是直接黑箱MLP?
    - 个人认为正是传统机器学习和深度学习的区别之一
      - 传统是特征工程为主，手工提取特征，然后送入分类/回归器等；

        - 深度学习是表示工程，通过网络结构细化表示方式/模块，更有方向学习，易于拓展，加深网络（效果越好）

- PointNet++：
    - 思路：在PointNet的基础上，借鉴了CNN多层特征提取的思想！
        - encoder：
            - set abstraction，级联的进行 sampling-grouping-PointNet 提取set的特征，达到逐层优化特征提取的效果；
        - classification：
            - 提取的特征经过一个PointNet/MLP，最终输出一个result vector，得到分类结果
            - 改进：
                - multi scale grouping（MSG）：不同半径RNN搭配不同PN，多尺度特征融合
                - multi resolution grouping（MRG）：多层特征融合
        - segmentation：decoder（分割要精确到点级别）
            - interpolation（插值）：插值周围邻居的feature + 跳跃连接

第六章 # **目标检测** 综述

- 概述：
    - 2D物体检测（3D由2D 拓展）
        - 物体检测的2个目标：直接导致RCNN 系列的诞生
            - 定位：先找出物体在哪里，不关心是什么
            - 分类：物体是什么类别
        - 评价标准：
            - IOU
            - precision & recall
            - 平均准确率AP：对于每个类别所有图
            - 终极标准mAP（对所有类别AP平均）
            - P-R curve
            - NMS
        - 发展历史：



- 3D物体检测：
  - voxel-base（空间分割）：Voxel-Net、PointPillars
  - 原生与3D点云：Point-RCNN
  - 3D 物体检测表示：
    - Input: Point cloud and/or images
    - Output:3D bounding boxes-[x, y, z, length, width, height, heading(朝向)，category]
      - 方向本是3个roll，pitch，yaw，但是在自动驾驶假设物体在水平面上，所以只有一个方向

- 深度学习物体检测

  - 2D目标检测：
    - 多阶段法：RCNN系列：
      - RCNN：
        - 生成候选区域：Selective Search
        - 处理候选区域：对锚框进行放缩到同样大小
        - 特征提取：每个框调用一次CNN
        - 分类：SVM
        - 位置精修：回归

      - Fast-RCNN：Selective Search + Fast-RCNN
        - 提取整个图片共享的特征图
        - **ROI pooling 候选区域特征：**
          - 不再对每个候选框调用CNN，而是从共享的候选框上面对应的扣
          - pooling是为了把不同空间大小的特征放缩到统一大小：
            - 可以统一的维度送入后面 全连接层（FC）**分类和回归**
            - 维度一致，可以batch处理
        - 生成候选区域仍旧费时

- Faster-RCNN：RPN + Fast-RCNN
  - **RPN取代了Selective Search**：包括 **二分类 + 回归**
    - 用神经网络来提取锚框，只用设置size和长宽比，自动生成
    - feature map与后面ROI pooling的共享
    - feature map 上的每个点，都对应**原图中**一组规格的锚框
    - 生成的锚框送入 二分类器，筛选出前景锚框，送入回归矫正，得到真正的候选框
  - 候选框送入 Fast-RCNN，多分类器和回归候选框
  - 真正的端到端训练

- Mask-RCNN：主要是用于多任务，目标检测和实例分割
  - ROI Align
  - 联合训练：目标检测 + 实例分割

算法名称	R-CNN	Fast R-CNN	Faster R-CNN
候选区域	Selective Search	Selective Search	RPN
分类方式	SVM	SoftmaxLoss	SoftmaxLoss
回归方式	边缘提取	SmoothL1Loss	SmoothL1Loss
重复计算	非常多	较多	比较少
训练方式	每个步骤都是独立训练	除候选区域生成外，其他步骤是端到端训练	端到端训练
检测速度	~0.002 FPS	~0.5 FPS	~5 FPS

- 3D目标检测：

  - 四种思路：都是one-stage、two-stage 只是使用的工具不一样（卷积工具）
    - 1️⃣2D方法：点云投影到不同方向/视角，得到不同图像，用传统方法处理
    - 2️⃣基于网格：voxel grid -> 三维网格，经过pointNet或者3D卷积方法变成2D feature map
    - 3️⃣基于点：直接用PointNet等原生的点云卷积 ，直接实现one-stage、two-stage
    - 4️⃣点云和图像融合：例如点云投影到图像、图像颜色赋予点云

  - 转化传统方法：

- MV3D：最早的点云物体检测；不好拓展，有更好的方法，不再使用
  - 先从俯瞰图过CNN得到3D proposal，再分别投影到不同视角（俯视、前视、RGB视角）
  - 用3D proposal的投影试图，代替Faster-RCNN的RPN，即候**选框生成阶段！**
  - 总之，等价于=不同视角的proposal 融合 + Faster-RCNN

- 基于voxel：
  - VoxelNet：
    - 核心：
      - 3D卷积+reshape获得类2D的feature map；
      - 一个网络只对一个类别（类RCNN，无ROI Pooling 部分）

    - part-1：建立voxel grid（D*H*W）
    - part-2：得到每个cell的feature（C*D*W*H）——PointNet
      - hash 映射（大多voxel是空的，全部过一个VFE网络浪费时间，通过点到K个容器，只需K个非空的容器VFE，大大提高效率—>得到（128*10*400*352）的encoder；
      - 128是特征维度，其余为网格数量；
    - part-3：进行3D卷积—>reshape到2D feature map，压缩D维度到1（C’* H’ * W’）
    - part-4：RPN（来自Fast-RCNN，用于生成候选框定位的），分类（只是2分类，不是多分类，没有ROI Pooling）和回归

  - PointPillars（点柱）：
    - 对VoxelNet 中part-2（得到cell的feature，3D柱状，后送入3D卷积得到2D柱状feature map）的改进，直接在2D划分柱子；
    - 送入SSD（基于锚框的单阶段法）进行处理
  - 总结：两个方法都是单阶段法，因为没有ROI Pooling

- 基于Point：原生于点云的网络
  - PointRCNN：
    - 整体思路和Faster-RCNN很像！
    - 两个主要阶段：
      - PointNet++：骨干网络，类似于提取特征图，这里针对每个点提取feature vector；
      - stage-1：RPN-per-point:
        - 前景分割：分类问题，只对前景点（指在认为在GT里的）生成3D候选框
        - bin-based 3D候选框：先分类（低精度），后回归（高精度）

- stage-2：ROI：
  - enlarge bbox，获取候选框周围的信息
  - feature和spatial信息融合送入MLP，多分类+refine回归

- fusion：
  - Frustum PointNet：和PointNet一个作者
    - 图片+点云
    - 架构：
      - image 2D proposal-->点云投影到2D，抠出2D proposal里的点（N*C），即只取proposal 对应的Frustum里的点—>送入PointNet提取feature
      - 3D 实例分割，分割Frustum里的点云，得到物体的点云特征（M*C）
      - 3D bbox的回归：对物体点云feature

    - 评价：效果并不好，实际很少使用
      - 如果mask-rcnn不是完美的，漏了某些物体（proposal），那这个网络就不会再去考虑这个物体
        - 即一旦出错，点就会丢失
      - 工业上：很难把图像和点云对齐，基本上不可能！
        - 相机和激光雷达不是重合的，会有视差问题
        - 捕捉到的数据不是同一时刻

  - PointPainting：处理F-PointNet中点丢失的问题，即简单弱结合image！
    - 思路：对图像做图像分割，点云投影到图像上获取label，增加到原始点云数据上，后续接各种点云目标检测网络
    - 评价：
      - 虽然同样有视差问题，但是这个一种弱结合，即使出错了，也没什么，后续经过PointPillars或者PointRCNN可以修正
      - 极端下视觉信息完全失效也是可以得到靠点云的网络或者比较好的结果的

  - 总结：F-PointNet，完全信任依赖image的建议（proposal），而PointPainting只是锦上添花，错了也无伤大雅！

- 总结：
  - 点云常用数据增强：
    - 点云旋转：
      - 有意义！可以增强泛化能力
      - 一般不会旋转太多，否则看起来不真实

    - GT的bbox 做平移或旋转：

- 不能旋转太多，如果太多就违法物理规律！雷达扫射位置与bbox中检测到点云的位置有关

- loss类别不均衡：
  - focal loss：将注意力转移到分类错的类别！当分类正确，（1-p）很小，权重就小，反之很大；

第七章# 特征提取（关键点检测）

- 图像特征（2D）：有趣/关键的点
  - 角点
  - 一块一块的东西也可以，当然最常用的是一个像素/点

  - 图像特征点：
    - Description：encoder——对特征点进行描述
    - Matching：对两个点描述的相关性
      - 例如：利用匹配，进行构建全景图

  - 应用：
    - SLAM（同时定位与构图）
      - 基于image feature的SLAM，其实是对特征点进行三维点云重建，仅仅对于特征点，故而往往比较稀疏；（找到特征点的深度（构图），相机的位置（定位））

- 点云特征点（3D）：
  - 应用：
    - 点云配准/匹配（registration）——进行场景重建构图：找旋转、平移矩阵 R、T
      - ICP（迭代最邻近点）：要求比较好的初始矩阵、2点云足够高的重合率
      - 特征点的提取、描述和匹配
    - 物体的定位：给定物体模型，找到位置 6D姿态估计
      - 假设物体是刚性的，也是一个匹配问题
      - 对模型提取特征点，对场景中的物体进行特征点匹配
    - 数字人驱动：人的照片/视频，讲表情转到3D数字人
      - 提取人的特征点进行匹配
      - 但是人（姿态/表情）是非刚性的，很难

- Detector：
  - 传统方法
    - 2D及其衍生：
      - Harris：协方差矩阵法，接近光流法；
        - with intensity：cost function E(u,v)表示点云的强度变化，是关于一阶导的协方差矩阵M的二次型
          - 2D图像：
            - x、y的关于intensity的一阶导的协方差矩阵M

        - M做特征值分解，得到主方向的投影，也是就是两个方向的变换强度（而边、角点在水平/竖直是有强度变化的）
        - 即特征值够大，证明在该方向有高强度变化！有边/角点！
    - 3D点云：
        - 类似2D，但是一阶导难求，因为对点云空间用网格划分来求很蠢，而且很麻烦，这里直接用代数求解，只要大于3个点（不相关），就有解
        - 即当邻居点足够多，就可以求解出一阶导e，进而求出3D的一阶导协方差矩阵M
        - 对e可以优化：求local surface的法向量n，对n投影，再用e减去得到优化后的一阶导e’（认为与面相切更好看，减少噪声的干扰），即 e’ = e-n(n^Te)

- no intensity：cost function表示的不是点云的强度变化，是移动后，patch中所有点到其原点的近似local surface的距离；
    - 同样可以转化为协方差矩阵M的二次型
    - f(x, y, z)=0 表示点p处的local surface，这里用平面来近似，即 f = a*x + b*y + c*z = 0，单位化则系数是法向量，即 n_x*x + n_y*y + n_z*z = 0；
    - 这里的协方差矩阵M是关于表面法向量分量的，而不是强度变化的一阶导分量；
    - ⚠️：对于表明，需要在三个主方向上都有较大的变化速率，才能保证该点的local surface足够奇怪，即可能是角点！

- 6D：intensity + surface：6D的协方差矩阵！
    - lambda_3：只要有三个特征值足够大，就认为是角点，约束太松了，和without intensity 没有区别；
    - lambda_5：即要满足几何上的三个特征值足够大，又要满足强度的两个足够大，太严格；
    - 折中取lambda_4！

- SUSAN：略
- SIFT：略

- 原生于3D：
    - ISS（Intrinsic Shape Signatures ）：
        - 核心：特征点周围点在三个方向的分布变化比较大，例如在平面上的点，z 方向上的变化为0；
        - 实现：对点P及其领域做PCA，只要最小的特征值够大，认为是特征值
        - trick：用于提升perform：
            - 加权的 点的 协方差矩阵，距离P越近的点，有比较大的权重
            - 多加几个限制条件：
                - NMS with lambda_3，本来目的也是想找

lambda_3（最小特征值）比较大的点！
- 要求三个特征值从大到小递减，不含等于（即平面和直线的点排除）

- 总结&缺点：

	Input	Covariance Matrix	Criteria	Intuition
Harris Image	Image	Intensity gradient	λ_2 is small	Intensity corners
Harris 3D	PC with Intensity	Intensity gradient	λ_2 is small	Intensity corners in local surface
Harris 3D	PC	Surface normals	λ_3 is small	Corners in 3D space
Harris 6D	PC with Intensity	Intensity gradient + surface normals	λ_4 is small	Corners in either 3D space / local surface intensity
ISS	PC	Weight point coordinates	$\frac{\lambda_i^2}{\lambda_i^1} < \gamma_{21}, \frac{\lambda_i^3}{\lambda_i^2} < \gamma_{32}$	Point distribution is different in 3 dimensions

  - 缺点：对噪声敏感/传统处理不了 ----> 引入深度学习
    - 例如kitti，检测出来的多是噪声点没啥用
    - 规则模型加入噪声之后，结果变坏，鲁棒性差

- 深度学习：没有通用数据集，很少做！
  - 问题：
    - 特征点 很难定义->难标注->数据集少
      - 比如特征 和尺度也是有关系的：花纹近处观察是特征，远处可能就不是了
    - 其他点云处理难点：旋转、稀疏性...

  - 2020 之前的方法：
    - USIP（Unsupervised Stable Interest Point Detection）：无监督
      - 核心：
        - 1.关键点无聊从哪个角度，都是关键点
        - 2.关键点的定义 与 尺度有关（scale/level of details）
          - 大尺度：物体中心是关键点
          - 小尺度：观察轮胎表面，轮胎纹理是关键点；不是关键点，观察车子；

          - 退化问题：大尺度特征（也是特征点）
            - 特征点都是中心点（不会随旋转平移变化），loss都是0，网络只会输出中心点！
            - 特征点集中在主向量上（PCA，不会随旋转平移变化，ISS原理）

          - 解决方案：只有“看到”整个物体，才能得到主向量和中心
            - 限制FPN感知域大小：感知域太大就会输出大尺度上的特征点
            - 文章里用So-Net做FPN 效果比较好，对感知域建模比较容易，且对点云密度自适应，不那么敏感

- 网络架构：FPN 不重要，文章用的So-Net，其他也可；
  - 点云->FPN -> 得到特征点和其置信度/不确定性（Q & sigma）
  - 点云+T（旋转、平移）-> FPN -> 得到特征点和其置信度（Q～ & sigma～）
  - 由核心1知，如果是特征点，应该只相差一个矩阵T
    - loss1：对Q～ & sigma～用T进行旋转、平移，与Q & sigma越相似越好
    - loss2：由于FPN没有保证提出的特征点一定在点云上，所以希望特征点与原始点云的点距离越近越好

- 补充：SO-Net（backbone，点云特征提取）介绍：
  - SOM：k-means的升级，更好控制感知域；
  - point2node：自适应密度，更好的切割（grouping）点云，还能控制每个点被选取的次数

- 3DFeat-Net，弱监督：略

第八章 # 特征描述

- 传统方法：

  - 基于直方图：
    - 概述：
      - 通常按照某 一个属性 进行**等级划分**并统计，比如按照距离
      - 缺点：
        - 不考虑点的绝对坐标，对镜像等变化不能区分
        - 不能捕捉 空间/结构 信息

    - PFH(Point feature histogram)：
      - 核心：
        - 6D-Pose independent：不依赖于平移旋转：
          - 考虑每个点对之间的相对关系值，没有旋转/平移的问题；
        - 捕捉邻近的表面变化：借助法向量，两点连线的向量
          - 根据法向量n1、n2和点p1、p2计算4个属性值
          - 两点相对距离d 通常不用：点云数据，雷达越远越稀疏，距离作为属性，对同一个物体可能对特征的描述不一致；即收到位置的影响
      - 算法流程：
        - 计算3D 数据点的特征描述 三元组
        - 每个维度划分B bins，将三元组看作三维空间的点（比较反直觉，复杂，FPFH中改进）
        - 3D voxel grid，将三元组投入
        - flatten voxel grid 成一个 B^3的array——PFH特征向量
        - Normalize, e.g., sum/norm equals to some value
      - 应用：
        - 传统的语义分割：per-point PFH + SVM -> per-point

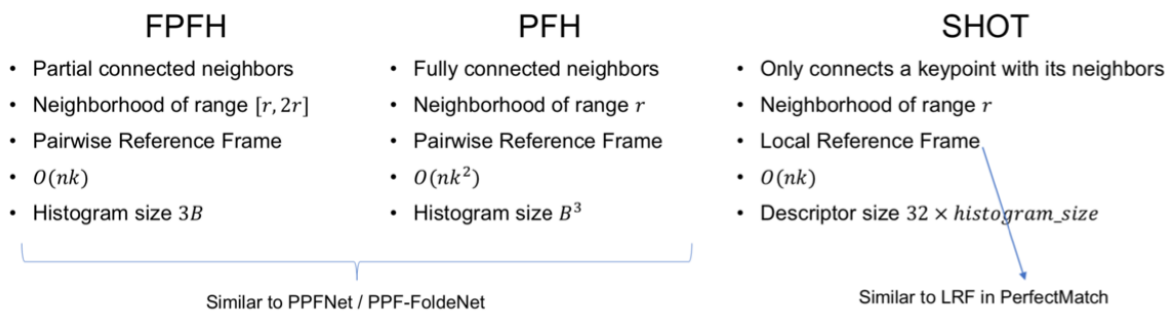
classification -> 分割；

- FPFH：PFH的改进版本
  - 核心：
    - SPFH（Simplified）：
      - 计算特征点和其邻居的单向PFH
      - 计算其邻居的和它们邻居的单项PFH
    - FPFH：自己的SPFH + 邻居们的SPFH加权平均；
    - 输出三个直方图的连接

- 基于信号（signature）：
  - 概述：
    - 考虑局部的信息计算几何度量：
      - 例如划分四个象限，计算每个象限点的平均坐标（共保存2*4个值）
    - 缺点：对旋转敏感
      - 当对象旋转后，不能保证划分的计算结果仍然不变

  - SHOT（Signature of Histograms of OrienTations）：既有signature又有histogram部分
    - 核心：
      - 6D-Pose independent：
        - LRF（Local Reference Frame）：
          - 核心：根据关键点的r邻居，用PCA 建立坐标系，保证旋转不变性
          - 实现：类似ISS，建立加权协方差矩阵，计算主向量并确定方向（特征向量有两个方向）
      - 划分空间volumes，每个volume构建直方图：
        - 问题：boundary effect
          - 硬分割：
            - 噪声点扰动，会导致完全不一样的投票结果，不稳定；
            - PCA得出的LRF也会被扰动；
        - 线性插值软投票：经、纬、半径；volume中的bin，四个方向上的团投票

- 🆚总结对比



- PFH/FPFH 效果差不多，但是一个O(nk^2)，一个O(nk)，通常选FPFH
- 深度学习的灵感往往来源于传统方法！PerfectMatch 目前SOTA 达到

90%以上准确

- 深度学习：基于 voxel grid / 基于点
  - 意义：
    - 传统方法都是基于 **特征点的几何信息**
      - 表面法向量的变化
      - 点之间的距离
    - 对下述情况 **效果很差：**
      - 噪声
      - 遮挡
      - 点稀疏
    - 深度学习：
      - 加入一些语义信息进行学习
      - 更好的encode geometry
      - 对噪声robust
  - 基于voxel grid：3D卷积
    - 3D Match：
      - 核心问题：
        - 怎样去建立数据集：利用RGB-D多视角frames 重建
          - positive：
            - 相同的物理位置，但是来自不同视角
            - 随机取距离anchor patch 很小偏移的patch
          - negative：其他位置随便取啦
        - 怎样定义相似和不同（设计loss）：
          - Contractive Loss
        - 问题：对旋转敏感！
    - Perfect Match：3D Match的改进加强版！
      - 核心思想：
        - Triple Loss
        - LRF（来源于SHOT传统方法，主要贡献⚠️）
        - SDV（贡献不大，无伤大雅）：新的输入数据表示方法；
          - 提取关键点patch的方法，3DMatch 是TDF
      - 流程：
        - 兴趣点法向量
        - LRF（和标准坐标系没对齐）
        - canonical representation（标准对其）
        - SVD（高斯平滑kernel，最终的patch表示）
        - normalized SVD 后作为3D卷积的输入
        - triple loss 训练
  - 基于 point cloud：类似PointNet 直接工作在点云上；
    - PPFNet：
      - 结构：基于点处理的网络，PointNet提取特征，concat global特征—>MLP—> final feature；

        - 创新点：
            - 不仅用了patch特征，还用每个frame的global来增强；
            - 输入是PPF（Point-Pair-feature）：除了点的信息（N*3），还加了很多别的信息（M*3，M>N）
                - 法向量；
                - 动机来源于SPFH传统方法中的三元组：一系列角度关系，法向量，两点连线；
            - N-tuples Loss：考虑多个patch之间的关系（3D match/perfect match 只考虑 二元组a+p/a+n 或三元组a+p+n）
                - 相似矩阵M（patch间物理距离）：N*N，1表示距离够小，用作指明 positive；
                - 特征空间距离矩阵D（description空间距离）：N*N，值是patch的description vector，用于计算相似度；
                - 效果：positive和negative之间的区分度更大了（给出了实验图），毕竟加了global信息；
    - PPF-FoldNet——Descriptor with Auto-encoder:
        - 动机：能不能无监督的学习，假设不能获取点云间的相对位置！
            - 3DMatch/Perfect Match/PPFNet 所采用的loss都是基于一个前提；
            - 即，数据集能够已知两个点云的相对位置（T），才能进行positive/negative的配对
        - 思路：如果encoder的vector能够decoder出原始的数据输入，就可以认为这个vector是输入的description
        - 输入：这里的输入是先将patch点云提取出PPF，直接作为输入（不包含点坐标和normal，其对于旋转不稳定，相当于丢掉了点的绝对信息）
    - 总结：

	Representation	Loss Function	Rotation Handling
3DMatch	Voxel grid	Contrastive Loss	No
PerfectMatch	Voxel grid	Triplet Loss	Local Reference Frame
PPFNet	Coordinates + surface normal+ PPF	N-tuple Loss	No
PPF-FoldNet	PPF	Reconstruction Chamfer	Rotation invariant

第九章 # 点云配准

- ICP（Iterative Closest Point）
    - 用一个优化问题来定义
    - 核心：
        - 采样点集——数据关联（点的描述、配对+去outlier）——计算（点-点、点-面）R、T的目标loss函数——迭代
    - 改进：特征提取和描述的pipeline方式可看作ICP的一种情况！
        - 点子集的采样方式：NSS（关注表明法向量）、特征提取（detector）等；

        - 数据结构：最近邻搜索的加速-树结构、特征描述（description）等；
        - 去噪：outlier
        - loss函数优化：通常是点对点（可能实际不匹配的对强行配对了），可用点对面（点对应一个面，可以对面上的其他点对其，扩大了一个点的覆盖范围）
    - 总结：
        - 简单；
        - 但需要好的初始化R、T；

- NDT（Normal Distribution Transform）：最大似然估计——最优化问题
    - ICP存在的问题：
        - 每个点只考虑了对应点的信息，没有考虑邻居（只有Point-to-Plane做SN时考虑了一点）
            - 怎么用概率表达邻居?
        - 要计算很多heavy association，比如NN，点多时很慢！
            - 可以避免NN吗?
    - 核心：
        - voxel grid 划分点云：通过floor x'的下标，来定位cell；
        - 以高斯模型的模式来描述一个cell里点是怎么分布的：即，一个点出现在这个地方的概率是多少；
    - 总结：
        - 过程复杂比ICP，但是更快，不用NN；
        - 也是需要好的初始化；

- Feature Based Registration：
    - why：前两种需要有较好的初始解，但很多情况是不存在的
    - 结合前面特征点的提取、描述进行配置

    - RANSAC Registration（配准）：
        - ICP、NDT需要好的初始化，没有怎么办?

        - 特征提取 + 特征描述 + matching + RANSAC（Random Sample Consensus）
            - 之前方法：特征提取、描述可能没有那么精确，在没有初始解的情况下，用来得到初始解，再用ICP、NDT；
            - 匹配点在特征空间来做；
            - RANSAC迭代解出R、t后，计算点变化后的欧式距离，计算inlier的点对
            - 做完之后，可以用ICP、NDT对R、t做一个refine，更加精确了

- 点云配准的pipeline：针对刚性点云，非刚性的则更复杂！
    - 数据预处理：采样、去噪
    - 确定初始pose：R，t
    - 配准算法：ICP、NDT、RANSAC