PCL 点云库

日志

资料

- 3D 视觉工坊

1. 绪论

1. 点云数据及获取

- 定义
  - 三维点的数据集合
- 属性
  - 三维坐标
  - 强度
  - 颜色
  - 时间戳
- 分类
  - 点云组织形式
    - orangized 点云图像 2D
    - unorganized
- 获取方式
  - 激光雷达
  - 深度相机
  - 双目相机
  - 光学相机多视角重建

2. 点云处理基本算法

点云滤波

- 检测和移除点云中噪声或不感兴趣的点
- 分类
  - 基于统计信息
  - 基于领域

    - 基于投影
    - 基于信号处理
    - 基于偏微分方程
- 常用方法
    - 基于体素
    - 移动平均最小二乘

  点云关键点

  特征：
    - 稳定
    - 有区分性
    - 出现次数多
- 常用方法
    - 移动平均最小二乘

点云分割

- 根据空间，集合 特征将点划分为不同的集合
- 常用方法
    - 基于边缘的方法： 变成图像，使用边缘信息
    - 基于区域生长
    - 几何模型拟合： 拟合平面，球形，圆柱

点云匹配

- 估计两帧或者多帧之间的 rigid body transformation 信息，将所有帧的点云配准在同一坐标系
- 分类
    - 初/粗 匹配： 适用于初始位姿差别大的两帧点云
    - 精匹配：优化两帧点云之间的变换
    - 全局匹配： 通常指优化序列点云匹配的误差

      如激光SLAM，两帧之间的匹配，全局匹配
- 常用方法
    - 基于 ICP 的方法
    - 基于特征的匹配方法
    - 深度学习匹配方法

点云目标检测

- 点云中检测某类物体
- 方法
    - 传统机器学习方法
    - 深度学习方法

点云分类/语义分割

- 为每个点云分配一个语义标签
- 方法
    - 传统机器学习方法
    - 深度学习方法

模型重建

- 从点云中获取更精简更紧凑的模型 木哦去 mesh 模型
- 常见的 3D 深度图，点云，体素，网格

3. 常用软件，开源库和数据集

CloudCompare

- 点云处理软件
- 开源，多平台
- 支持常见点云格式，编辑操作
- 支持插件，新功能
- 适合于点云可视化，简单编辑处理

Meshlab

- 处理编辑3D 三角形网格的开源系统
- 主要是编辑，清理，修复，检查，渲染，纹理，转换网络
- 支持多平台

PCL

- 特点
    - 多平台
    - 功能全
    - 开源算法

Open3D

- 特点
    - 支持多平台
    - python集成成熟，可和Pytorch,Tensorflow 集成

2. PCL 基础

1. 介绍和配置

1. 概念

2. 设计理念与基本架构

-

3. 如何安装配置PCL以及使用

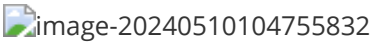
- win下
- linux 下

```
cd pcl
mkdir build
cd build
cmake-gui（可视化的界面）
make -j12sudo make install
```

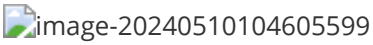
2. PCL库编译

3. PCL中类的介绍

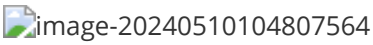
- filters : 实现采样，去除离群点，特征提取，拟合估计
- features : 实现多种三维特征的筛选，曲面法线，曲率，边界点估计
- I/O : 实现数据的输入和输出操作
- surface : 表面重建技术
- registeration: 实现点云配准方法，如ICP
- keypoints : 实现不同的关键点提取方法
- range image : 实现支持不同点云数据集生成的深度图像
- visualization: 可视化 使用VTK 库实现可视化



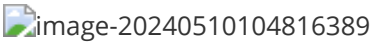
4. PCL 中头文件的介绍



5. 自定义PCL的点类型



6. 在项目中使用PCL库



3. PCL 数据读取及可视化

1. PCL支持点云格式与硬件设备

2. PCL点云存储格式

1. PCD 格式介绍
    - pcd
    - pyl
    - obj
    - xyz(ascii)
    - vtk
    - png
    - tif
2. PCD 文件读取和写入

```
#include <iostream>
#include <pcl/io/pcd_io.h>
#include <pcl/point_types.h>

typedef pcl::PointXYZ PointT; //别名
typedef pcl::PointCloud<PointT> PointCloud;
PointCloud::Ptr cloud(new PointCloud);

//读取
//方法1
pcl::PCDReader reader;
reader.read("xxx.pcd",*cloud);
```

```
//方法2
pcl::io::loadPCDFile("xxx.pcd",*cloud);

//保存
//方法1
pcl::PCDWriter writer;
writer.write("xxx.pcd",*cloud);
writer.writerBinary("xxx.pcd",*cloud);
```

PLY 文件读取和写入

```
#include <pcl/io/ply_io.h>

//读取
pcl::PLYReader reader;

//保存
pcl::PLYWriter writer;
```

1. PCD和PLY和LAS数据格式的转换

3. 常用点云数据集介绍

4. PCL中的数据结构

1. KD-tree
2. Octree

5. 点云可视化

1. Visualization 模块
2. PCLVisualizer 可视化类

6. 点云和其他类型数据的转换

1. 点云和图像的转换
2. 点云和深度图的转换

4. 点云滤波

点云滤波，作为常见的点云处理方法，一般是点云处理的第一步，对后续处理有很重要作用

原始点云数据往往包含大量散列点，孤立点

去噪声，下采样（抽稀），去除地面点

1. 经典滤波介绍

直通滤波器

根据点云的属性在点的属性上设置范围，对点进行滤波，保留范围内的或者保留范围之外的

- 指定一个维度以及该维度以下的值域
- 遍历点云中每个点，判断该点在指定维度上的取值是否在值域内，删除值不在值域内的点
- 遍历结束，留下的点即构成滤波后的点云

```
#include  <pcl/filters/passthrough.h>

//  原点云获取后进行滤波
pcl::PassThrough<pcl::PointXYZ> pass; //创建滤波器对象
pass.setInputCloud (cloud) ; //设置输入点云
pass.setFilterFieldName("z"); //滤波字段名被设置为Z轴方向
pass.setFilterLimits(0.0,1.0);// 可以接受的范围（0，1）
//pass.setFilterLimitsNegative(true); //设置保留范围内，还是过滤掉范围内
pass.filter(*cloud_filtered); //执行滤波，保存过滤结果在cloud_filtered
```

体素滤波器

体素滤波器可以达到下采样的同时，还不会破坏点云本身几何结构的功能，但是会移动点的位置

此外体素滤波器可以去除一定程度的噪音点和离群点，主要功能是用来进行降采样

- 原理： 根据输入的点云，首先计算一个能够刚好包裹煮该点云的立方体，然后根据设定的分辨率，将该大立方体分割成不同的小立方体，对于每一个小立方体内的点，计算他们的质心，并用该质心的坐标来近似该立方体内的若干个点

- ApproximateVoxelGrid 的不同在于这种方法是利用每一个小立方体的中心来近似该立方体的若干点，相对于VoxelGrid 计算速度更快，但是损失了原始点云局部形态的精细度

```
#include <pcl/filter/voxel_grid.h>
//VoxelGrid
pcl::VoxelGrid<pcl::PCLPointCloud2> sor;
sor.setInputCloud(cloud);
sor.setLeafSize(0.01f,0.01f,0.01f);
sor.filter(*cloud_filtered);
// Approximate 体素格滤波器
pcl::ApproximateVoxelGrid<pcl::PointXYZ> approximate_voxel_filter;
approximate_voxel_filter.setLeafSize(0.2,0.2,0.2);
approximate_voxel_filter.setInputCloud(input_cloud);
approximate_voxel_filter.filter(*filtered_cloud);
```

均匀采样滤波

均匀采样滤波基本上等同于体素滤波器，但是其不改变点的位置。下采样后，其点分布基本均匀，但是其点云的准确度要好于体素滤波，因为没有改变移动点的位置。

均匀采样算法： 均匀采样通过构建指定半径的球体对点云进行下采样滤波，将每一个球内距离球体中心最近的点作为下采样之后的点输出。

体素滤波是建立立方体，均匀采样时建立一个球

```
#include <pcl/keypoints/uniform_sampling.h>

pcl::UniformSampling<pcl::PointXYZ> filter;
filter.setInputCloud(cloud);
filter.setRadiusSearch(0.01f);

pcl::PointCloud<int> keypointIndices;
filter.compute(keypointIndices);
```

统计滤波器-去噪

统计滤波器主要用于去除明显离群点

离群点特征在空间中分布稀疏，定义某处点云小于某个密度，即点云无效，计算每个点到其最近的K个点平均距离

则点云中所有点的距离应构成高斯分布，根据给定的均值与方差，可剔除方差之外的点。

```
#include <pcl/filters/statistical_outlier_removal.h>

pcl::StatisticalOutlierRemoval<pcl::PointXYZ> sor;
sor.setInputCloud (cloud);
sor.setMeanK (50); // 设置考虑差项点临近点数
sor.setStddevMulThresh(1.0); //设置判断是否为离群点的阈值
sor.filter (*cloud_filtered);
//然后使用同样的参数调用该滤波器，但是利用函数setNegative 设置使输出取外点，以获取离群点数据（原本滤掉的点）
sor.setNegative (true);
sor.filter(*cloud_filtered);
```

条件滤波器

条件滤波器通过设定滤波条件进行滤波，删除不符合用户指定的一个或者多个条件

直通滤波器时一种比较简单的条件滤波器

```
#include <pcl/filters/conditional_removal.h>
//创建条件定义对象
pcl::ConditionAnd<pcl::PointXYZ>::Ptr range_cond (new
pcl::ConditionAnd<pcl::PointXYZ>());
//添加在Z字段上大于0 的比较算子
range_cond->addComparison (pcl::FieldComparison<pcl::PointXYZ>::ConstPtr(new
pcl::FieldComparison<pcl::PointXYZ>
("z",pcl::ComparisonOps::GT,0,0)));
//添加在Z字段上小于0.8的比较算子
range_cond->addComparison (pcl::FieldComparison<pcl::PointXYZ>::ConstPtr
(new pcl::FieldComparison<pcl::PointXYZ>

("z", pcl::ComparisonOps::LT, 0.8)));

//创建滤波器并用条件定义对象初始化
pcl::ConditionalRemoval<pcl::PointXYZ> condrem;
condrem.setCondition (range_cond);
condrem.setInputCloud(cloud);
condrem.setKeepOrganized(true);// 保持点云的结构
condrem.filter(*cloud_filtered);//执行滤波
```

半径滤波

半径滤波器以某点为中心画一个圆计算落在该点圆中间数量，当数量大于给定值时，则保留该点，数量小于给定值则剔除该点

主要还是用于去除离群点，在一定程度上可以用来筛选边缘点

```
#include <pcl/filters/radius_outlier_removal.h>
pcl::RadiusOutLierRemoval<pcl::PointXYZ> outrem;
outrem.setInputCloud(cloud);
outrem.setRadiusSearch(0.8); //设置半径为0.8 的范围内找临近点
outrem.setMinNeighborsInRadius (2); //设置查询点的临近点集数小于2的删除
outerm.filter(*cloud_filtered); //在半径为0.8 在此半径内必须要有两个邻居点，此点才会保存
```

投影滤波

把点投影到一个参数化模型上，这个参数模型可以时平面，圆球，圆柱，锥形等进行投影滤波，把三维点云投影到二维图像上，然后用图像处理的方法进行处理

```
//填充ModelCoefficients的值，使用ax+by+cz+d=0 平面模型，其中a=b=d=0.c=1. 就是X-Y平面
//定义模型系数对象，并填充对应的数据
pcl::ModelCoefficients::Ptr coefficients(new pcl::ModelCoefficients());
coefficients->values.resize(4);
coefficients->values[0] = coefficients->values[1] =0;
coefficients->values[2]= 1.0;
coefficients->values[3]= 0;

//创建ProjectInliers 对象，使用ModelCoefficients 作为投影对象的模型参数
pcl::ProjectInliers<pcl::PointXYZ> proj; // 创建投影滤波对象
proj.setModelType(pcl::SACMODEL_PLANE); //设置对象对应的投影模型
proj.setInputCloud(cloud); //设置输入点云
proj.setModelCoefficients(coefficients); //设置模型对应的系数
proj.filter(*cloud_projected); //投影结果储存

```

模型滤波

根据点到模型的距离，设置阈值过滤非模型点

基于模型的点分割操作，将模型外的点从点云中剔除

```
//x^2 + y^2 + z^2 = 1
pcl::ModelCoefficients sphere_coeff;
sphere_coeff.values.resize (4);
sphere_coeff.values[0] = 0;
sphere_coeff.values[1] = 0;
sphere_coeff.values[2] = 0;
sphere_coeff.values[3] = 1;
pcl::ModelOutlierRemoval<pcl::PointXYZ> sphere_filter;
sphere_filter.setModelCoefficients (sphere_coeff);
sphere_filter.setThreshold (0.05);
sphere_filter.setModelType (pcl::SACMODEL_SPHERE);
sphere_filter.setInputCloud (cloud);
sphere_filter.filter (*cloud_sphere_filtered);
```

高斯滤波（去噪，平滑）

基于高斯核的卷积滤波实现，高斯滤波器相当于一个具有平衡性能的低通滤波器，通过该类处理后的点云，相对平滑

双边滤波

双边滤波是一种非线性滤波器，它可以达到保持边缘，降噪平滑的效果，一定程度上弥补了高斯滤波的缺点，双边滤波对高斯噪声效果比较好

总结

```
pcl::PassThrought<pcl::PointXYZ> pass;  直通滤波器
pcl::VoxelGrid< pcl::PointXYZ > vox;体素滤波器
pcl::StatisticalOutlierRemoval<pcl::PointXYZ> sor;统计滤波器
pcl::RadiusOutlierRemoval<pcl::PointXYZ> rador;半径滤波器
pcl::UniformSampling<pcl::PointXYz> unisam;均匀采样
pcl::ConditionalRemoval<pcl::PointXYZ> condr;条件滤波器
pcl::ProjectInliers<pcl::PointXYZ> proj;投影滤波器
pcl::ModelOutlierRemoval<pcl::PointXYZ> modr;模型滤波器
pcl::ExtractIndices<pcl::PointXYz> extr;索引提取
空间裁剪滤波
pcl::Clipper3D<pcl::PointXYZ>
pcl::BoxCliper3D<pcl::PointXYZ>
pcl::CropBox<pcl::PointXYZ>
pcl::CropHull<pcl::PointXYZ>
pcl::BilateralFilter<pcl::PointXYZ> bf:双边滤波
pcl::filters::GaussianKernal<PointInT，PointOutT>高斯滤波
```

2. 针对机载激光雷达的滤波方法

1. 渐进三角网滤波
2. 形态学滤波
3. 布料滤波方法

5. 点云关键点，特征描述与提取

1. 介绍

2.

3. 深度学习

4. 例子

6. 点云分割算法

1. 基本概念

点云分割： 根据空间，几何和纹理等特征点进行划分，是同一划分内的点云，拥有相似的特征，点云分割的目的是分块，便于单独处理

点云分类： 为一个点分配一个语义标记，点云的分类是将点云分类到不同的点云集，同一个点云具有相似或者相同的属性，例如 地面，树木，人

也叫点云语义分割。

2. 经典分割算法

- 随机采样一致的方法 （RANSAC）

算法流程：

1. 要得到一个直线模型，需要两个点唯一确定一个直线方程，所以第一步随机选择两个点
2. 通过这两个点，可以计算出这两个点所表示的方程： y=ax+b
3. 找到所有数据点带入这个模型中计算误差
4. 找到所有满足误差阈值的点
5. 然后我们重复1-4这个过程，知道达到一定迭代次数后，选出那个被支持的最多的模型，作为问题的解

- 欧式聚类分割方法

聚类方法，通过特征空间确定点与点之间的亲疏程度

算法流程：

1. 找到空间中某点p，有kdTree 找到离他最近的n个点，判断这n个点到p的距离，将距离小于阈值r的点放在p1,p2,p3 放在类Q中
2. 在Q 里面 找到一个点P1 重复1，找到p22 p23 p24 全部放进Q里
3. 当Q 再也不能有新点加入了，则完成搜索了

- 条件欧式聚类分割

使用类pcl::ConditionEuclidenanClustering 实现点云分割，与其他分割方法不同的是该方法的聚类约束条件（欧式距离，平滑度，RGB颜色）可以由用户自己定义，即当搜索到一个近邻点时，用户可以自定义该领域点是否合并到当前聚类的条件

```
pcl::ConditionalEuclidenanClustering<PointTypeFull> cec(true);//创造条件聚类分割对象，进行初始化
cec.setInputCloud (cloud_with_normals); //设置输入点集
//用于选择不同条件函数
switch(Method) {
    case 1 :
        cec.setConditionFunction(&enforceIntensitySimilarity);
        break;
    case 2:
        cec.setConditionFunction(&enforceCurvatureOrIntensitySimilarity);

}
cec.setClusterTolerance (500.0); //设置聚类参考点的搜索距离
cec.setMinClusterSize(cloud_with_normals->points.size()/1000); //设置过小聚类的标准
cec.setMaxClusterSize(cloud_with_normals->points.size() /5); //设置多大聚类的标准
cec.segment (*clusters);// 获取聚类的结果，分割结果保存在点云索引的向量
cec.getRemovedClusters(small_clusters, large_cluster);//获取无效尺寸的聚类


//如果此函数返回true。则将添加候选点到种子点的聚类中
bool customCondition(const pcl::PointXYZ& seedPoint,const pcl::PointXYZ& candidatePoint, float squaredDistance) {
    //添加自定义条件
    return false;
    return true;

}
```

- 基于区域生长的分割

将具有相似性的点云集合起来构成区域

首先对于每个需要分割的区域找出一个种子点作为生长的起点，然后将种子点周围邻域中与种子有相同或相似性质的点合并到种子像素

所在的区域中，而新的点继续作为种子向四周生长， 直到再没有满足条件的像素可以包括进来，一个区域就生长而成了

算法流程：

1. 计算 法线normal 和曲率curvatures ,依据曲率升序排列
2. 选择曲率最低的为初始种子点，种子周围的临近点和种子点云相比较
3. 法线的方向是否足够相近（法线夹角足够 rpy) 法线夹角阈值
4. 曲率是否足够小（表面处于同一个弯曲程度），区域插值阈值
5. 如果满足2，3 则该点可用作种子点
6. 如果只满足2 ，则归类而不做种子

```
//区域增长聚类分割对象 <点，法线>
pcl::RegionGrowing<pcl::PointXYZ,pcl::Normal> reg;
reg.setMinClusterSize (50); //最小的聚类的点数
reg.setMaxClusterSize(100000); //最大的聚类的点数
reg.setSearchMethod(tree);//搜索方式
reg.setNumberOfNeighbours(30);//设置搜索的邻域点的个数
reg.setInputCloud(cloud); //输入点
reg.setInputNormals(normals);//输入的法线
reg.setSmoothnessThreshold(3.0/180*M_PI);// 设置平滑度 法线差值阈值
reg.setCurvatureThreshold(1.0);// 设置曲率的阈值
```

- 基于颜色的区域生长分割
- 最小图割的分割

图论的最小割广泛应用在网络规划，求解桥问题，图像分割等领域

如果要分开最左边的点，和最右边的点，红绿两种割法都是可行的，但是红线跨过了三条线，绿线只跨过了两条，单从跨线数量上来论

可以得出绿线这种切割方法更优的结论，但假设线上有不同的权值，那么最优切割则和权值有关了

- 基于法线微分的分割
- 基于超体素的分割

与体素滤波器类似，本质是一个个小方块，与之前提到了所有分割不同，对点云实施分割，将场景点云化成很多小块，并研究每个小块之间的关系

以八叉树对点云进行划分，获得不同点团之间的临近关系，与图像相似，点云的临近关系有很多，如面邻接，线邻接，点邻接

3. 基于深度学习的语义分割算法

4. 点云分割实战讲解：杂乱点云数据的分割

‘

7. 点云配准基础

1. 点云拼接

- 两帧点云P和Q，在不同的位置获取，点云坐标是仪器坐标系，点云拼接是把P和Q的点云放到同一个坐标系

主要应用：

- 大场景制图建模
- 不同点云融合
- SLAM

3. 点云拼接流程

点云预处理

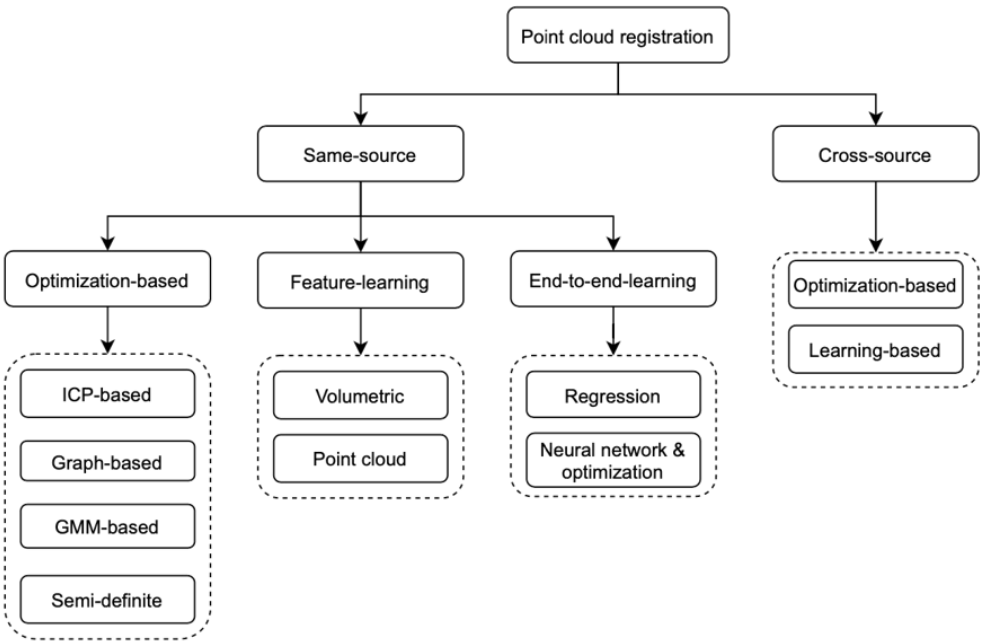
- 去噪声
- 降采样

寻找：

- 提取特征点，线，平面，物体
- 找到对应的点，线，平面 物体

估计坐标转换关系 R T

- 确定优化的目标函数
- 使用非线性优化，估计R，T



8. 点云多帧匹配

1. 多帧点云拼接一般流程

1. 每帧点云预处理
    - 去噪声
    - 降采样
2. 寻找两帧点云对
3. 两帧点云之间寻找 correspondence
    - 提取特征点，线，平面，物体
    - 找到对应的点，线，面，物体
4. 联合估计点云之间的坐标转换关系 R T
    - 确定优化的目标函数
    - 使用非线性优化，估计 R T

2. 难点

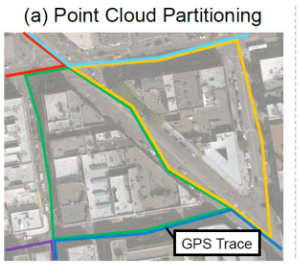
- 稀疏点云
- 点云数据量大
    - 降采样
    - 分区域+ 融合
    - 使用feature
- 最终融合的拼接质量
    - 基于图的优化

3. 基于图优化的大范围配准

- 基于回环的大范围点云配准
    - 基于轨迹的区域划分

• 基于轨迹的区域划分

- 使用GPS/IMU轨迹
- 和街道地图（ street map ）结合，根据道路连接构建图
- 使用隐藏马尔科夫模型（ HMM ）找到轨迹和地图的匹配
- 检测loops



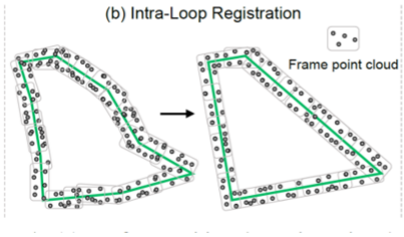
○ 区域内拼接

- 使用Generalized ICP的cost

$$E = \sum_{(\mathbf{p}_i^m, \mathbf{p}_j^n) \in \mathcal{S}} \mathbf{d}\left(\mathbf{p}_i^m, \mathbf{p}_j^n\right)^{\mathsf{T}} \Sigma_{mn}^{-1} \mathbf{d}\left(\mathbf{p}_i^m, \mathbf{p}_j^n\right)$$

- 优化平移
- 优化平移和旋转
  - 小角度假设
  - 只有yaw角度的变化

$$R = \begin{pmatrix} 1 & \theta_z & 0 \\ -\theta_z & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$



○ 区域间拼接

- 基于重叠的 pose，估计区域之间的变换

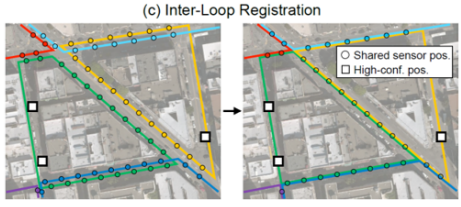
$$A_i \mathbf{s}_i + \mathbf{b}_i = A_j \mathbf{s}_j + \mathbf{b}_j,$$

旋转矩阵　平移矩阵　车位置

- 使用置信度高的GPS点作为锚点

$$A_i \mathbf{s}_i^H + \mathbf{b}_i = \hat{\mathbf{s}}^H,$$

GPS置信度高的点位置　　GPS位置



4. 基于SLAM的大范围配准

- 两帧匹配
    - ICP，NDT
    - 特征匹配
- SegMatch
    - ICP 匹配
    - 使用Segment 进行回环检测
- 基于Segment 的SLAM
    - 两帧匹配
    - 点云分割
    - 提取特征描述子
    - 使用segment定位
    - 更新地图

9. 点云重建

1. 介绍

- 离散点云
    - 数据量大
    - 渲染显示大
    - 模型操作计算不方便
- 网络模型
    - 数据量小
    - 渲染方便
    - 模型操作计算方便

点云模型重建

- 预处理
    - 去噪
    - 滤波
    - 简化
    - 平滑
- 模型重建
- 后处理
    - 填充
    - 生成Mesh

2. 凸包算法

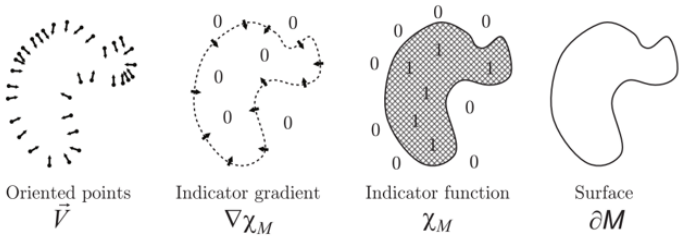
- 凸包
  - 平面凸包： 平面的一个子集S 被称为是" 凸" 的，而且仅当对于两点p q 属于S，线段pq都完全属于S
  - 二维的凸包称为凸多边形，三维的凸包称为凸多面体
- 应用
  - 避免碰撞
  - 计算最小包围盒

3. 曲面重建

Possion重建是一个非常直观的方法。它的核心思想是点云代表了物体表面的位置，其法向量代表了内外的方向。通过隐式地拟合一个由物体派生的指示函数，可以给出一个平滑的物体表面的估计。

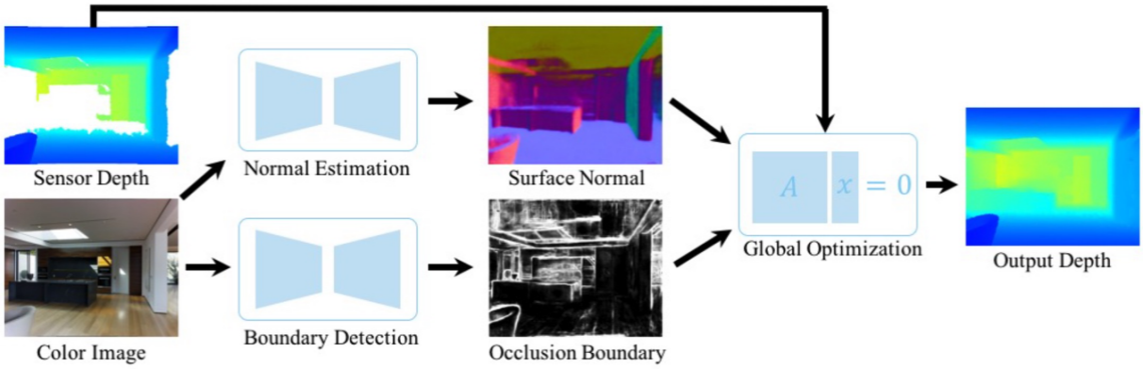
给定一个区域M 及其边界∂M ，指示函数χ_M定义为

$$\chi_M(x) = \begin{cases} 1 & x \in M \\ 0 & x \notin M \end{cases}$$



https://sites.google.com/site/pointcloudreconstruction/marching-cubes

4. 点云补全



Yinda Zhang, Thomas Funkhouser.
Deep Depth Completion of a Single RGB-D Image

10. 点云处理进阶