PCL_filters

PCL中总结了需要进行点云滤波处理的情况

- 点云数据密度不规则需要平滑。
- 因为遮挡等问题造成离群点需要去除。
- 大量数据需要进行下采样。
- 噪声数据需要去除。

滤波器种类

- 直通滤波
- 双边滤波
- 体素格滤波
- 均匀采样
- 增采样
- 统计滤波
- 半径滤波
- 参数化模型投影点云
- ExtractIndices滤波
- ConditionalRemoval
- RadiusOutilerRemoval
- CropHull任意多边形内部点云提取

直通滤波器

- 过滤掉在指定维度方向上取值不在给定值域内的点
- 实现原理
  - 首先，指定一个维度以及该维度下的值域
  - 其次，遍历点云中的每个点，判断该点在指定维度上的取值是否在值域内，删除取值不在值域内的点
  - 最后，遍历结束，留下的点即构成滤波后的点云。
- 直通滤波器简单高效，适用于消除背景，消除高程等操作。

直通滤波器

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

```
void passthrough(pcl::PointCloud<pcl::PointXYZ>::Ptr &input,
                 pcl::PointCloud<pcl::PointXYZ>::Ptr &output)
{
    pcl::PassThrough<pcl::PointXYZ> pass;
    pass.setInputCloud ( cloud: input);              //设置输入点云
    pass.setFilterFieldName ( field_name: "z");          //设置过滤时所需要点云类型的Z字段
    pass.setFilterLimits ( limit_min: 0.0,  limit_max: 1.0);          //设置在过滤字段的范围
    //pass.setFilterLimitsNegative (true);    //设置保留范围内还是过滤掉范围内
    pass.filter ( &: *output);                //执行滤波，保存过滤结果
}
```

双边滤波器

- 核心思想：通过临近采样点的加权平均来修正当前采样点的位置。同时也会有选择的剔除部分与当前采样点差异太大的相邻采样点，从而达到保持原有特征的目的。
- 滤波器核有两个函数生成：空间域核，值域核
  - 空间域核：$w_d(x,y,z,i,j,k) = exp\left(-\frac{(x-i)^2+(y-j)^2+(z-k)^2}{2\sigma_d^2}\right)$
  - 值域核：$w_r(x,y,z,i,j,k) = exp\left(-\frac{\|f(x,y,z)-f(i,j,k)\|^2}{2\sigma_r^2}\right)$
  - 双边滤波器权值模板：$w(x,y,z,i,j,k) = w_d(x,y,z,i,j,k) * w_r(x,y,z,i,j,k)$
  - $g(x,y,z) = \frac{\sum f(x,y,z)w(x,y,z,i,j,k)}{\sum w(x,y,z,i,j,k)}$

双边滤波器

- 双边滤波可以保边去噪。双边滤波卷积计算两个权值，一个用于空间域核（高斯滤波），一个用于值域核，如果卷积在边缘（特征差异较大），值域核权值将会变小降低权重，抑制空间域核的作用。

```
void bilateralFilter(pcl::PointCloud<pcl::PointXYZI>::Ptr &input,
                     pcl::PointCloud<pcl::PointXYZI>::Ptr &output)
{
    pcl::search::KdTree<pcl::PointXYZI>::Ptr tree1( p: new pcl::search::KdTree<pcl:
    // Apply the filter
    pcl::BilateralFilter<pcl::PointXYZI> fbf;
    fbf.setInputCloud( cloud: input);
    fbf.setSearchMethod( tree: tree1);
    fbf.setStdDev( sigma_r: 0.1);
    fbf.setHalfSize( sigma_s: 0.1);
    fbf.filter( &: *output);
}
```

```
void fastBilateralFilter(pcl::PointCloud<pcl::PointXYZI>::Ptr &input,
                         pcl::PointCloud<pcl::PointXYZI>::Ptr &output)
{
    pcl::FastBilateralFilter<pcl::PointXYZI> filter;
    filter.setInputCloud( cloud: input);
    filter.setSigmaS( sigma_s: 0.5);
    filter.setSigmaR( sigma_r: 0.004);
    filter.applyFilter( &: *output);
}
```

体素格滤波器（VoxelGrid）

- 体素化网格实现下采样（减少点的数量），同时保持点云的形状特征。
- 实现原理
  - 使用点云数据创建出相应的三维体素栅格（微小三维立方体的集合）
  - 然后在每个体素内，用体素内的所有点的重心来近似代表其他的点（体素内的所有点用其重心点表示）

体素格滤波器（VoxelGrid）

- #include <pcl/filters/voxel_grid.h>

```
void voxelGrid(pcl::PointCloud<pcl::PointXYZI>::Ptr &input,
               pcl::PointCloud<pcl::PointXYZI>::Ptr& output)
{
    pcl::VoxelGrid<pcl::PointXYZI> sor;  //创建滤波对象
    sor.setInputCloud (cloud: input);            //设置需要过滤的点云给滤波对象
    sor.setLeafSize (lx: 0.1f, ly: 0.1f, lz: 0.1f);  //设置滤波时创建的体素体积为10cm的立方体
    sor.filter (&: *output);           //执行滤波处理，存储输出
}
```

均匀采样（UniformSampling）

- 实现原理：和VoxelGrid滤波很相似，但是VoxelGrid滤波采用的是立方体体素重心，而UniformSamping取半径为r的球体的重心。

```
void uniformSampling(pcl::PointCloud<pcl::PointXYZ>::Ptr &input,
                     pcl::PointCloud<pcl::PointXYZ>::Ptr& output)
{
    pcl::UniformSampling<pcl::PointXYZ> filter;// 均匀采样
    filter.setInputCloud( cloud: input);//输入点云
    filter.setRadiusSearch( radius: 0.01f);//设置半径
    filter.filter( &: *output);
}
```

增采样（UpSampling）

- 增采样是一种表面重建方法，当点云数据少时，增采样可以通过内插目前拥有的点云数据，恢复出原有的表面。

```
void UpSampling(pcl::PointCloud<pcl::PointXYZ>::Ptr &input,
                pcl::PointCloud<pcl::PointXYZ>::Ptr& output)
{
    // 滤波对象
    pcl::MovingLeastSquares<pcl::PointXYZ, pcl::PointXYZ> filter;
    filter.setInputCloud( cloud: input);
    //建立搜索对象
    pcl::search::KdTree<pcl::PointXYZ>::Ptr kdtree;
    filter.setSearchMethod( tree: kdtree);
    //设置搜索邻域的半径为3cm
    filter.setSearchRadius( radius: 0.03);
    // Upsampling 采样的方法有 DISTINCT_CLOUD, RANDOM_UNIFORM_DENSITY
    filter.setUpsamplingMethod(
            method: pcl::MovingLeastSquares<pcl::PointXYZ, pcl::PointXYZ>::SAMPLE_LO
    );
    filter.setUpsamplingRadius( radius: 0.03);  // 采样的半径是
    filter.setUpsamplingStepSize( step_size: 0.02);  // 采样步数的大小
    filter.process( &: *output);
}
```

统计滤波器（StatisticOutilerRemoval）

- 使用统计分析技术来去除离群点。
- 核心思想：假设点云中所有的点与其最近的k个邻居点的平均距离满足高斯分布，那么，根据均值和方差可确定一个距离阈值，当某个点与其最近k个点的平均距离大于这个阈值时，判定该点为离群点并去除

统计滤波器（StatisticOutilerRemoval）

- 实现原理：首先，遍历点云，计算每个点与其最近的k个邻居点之间的平均距离；其次，计算所有平均距离的均值μ与标准差σ，则距离阈值dmax可表示为dmax=μ + α×σ，α是一个常数，可称为比例系数，它取决于邻居点的数目；最后，再次遍历点云，剔除与k个邻居点的平均距离大于dmax的点。

```
// 创建滤波器，对每个点分析的临近点的个数设置为50 ，并将标准差的倍数设置为1  这意味着如果一
 //个点的距离超出了平均距离一个标准差以上，则该点被标记为离群点，并将它移除，存储起来
pcl::StatisticalOutlierRemoval<pcl::PointXYZ> sor;    //创建滤波器对象
sor.setInputCloud (cloud);                            //设置待滤波的点云
sor.setMeanK (50);                                //设置在进行统计时考虑查询点临近点数
sor.setStddevMulThresh (1.0);                       //设置判断是否为离群点的阀值
sor.filter (*cloud_filtered);                     //存储
```

参数化模型投影点云

- 将点云投影到参数化模型上（平面，球）。参数化模型由一组参数来设定。比如平面$ax + by + cy + d = 0$由四个参数所确定。PCL中有特意存储常见模型系数的数据结构。

```
// 填充ModelCoefficients的值,使用ax+by+cz+d=0平面模型，其中 a=b=d=0,c=1 也就是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);                 //投影结果存储cloud_projected
```

ExtractIndices滤波器

- 基于某一分割算法提取点云中的一个子集。

```
pcl::ModelCoefficients::Ptr coefficients ( p: new pcl::ModelCoefficients ());
pcl::PointIndices::Ptr inliers ( p: new pcl::PointIndices ());

pcl::SACSegmentation<pcl::PointXYZ> seg;          //创建分割对象
seg.setOptimizeCoefficients ( optimize: true);    //设置对估计模型参数进行优化处理
seg.setModelType ( model: pcl::SACMODEL_PLANE);   //设置分割模型类别
seg.setMethodType (pcl::SAC_RANSAC);              //设置用哪个随机参数估计方法
seg.setMaxIterations ( max_iterations: 1000);     //设置最大迭代次数
seg.setDistanceThreshold ( threshold: 0.01);      //判断是否为模型内点的距离阀值
seg.setInputCloud ( cloud: input);
seg.segment ( &: *inliers,  &: *coefficients);
// Extract the inliers
// 设置ExtractIndices的实际参数
pcl::ExtractIndices<pcl::PointXYZ> extract;       //创建点云提取对象
extract.setInputCloud ( cloud: input);
extract.setIndices ( indices: inliers);  //
extract.setNegative ( negative: false);
extract.filter ( &: *output);
```

RadiusOutilerRemoval

- RadiusOutilerRemoval核心思想：删除输入点云一定范围内近邻点数量没有达到要求的点。

```
pcl::RadiusOutlierRemoval<pcl::PointXYZ> outrem; //创建滤波器

outrem.setInputCloud(cloud);          //设置输入点云
outrem.setRadiusSearch(0.8);          //设置半径为0.8的范围内找临近点
outrem.setMinNeighborsInRadius(2); //设置查询点的邻域点集数小于2的删除
// apply filter
outrem.filter(*cloud_filtered); //在半径为0.8 在此半径内必须要有两个邻居点，
```

ConditionalRemoval

- 用于删除点云中不符合用户指定的一个或多个条件的点。

```
//为条件定义对象添加比较算子
pcl::ConditionAnd<pcl::PointXYZ>::Ptr range_cond(
        p: new pcl::ConditionAnd<pcl::PointXYZ>()
); //创建条件定义对象
//添加在Z字段上大于0的比较算子
range_cond->addComparison(
        comparison: pcl::FieldComparison<pcl::PointXYZ>::ConstPtr(
                p: new pcl::FieldComparison<pcl::PointXYZ>( field_name: "z",  op: pc
//添加在Z字段上小于0.8的比较算子
range_cond->addComparison(
        comparison: pcl::FieldComparison<pcl::PointXYZ>::ConstPtr(
                p: new pcl::FieldComparison<pcl::PointXYZ>( field_name: "z",  op: pc
pcl::ConditionalRemoval<pcl::PointXYZ> condrem;  // 创建滤波器并用条件定义对象初始化
condrem.setCondition( condition: range_cond);
condrem.setInputCloud( cloud: input);    //输入点云
condrem.setKeepOrganized( val: true); //设置保持点云的结构
condrem.filter( &: *output);  //大于0.0小于0.8这两个条件用于建立滤波器
```

CropHull任意多边形内部点云提取

- 输入一个2D封闭的多边形和一个2D平面点云（这些平面点是多边形的顶点），然后提取属于该2D封闭的多边形内部或外部的点。

```
// 输入2D多边形
pcl::PointCloud<pcl::PointXYZ>::Ptr boundingbox_ptr(new pcl::PointCloud<pcl::PointXYZ>);
boundingbox_ptr->push_back(pcl::PointXYZ(0.1, 0.1, 0));
boundingbox_ptr->push_back(pcl::PointXYZ(0.1, -0.1, 0));
boundingbox_ptr->push_back(pcl::PointXYZ(-0.1, 0.1, 0));
boundingbox_ptr->push_back(pcl::PointXYZ(-0.1, -0.1, 0));


pcl::ConvexHull<pcl::PointXYZ> hull;  // 创建凸包对象
hull.setInputCloud(boundingbox_ptr);  // 设置输入点云
hull.setDimension(2);  // 设置凸包维度

std::vector<pcl::Vectices> polygons;  // 设置顶点类型向量，保存凸包的顶点
pcl::PointCloud<pcl::PointXYZ>:Ptr surface_hull(new pcl::PointCloud<pcl::PointXYZ>);
hull.reconstruct(*surface_hull, polygons);  // 计算2D凸包结果

pcl::PointCloud<pcl::PointXYZ>::Ptr objects(new pcl::PointCloud<pcl::PointXYZ>);
pcl::CropHull<pcl::PointXYZ> bb_filters;  // 创建CropHull对象
bb_filters.setDim(2);  // 设置维度
bb_filters.setInputCloud(cloud);  // 设置需要滤波的点云
bb_filters.setHullIndices(polygons);  // 设置封闭多边形顶点
bb_filters.setHullCloud(surface_hull);  // 设置封闭多边形形状
bb_filters.filter(*objects);  // 执行滤波
```