

PCL


PCL简介

PCL(Point Cloud Library)是开源的点云算法库，点云就是在三维空间中的点，其信息通常包括最基本的空间坐标（直角坐标系or球坐标系），强度信息，颜色信息。



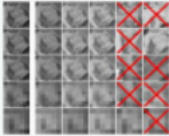

PCL安装

依赖

The following code libraries are **required** for the compilation and usage of the PCL libraries shown below:




 **Note**

pcl_* denotes all PCL libraries, meaning that the particular dependency is a strict requirement for the usage of anything in PCL.

Logo	Library	Minimum version	Mandatory
	Boost	1.65	pcl_*
	Eigen	3.3	pcl_*
	FLANN	1.9.1	pcl_*
	VTK	6.2	pcl_visualization

Optional

The following code libraries enable certain additional features for the PCL libraries shown below, and are thus **optional**:

Logo	Library	Minimum version	Mandatory
	Qhull	2011.1	pcl_surface
	OpenNI	1.3	pcl_io
	CUDA	9.2	pcl_*

1. 在[官网](#)下载你需要的版本的源码
2. 在安装包目录下

```
mkdir build && cd build
cmake -DCMAKE_INSTALL_PREFIX=/usr/local/pcl-1.10.0/pcl .. (这个目录自己定)
# 或者直接cmake ..
make -j8
sudo make install
```

3. 等待完成即可
4. 当需要指定用具体版本的PCL时，在Cmake中：

```
set(PCL_DIR "/usr/local/pcl-1.10.0/pcl/share/pcl-1.10")
find_package(PCL 1.10 REQUIRED)
```

如果只有单一版本的PCL

```
find_package(PCL REQUIRED)
```

[官方文档](#)

PCL基础类型

空间中有很多点，这些点组成的集合或其子集就是点云

```
pcl::PointCloud
//存储点的集合
//PointCloud类中包括几个重要的成员变量
//header: 包含点云的信息，如时间等。
//points: 保存点云的容器，类型为 std::vector<PointT>。
//width: 类型为uint32_t，表示点云宽度，即一行点云的数量。
```

//height: 类型为uint32_t, 表示点云高度。若为有序点云, height 可以大于 1, 即多行点云, 每行固定点云的宽度; 若为无序点云, height 等于 1, 即一行点云, 此时 width 的数量即为点云的数量。
//is_dense: bool 类型, 若点云中的数据都是有限的 (不包含 inf/NaN 这样的值), 则为 true, 否则为 false。

```
pcl::PointT  
//eg: pcl::PointXYZ、pcl::PointXYZI、pcl::PointXYZRGB  
  
pcl::PointCloud::Ptr  
//指向PointCloud的指针
```

输入输出

PCD(Point Cloud Data)是一种文件格式, 包含有头和数据, 头存储了该点云文件的一些信息。

从PCD文件中读取点云数据

第一种方法

```
#include <iostream>  
#include <pcl/io/pcd_io.h>  
#include <pcl/point_types.h>  
  
int main ()  
{  
    pcl::PointCloud<pcl::PointXYZ>::Ptr cloud (new  
    pcl::PointCloud<pcl::PointXYZ>); //创建PointCloud指针  
  
    if (pcl::io::loadPCDFile<pcl::PointXYZ> ("test_pcd.pcd", *cloud) == -1) //读取  
    数据  
    {  
        PCL_ERROR ("Couldn't read file test_pcd.pcd \n");  
        return (-1);  
    }  
    std::cout << "Loaded "  
        << cloud->width * cloud->height  
        << " data points from test_pcd.pcd with the following fields: "  
        << std::endl;  
    for (const auto& point: *cloud){  
        std::cout << "    " << point.x  
            << " "    << point.y  
            << " "    << point.z << std::endl;  
    }  
  
    return (0);  
}
```

第二种方法

```
pcl::PCDReader pcd_reader;  
pcd_reader.read("test_pcd.pcd", *cloud);
```

写入点云数据到PCD文件中

第一种方法

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

int main ()
{
    //构造了一个点云，可以观察到一些基本的点云信息是必须的，见PCL基础类型
    //这里是PointCloud类型，不是指针
    pcl::PointCloud<pcl::PointXYZ> cloud;
    cloud.width    = 5;
    cloud.height   = 1;
    cloud.is_dense = false;
    cloud.resize (cloud.width * cloud.height);
    for (auto& point: cloud)
    {
        point.x = 1024 * rand () / (RAND_MAX + 1.0f);
        point.y = 1024 * rand () / (RAND_MAX + 1.0f);
        point.z = 1024 * rand () / (RAND_MAX + 1.0f);
    }

    //写入文件中
    pcl::io::savePCDFileASCII ("test_pcd.pcd", cloud);
    std::cerr << "Saved " << cloud.size () << " data points to test_pcd.pcd." <<
    std::endl;

    return (0);
}
```

第二种方法

```
pcl::PCDWriter pcd_writer;
pcd_writer.write<pcl::PointXYZ>("test_pcd.pcd", cloud, false); //false保存为ascii文件
```

可视化

简单的方法: CloudViewer

```
//...cloud exist
pcl::visualization::CloudViewer viewer("xxx");
viewer.showCloud(cloud);
```

终端中查看PCD文件

```
pcl_viewer xxx.pcd

# 需要提前安装sudo apt-get install pcl-tools
```

滤波

直通滤波 (PassThrough filter)

含义：去掉三维坐标的某个维度坐标在某个范围内或者外的所有点

```
pcl::PassThrough<pcl::PointXYZ> pass;//创建直通滤波的对象
pass.setInputCloud(cloud);//输入的点云，是指针
pass.setFilterFieldName("z");//设置直通滤波要在xyz哪个坐标上使用
pass.setFilterLimits(0.0, 1.0);//设置在某个坐标方向上的范围
pass.setNegative(true);//如果设置为true，则在上面的这个范围内的所有点都被去掉，如果设置为false则仅保留范围内的点；默认为false
pass.filter(*cloud_filtered);//滤波后的结果
```

体素滤波 (VoxelGrid filter)

含义：降采样，减少点的数量。通过在三维点云空间中画小盒子的方式，将每个小盒子中的点全都近似为盒子质心的点，将盒子里其他的点删掉，可以不破坏点云的整体形状但减少点的数量。

```
pcl::VoxelGrid<pcl::PointXYZ> sor;//创建体素滤波对象
sor.setInputCloud(cloud);//输入点云指针
sor.setLeafSize(0.01f, 0.01f, 0.01f);//设置体素体积为1立方厘米的立方体
sor.filter(*cloud_filtered);//结果
```

离群点滤波 / 统计滤波 (StatisticalOutlierRemoval)

含义：去除噪点。在雷达采样的时候会出现采样出错的情况，会影响我们对点云数据进行处理。统计滤波就是对点云图中的每一个点考虑其与其最邻近的k个点的距离并求平均距离。在所有点的距离的集合应构成高斯分布的假设下，可以剔除一些在统计意义上离均值过远的点。

```
pcl::StatisticalOutlierRemoval<pcl::PointXYZ> sor;//创建统计滤波对象
sor.setInputCloud(cloud);//输入
sor.setMeanK(50);//k个邻近点
sor.setStddevMulThresh(1.0);//标准差倍数，判断是否为离群点的阈值
sor.filter(*filtered_cloud);//结果
```

投影参数模型 (很少用到)

含义：将点云投影到某一个模型上，比如平面、球体

```
//创建一个平面模型,  $Ax+By+Cz+D = 0$ , 其中  $A = B = 0, C = 1, D = 0$ 
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;

pcl::ProjectInliers<pcl::PointXYZ> proj;//创建类对象
proj.setModelType(pcl::SACMODEL_PLANE);//设置投影模型
proj.setInputCloud(cloud);//输入
proj.setModelCoefficients(coefficients);//投影模型参数设置
proj.filter(*cloud_projected);//输出
```

PCL内置了多种采样模型

```
namespace pcl
{
    enum SacModel
    {
        SACMODEL_PLANE,
        SACMODEL_LINE,
        SACMODEL_CIRCLE2D,
        SACMODEL_CIRCLE3D,
        SACMODEL_SPHERE,
        SACMODEL_CYLINDER,
        SACMODEL_CONE,
        SACMODEL_TORUS,
        SACMODEL_PARALLEL_LINE,
        SACMODEL_PERPENDICULAR_PLANE,
        SACMODEL_PARALLEL_LINES,
        SACMODEL_NORMAL_PLANE,
        SACMODEL_NORMAL_SPHERE,
        SACMODEL_REGISTRATION,
        SACMODEL_REGISTRATION_2D,
        SACMODEL_PARALLEL_PLANE,
        SACMODEL_NORMAL_PARALLEL_PLANE,
        SACMODEL_STICK
    };
}
```

分割 (Extracting indices from a PointCloud)

含义: 依照某种模型从点云图中提取出最符合这种模型的点云图的子集

```
// 平面分割, 去除
pcl::PointCloud<pcl::PointXYZ>::Ptr cloudplane(
    new pcl::PointCloud<pcl::PointXYZ>);
pcl::ModelCoefficients::Ptr coefficients(new pcl::ModelCoefficients);//存模型参数的
pcl::PointIndices::Ptr inliers(new pcl::PointIndices);//标记点云哪些是模型上的点
pcl::SACSegmentation<pcl::PointXYZ> seg;//分割的对象

seg.setOptimizeCoefficients(true);//是否使用优化
seg.setModelType(pcl::SACMODEL_PLANE);//使用平面模型
seg.setMethodType(pcl::SAC_RANSAC);//使用RANSAC算法
```

```

seg.setDistanceThreshold(0.01); //允许在这个模型上的点距离模型的距离 (m)
seg.setInputCloud(cloudpass); //输入
seg.segment(*inliers, *coefficients); //分割得到标记和模型参数

pcl::ExtractIndices<pcl::PointXYZ> extract;
extract.setInputCloud(cloudpass);
extract.setIndices(inliers);
extract.setNegative(true);
extract.filter(*cloudplane); //分割结果

```

半径滤波

含义：去掉那些在其一定半径范围内没有一定数量邻近点的点

```

pcl::PointCloud<pcl::PointXYZ>::Ptr cloud_filter(new
pcl::PointCloud<pcl::PointXYZ>);
pcl::RadiusOutlierRemoval<pcl::PointXYZ> sor;
sor.setInputCloud(cloud);
sor.setRadiusSearch(0.02);
sor.setMinNeighborsInRadius(15);
sor.setNegative(false);
sor.filter(*cloud_filter);

```