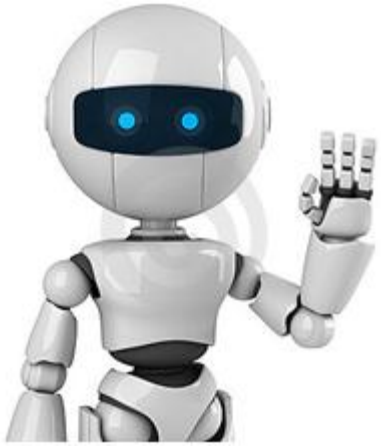
6. 点云



东北大学 张云洲

内 容 提 纲

6.1 理解点云库

6.2 编写 PCL 程序

6.3 分割

6.1 理解点云库

■点云

- 点云是一种能够直观地**表示和操作 3D 传感器所提供数据**的方式
- 传感器（相机/激光雷达）在 3D 坐标参考系下**对空间进行有限点采样构成点云**

■点云库 PCL（point cloud library）

- PCL是一个独立的、大规模开源 **3D 点云处理软件包**
- 提供大量**数据类型和数据结构**，方便表示采样空间的点及其属性
- 提供大量**算法**对数据样本进行处理，比如滤波\表面重建等

■PCL 与 ROS 的 PCL 接口

- PCL：为处理 3D 数据提供一组数据结构和算法
- ROS 的 PCL 接口：提供一组消息及消息与 PCL 数据结构之间的转换函数

■点云中的公共字段

- header：pcl::PCLHeader类型，指定了点云的获取时间
- points：std::vector<PointT,…>类型，它是存储所有点的容器
- width：指定了点云组成一种图像时的宽度，否则它包含的是云中点的数量
- height：指定了点云组成一种图像时的高度，否则它总是1
- is_dense：指定了点云中是否有无效值（无穷大或NaN值）
- sensor_origin_：Eigen::Vector4f类型，并且定义了传感器根据相对于原点的平移所得到的位姿
- sensor_orientation_：Eigen::Quaternionf类型，并且定义了传感器旋转所得到的位姿PCL算法利用这些字段来处理数据，并且用户可以利用它们来创建自己的算法。

6.1.1 不同的点云类型

■PCL 定义了多种不同类型的点，常见的类型：

- **pcl::PointXYZ**：最简单也最常见的点类型，只存储 3D xyz 的信息
- **pcl::PointXYZI**：在上面类型的基础上还包括了一个描述点亮度的字段
- **pcl::PointXYZRGBA**：存储 3D xyz 信息、RGB 颜色信息和透明度
- **pcl::PointXYZRGB**：与上一个类型相似，没有透明度信息
- **pcl::Normal**：最常用的点类型，表示曲面上给定点出的法线以及测量的曲率
- **pcl::PointNormal**：包含上一类型的给定点出的法线、曲率还包括 3D xyz 信息

6.1.2 PCL 中的算法

PCL 将每个算法左乘一个类，这个类属于一个有着特定共性的继承类，允许开发者通过获取以及存在的算法，加上新算法所需要的参数，以重用这些算法，并且只需要提供所需要的参数值，其余参数取默认值

```
pcl::PointCloud<pcl::PointXYZ>::Ptr cloud(new pcl::PointCloud<pcl::PointXYZ>);
pcl::PointCloud<pcl::PointXYZ>::Ptr result(new pcl::PointCloud<pcl::PointXYZ>);
pcl::Algorithm<pcl::PointXYZ> algorithm;
algorithm.setInputCloud(cloud);
algorithm.setParameter(1.0);
algorithm.setAnotherParameter(0.33);
algorithm.process (*result);
```

6.1.3 ROS 的 PCL 接口

通过 ROS 自带的基于消息的通信系统，ROS 的 PCL 接口提供了与 PCL 数据结构进行通信所需要的方法，这里定义了**不同的消息类型去处理点云和其他 PCL 算法中产生的数据**，也提供了一组将本地 **PCL 数据类型转换成消息的函数。**

- **std_msgs::Header**：这不是真的消息类型，但它通常是每一个ROS消息的一部分。
- **sensor_msgs::PoinCloud2**：可能是最重要的消息类型，用来转换 pcl::PointCloud 类型
- **pcl_msgs::PointIndices**：储存一个点云中点的索引，等价的 PCL 类型是pcl::PointIndices
- **pcl_msgs::PolygonMesh**：保存描绘网格、顶点和多边形的信息，等价的PCL 类型是 pcl::PolygonMesh。
- **pcl_msgs::Vetices**：将一组顶点的索引保存在一 个数组中， 例如描述一个多边形，等价的 PCL 类型是pcl::Vertices。
- **pcl_msgs::ModelCoefficients**：储存一个模型的不同系数， 例如描述一个平面需要的四个参数，等价的 PCL 类型是 pcl::ModelCoefficients.

通过 ROS 的 PCL 功能包提供的转换函数可以将前面的消息转换成 PCL 类型或者从 PCL 类型转换成消息。

在 pcl_conversions 命名空间里面提供的函数

```
void fromPCL(const <PCL Type> &, <ROS Message type> &);
void moveFromPCL(<PCL Type> &, <ROS Message type> &);
void toPCL(const <ROS Message type> &, <PCL Type> &);
void moveToPCL(<ROS Message type> &, <PCL Type> &);
```

标准版本：深拷贝

Move版本：浅拷贝

深拷贝 Vs 浅拷贝：

浅拷贝：只是对指针的拷贝，拷贝后两个指针指向同一个内存空间。

深拷贝：不但对指针进行拷贝，而且对指针指向的内容进行拷贝，经深拷贝后的指针是指向两个不同地址的指针。

https://www.cnblogs.com/echolun/p/7889848.html

6.2 编写 PCL 程序

准备工作：PCL 库测试

■在工作区中创建 ROS 软件包

```
$ catkin_create_pkg chapter6_tutorials pcl_conversions pcl_ros pcl_msgs sensor_msgs
```

■在软件包中创建一个源文件目录

```
$ rospack profile
$ roscd chapter6_tutorials
$ mkdir src
```

■pcl_sample.cpp： 创建一个 ROS 节点并发布一个带有 100 个像素的点云

■添加 PCL 库到 CMakeList.txt，链接到系统的 PCL 库，并产生可执行文件

```
find_package(PCL REQUIRED)
include_directories(include ${PCL_INCLUDE_DIRS})
link_directories(${PCL_LIBRARY_DIRS})
add_executable(pcl_sample src/pcl_sample.cpp)
target_link_libraries(pcl_sample ${catkin_LIBRARIES} ${PCL_LIBRARIES})
```

6.2.1 创建点云 创建仅由随机的点组成的 PCL 点云

```
#include <ros/ros.h>
#include <pcl/point_cloud.h>
#include <pcl_conversions/pcl_conversions.h>
#include <sensor_msgs/PointCloud2.h>
```

标准的ROS 头文件

PCL 头文件

消息头文件

```
main (int argc, char **argv)
{
ros::init (argc, argv, "pcl_create");
ros::NodeHandle nh;
```

初始化节点

```
ros::Publisher pcl_pub = nh.advertise<sensor_msgs::PointCloud2>
("pcl_output", 1);
pcl::PointCloud<pcl::PointXYZ> cloud;
sensor_msgs::PointCloud2 output;
```

创建 ROS 发布者并进行广播

```
cloud.width = 100;
cloud.height = 1;
cloud.points.resize(cloud.width * cloud.height);
for (size_t i = 0; i < cloud.points.size (); ++i){
cloud.points[i].x = 1024 * rand () / (RAND_MAX + 1.0f);
cloud.points[i].y = 1024 * rand () / (RAND_MAX + 1.0f);
cloud.points[i].z = 1024 * rand () / (RAND_MAX + 1.0f);
}
```

分配空间，填充点云

将 PCL 点云类型转化为ROS 消息类型

```
//Convert the cloud to ROS message
pcl::toROSMsg(cloud, output);
output.header.frame_id = "odom";
ros::Rate loop_rate(1);
while (ros::ok())
{
pcl_pub.publish(output);
ros::spinOnce();
loop_rate.sleep();
}
return 0;
}
```

以 1 hz 的频率发布消息

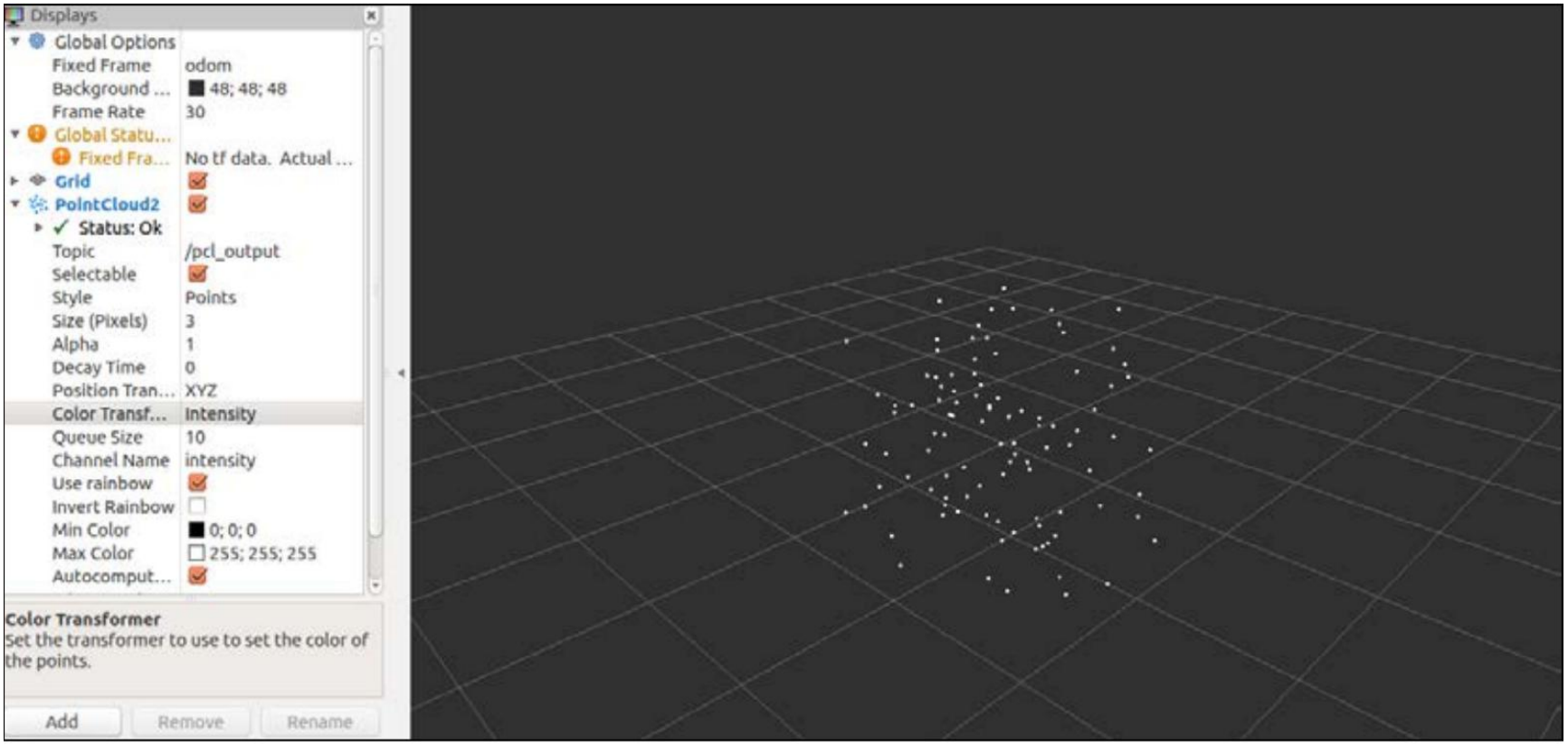
消息的 frame_id 设置为 odom，是为了在 rviz 中可视化 PointCloud2消息

6.2.1 创建点云 创建仅由随机的点组成的 PCL 点云

运行

终端1： roscore
终端2： rosrun chapter6_tutorials pcl_create
终端3： rosrun rviz rviz

通过左下角的 Add 添加 pcl_output 主题来增加一个PointCloud2的对象



6.2.2 加载和保存点云

■PCL 提供了标准的**PCD 格式**去**加载和存储点云**到硬盘

- 这种格式开始是由一个包含**关于点云中点类型和元素数目信息的数据头**
- 然后是**符合指定类型点的列表**

```
# .PCD v.5 - Point Cloud Data file format
FIELDS x y z intensity distance sid
SIZE 4 4 4 4 4 4
TYPE F F F F F F
COUNT 1 1 1 1 1 1
WIDTH 460400
HEIGHT 1
POINTS 460400
DATA ascii
```

扩展阅读：

https://blog.csdn.net/weixin_35695879/article/details/84992057

■**如何加载 PCD 文件并将点云结果发布为 ROS 消息?**

6.2.2 加载和保存点云

pcl_read.cpp

```
#include <ros/ros.h>
#include <pcl/point_cloud.h>
#include <pcl_conversions/pcl_conversions.h>
#include <sensor_msgs/PointCloud2.h>
#include <pcl/io/pcd_io.h>      包含加载点云到 PCD 的相关定义的头文件
main(int argc, char **argv){
ros::init (argc, argv, "pcl_read");
ros::NodeHandle nh;
ros::Publisher pcl_pub = nh.advertise<sensor_msgs::PointCloud2>
("pcl_output", 1);
sensor_msgs::PointCloud2 output;
pcl::PointCloud<pcl::PointXYZ> cloud;
pcl::io::loadPCDFile ("test_pcd.pcd", cloud);      从硬盘中加载 PCD 文件
pcl::toROSMsg(cloud, output);
output.header.frame_id = "odom";
ros::Rate loop_rate(1);
while (ros::ok())
 {
  pcl_pub.publish(output);
  ros::spinOnce();
  loop_rate.sleep();
 }
  return 0;
}
```

其余程序与上一节相同
不同点在于上一节随机生成点云
此程序读取 PCD 文件的点云

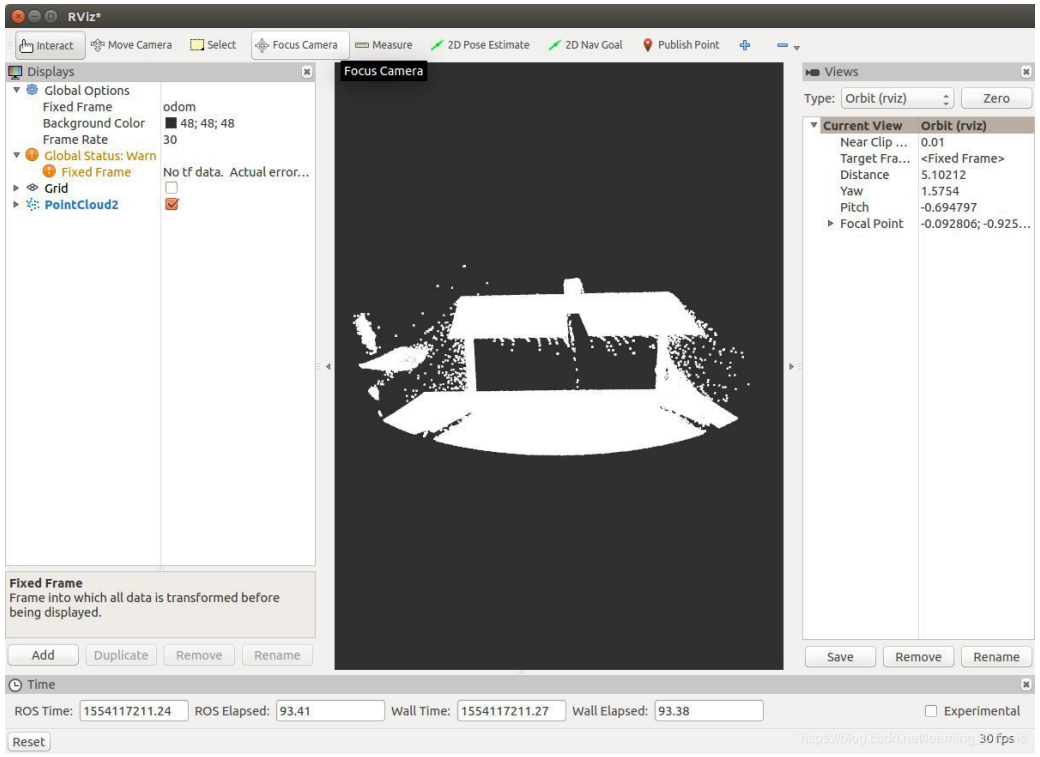
6.2.2 加载和保存点云

运行 终端1：roscore

终端2：roscd chapter6_tutorials/data

rosrun chapter6_tutorials pcl_read

终端3：rosrun rviz rviz



6.2.2 加载和保存点云

将接收的点云存储到 PCD 文件中

pcl_write.cpp

```
#include <ros/ros.h>
#include <pcl/point_cloud.h>
#include <pcl_conversions/pcl_conversions.h>
#include <sensor_msgs/PointCloud2.h>
#include <pcl/io/pcd_io.h>
void cloudCB(const sensor_msgs::PointCloud2 &input)
{
  pcl::PointCloud<pcl::PointXYZ> cloud;
  pcl::fromROSMsg(input, cloud);
  pcl::io::savePCDFileASCII ("write_pcd_test.pcd", cloud);
}
main (int argc, char **argv)
{
  ros::init (argc, argv, "pcl_write");
  ros::NodeHandle nh;
  ros::Subscriber bat_sub = nh.subscribe("pcl_output", 10, cloudCB);
  ros::spin();
  return 0;
}
```

保存为 write_pcd_test.pcd 文件

订阅的主题

6.2.2 加载和保存点云到硬盘

为运行这个示例，必须有一个发布者通过 pcl_output 主题来提供点云，使用前面的 pcl_read 来满足这个要求

终端1：roscore
终端2：roscd chapter6_tutorials/data && rosrun chapter6_tutorials pcl_read
终端3：roscd chapter6_tutorials/data && rosrun chapter6_tutorials pcl_write

6.2.3 可视化点云

PCL 提供了几种方式来可视化点云，最简单的方式是基本的点云查看器，可以在 3D 查看器中展示任何类型的 PCL 点云，同时还提供了一组回调函数供用户交互使用

pcl_visualize.cpp

```
#include <iostream>
#include <ros/ros.h>
#include <pcl/visualization/cloud_viewer.h>
#include <sensor_msgs/PointCloud2.h>
#include <pcl_conversions/pcl_conversions.h>
```

```
class cloudHandler
{
public:
    cloudHandler()
    : viewer("Cloud Viewer")
    {
        pcl_sub = nh.subscribe("pcl_output", 10,
&cloudHandler::cloudCB, this);
        viewer_timer = nh.createTimer(ros::Duration(0.1),
&cloudHandler::timerCB, this);
    }
```

所有函数封装在一个类中

设置接收 pcl_output 主题，每100ms 回调一下

```
    void cloudCB(const sensor_msgs::PointCloud2 &input)
    {
        pcl::PointCloud<pcl::PointXYZ> cloud;
        pcl::fromROSMsg(input, cloud);

        viewer.showCloud(cloud.makeShared());
    }
```

回调函数传递给查看器，查看器自动更新

```
    void timerCB(const ros::TimerEvent&)
    {
        if (viewer.wasStopped())
        {
            ros::shutdown();
        }
    }
```

如果查看器关闭则终止节点

```
protected:
    ros::NodeHandle nh;
    ros::Subscriber pcl_sub;
    pcl::visualization::CloudViewer viewer;
    ros::Timer viewer_timer;
};

main (int argc, char **argv)
{
    ros::init (argc, argv, "pcl_visualize");

    cloudHandler handler;

    ros::spin();

    return 0;
}
```

6.2.3 可视化点云

运行：

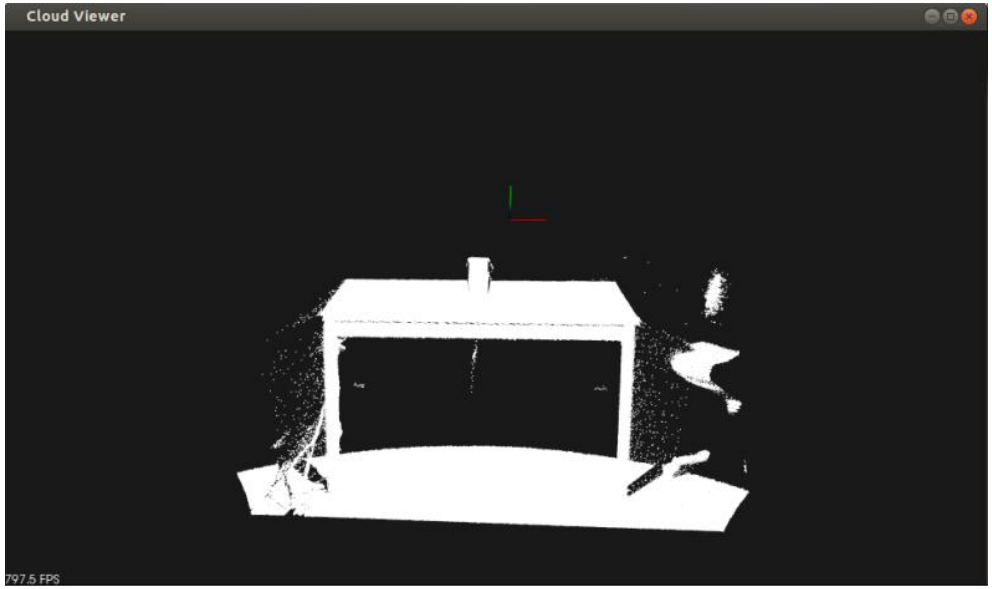
终端1：roscore

终端2：运行 pcl_read 示例和数据源

roscd chapter6_tutorials/data

rosrun chapter6_tutorials pcl_read

终端3：rosrun chapter6_tutorials pcl_visualize



6.2.4 滤波和缩减采样

- 当尝试处理点云时，可能会遇到两个主要问题：
  - **噪声过多**：导致算法错误地解释数据，产生不正确、不准确的结果
  - **密度太大**：使算法需要花费很多时间去完成
- 采用滤波的方式减少噪声或离群值 → **pcl_filtered.cc**
- 采用降采样的方式减少密度而不损失有价值的信息 → **pcl_downsampling.cc**

pcl_filtered.cc

```
#include <ros/ros.h>
#include <pcl/point_cloud.h>
#include <pcl_conversions/pcl_conversions.h>
#include <sensor_msgs/PointCloud2.h>
#include <pcl/filters/statistical_outlier_removal.h>
class cloudHandler
{
public:
    cloudHandler()
    {
        pcl_sub = nh.subscribe("pcl_output", 10,
&cloudHandler::cloudCB, this);
        pcl_pub =
nh.advertise<sensor_msgs::PointCloud2>("pcl_filtered", 1);
    }
    void cloudCB(const sensor_msgs::PointCloud2& input)
    {
        pcl::PointCloud<pcl::PointXYZ> cloud;
        pcl::PointCloud<pcl::PointXYZ> cloud_filtered;
        sensor_msgs::PointCloud2 output;
        pcl::fromROSMsg(input, cloud);
        pcl::StatisticalOutlierRemoval<pcl::PointXYZ>
statFilter;
        statFilter.setInputCloud(cloud.makeShared());
        statFilter.setMeanK(10);
        statFilter.setStddevMulThresh(0.2);
        statFilter.filter(cloud_filtered);
        pcl::toROSMsg(cloud_filtered, output);
        pcl_pub.publish(output);
    }
protected:
    ros::NodeHandle nh;
    ros::Subscriber pcl_sub;
    ros::Publisher pcl_pub;
};
main(int argc, char** argv)
{
    ros::init(argc, argv, "pcl_filter");
    cloudHandler handler;
    ros::spin();
    return 0;
}
```

所有函数封装在一个类中

回调函数定义两个点云，分别是滤波前后

输入点云转换为消息

回调函数中的滤波部分

滤波后的点云发布

运行滤波例程：

终端1：roscore

终端2：运行 pcl_read 示例和一个数据源

roscd chapter6_tutorials/data

rosrun chapter6_tutorials pcl_read

终端3：rosrun chapter6_tutorials pcl_filter

pcl_downsampling.cc

```
#include <ros/ros.h>
#include <pcl/point_cloud.h>
#include <pcl_conversions/pcl_conversions.h>
#include <sensor_msgs/PointCloud2.h>
#include <pcl/filters/voxel_grid.h>
class cloudHandler
{
public:
    cloudHandler()
    {
        pcl_sub = nh.subscribe("pcl_filtered", 10,
&cloudHandler::cloudCB, this);
        pcl_pub =
nh.advertise<sensor_msgs::PointCloud2>("pcl_downsa
mpled", 1);
    }
    void cloudCB(const sensor_msgs::PointCloud2
&input)
    {
        pcl::PointCloud<pcl::PointXYZ> cloud;
        pcl::PointCloud<pcl::PointXYZ>
cloud_downsampled;
        sensor_msgs::PointCloud2 output;
        pcl::fromROSMsg(input, cloud);
        pcl::VoxelGrid<pcl::PointXYZ> voxelSampler;
        voxelSampler.setInputCloud(cloud.makeShared());
        voxelSampler.setLeafSize(0.5f, 0.5f, 0.5f);
        voxelSampler.filter(cloud_downsampled);
        pcl::toROSMsg(cloud_downsampled, output);
        pcl_pub.publish(output);
    }
protected:
    ros::NodeHandle nh;
    ros::Subscriber pcl_sub;
    ros::Publisher pcl_pub;
};

main(int argc, char **argv)
{
    ros::init(argc, argv, "pcl_downsampling");

    cloudHandler handler;
    ros::spin();

    return 0;
}
```

将点云分解成体素，用子云的中心代替每个体素的所有点

运行降采样例程： 终端1：roscore

终端2：运行 pcl_read 示例和一个数据源

roscd chapter6_tutorials/data

rosrun chapter6_tutorials pcl_read

终端3：rosrun chapter6_tutorials pcl_downsampling

PCL的点云滤波：

PCL中的点云处理模块提供了很多灵活实用的滤波处理算法，例如双边滤波、高斯滤波、条件滤波、直通滤波、基于随机采样一致性滤波等。

需要进行点云滤波处理的情况：

1. 点云数据密度不规则需要平滑；
2. 由于遮挡等问题造成离群点需要去除；
3. 大量数据需要进行下采样；
4. 噪音数据需要去除。

对应的方法主要如下：

1. 按具体给定的规则限制过滤去除点。
2. 通过常用滤波算法修改点的部分属性。
3. 对数据进行下采样。

资料链接：https://blog.csdn.net/Yong_Qi2015/article/details/102487593

PCL的5种滤波器：

1. 直通滤波器：对于在空间分布有一定空间特征的点云数据，比如使用线结构光扫描的方式采集点云，沿z向分布较广，但x,y向的分布处于有限范围内。此时可使用直通滤波器，确定点云在x或y方向上的范围，可较快剪除离群点，达到第一步粗处理的目的。

2. 体素滤波器：体素的概念类似于像素，使用AABB包围盒将点云数据体素化，一般体素越密集的地方信息越多，噪音点及离群点可通过体素网格去除。另一方面如果使用高分辨率相机等设备对点云进行采集，往往点云会较为密集。过多的点云数量会对后续分割工作带来困难。体素滤波器可以达到向下采样同时不破坏点云本身几何结构的功能。

3. 统计滤波器：考虑到离群点的特征，则可以定义某处点云小于某个密度，既点云无效。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。给定均值与方差，可剔除3∑之外的点。

4. 条件滤波：通过设定滤波条件进行滤波，类似于分段函数，当点云在一定范围则留下，不在则舍弃。

5. 半径滤波器：与统计滤波器相比更加简单粗暴。以某点为中心画一个圆计算落在该圆中点的数量，当数量大于给定值时，则保留该点，否则剔除该点。此算法运行速度快，依序迭代留下的点一定是最密集的，但是圆的半径和圆内点的数目都需要人工指定。

资料链接：https://blog.csdn.net/qq_39482438/article/details/81110036

滤波器如何使用？

1、如果是线结构光的采集方式得到的点云，则沿z向的分布较广，但沿x、y方向的分布则处于有限的范围内。此时，可采用**直通滤波**，确定x或者y方向的范围，快速裁剪离群点。

2、如果使用高分辨率相机等设备对点云进行采集，则点云往往较为密集。过多的点云数据对后续的分割工作带来困难。**体素法滤波**可以达到下采样的同时不破坏点云本身几何结构的功能。

3、**统计滤波器**用于去除明显的离群点（离群点往往由噪声引入）。噪声信息属于无用信息，信息量较小。所以离群点表达的信息可以忽略不计。考虑到离群点的特征，则可以定义某处点云小于某个密度，既点云无效。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。给定均值与方差，可剔除3∑之外的点。

4、**半径滤波器**与统计滤波器相比更加简单粗暴。以某点为中心画一个圆计算落在该圆中点的数量，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。此算法运行速度快，依序迭代留下的点一定是最密集的，但是圆的半径和圆内点的数目都需要人工指定。

可组合使用完成任务，效果更佳。

资料链接：https://blog.csdn.net/Yong_Qi2015/article/details/102487593

6.2.5 配准与匹配

■配准与匹配可以使用在两个数据集中寻找共同的结构或特征，然后利用它们将数据集拼接在一起

■PCL 提供了一种迭代最近点（ICP）算法执行配准与匹配，可以用于运动估计，点云拼接之后可以减少位姿估计的不确定性

pcl_matching.cpp

```
// 头文件略
class cloudHandler
{
public:
    cloudHandler()
    {
        pcl_sub = nh.subscribe("pcl_downsampled", 10,
&cloudHandler::cloudCB, this);
        pcl_pub =
nh.advertise<sensor_msgs::PointCloud2>("pcl_matched", 1);
    }
    void cloudCB(const sensor_msgs::PointCloud2 &input)
    {
        pcl::PointCloud<pcl::PointXYZ> cloud_in;
        pcl::PointCloud<pcl::PointXYZ> cloud_out;
        pcl::PointCloud<pcl::PointXYZ> cloud_aligned;
        sensor_msgs::PointCloud2 output;
        pcl::fromROSMsg(input, cloud_in);
        cloud_out = cloud_in;
        for (size_t i = 0; i < cloud_in.points.size (); ++i)
        {
            cloud_out.points[i].x = cloud_in.points[i].x + 0.7f;
        }
        pcl::IterativeClosestPoint<pcl::PointXYZ,
pcl::PointXYZ> icp;
        icp.setInputSource(cloud_in.makeShared());
        icp.setInputTarget(cloud_out.makeShared());
        icp.setMaxCorrespondenceDistance(5);
        icp.setMaximumIterations(100);
        icp.setTransformationEpsilon (1e-12);
        icp.setEuclideanFitnessEpsilon(0.1);
        icp.align(cloud_aligned);
        pcl::toROSMsg(cloud_aligned, output);
        pcl_pub.publish(output);
    }

protected:
    ros::NodeHandle nh;
    ros::Subscriber pcl_sub;
    ros::Publisher pcl_pub;
};
// main()函数略
```

使用降采样的主题作为输入

发布

三个点云：输入点云、结果点云，与输入点云对齐的固定点云

假设使用输入原始点云作为要对其的固定点云

输入点云沿 x 轴产生了一点偏移

调用 ICP 算法进行对齐

匹配例程： 终端1：roscore

终端2：运行 pcl_read 示例和一个数据源

roscd chapter6_tutorials/data

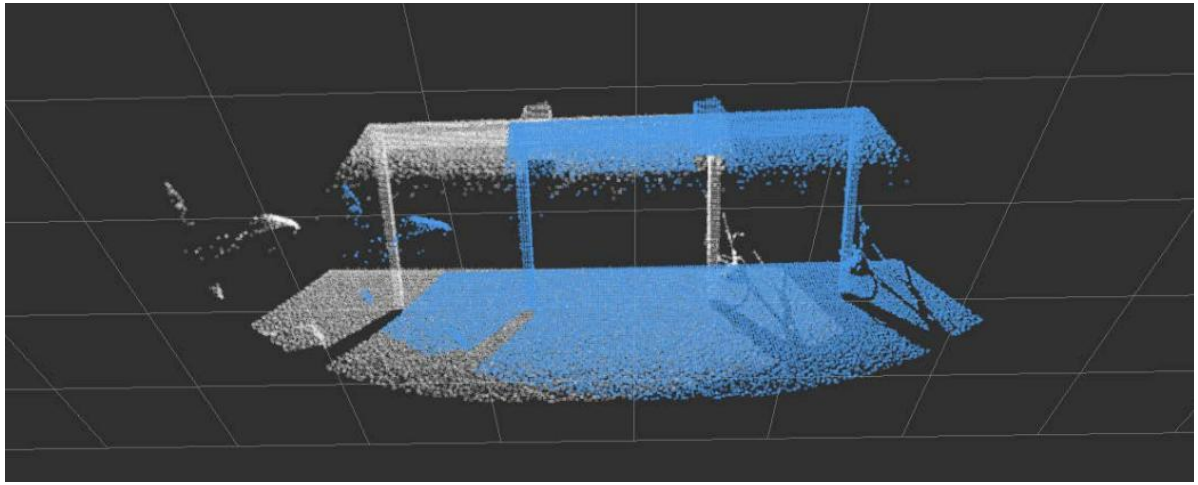
rosrun chapter6_tutorials pcl_read

终端3：运行滤波例程

rosrun chapter6_tutorials pcl_filter

终端4：运行配准和匹配节点，需要订阅前面节点产生的 pcl_Downsampled

rosrun chapter6_tutorials pcl_matching



6.2.6 点云分区

■在处理点云数据时也需要访问一个局部区域的点云或者操作特定点的相邻区域，但由于点云在**一维数据结构**中存储数据，造成这种操作的复杂性

■为解决该问题，PCL 提供了两种空间数据结构，**称为 k-d 树和八叉树**（octree），可以作为另一种替代选择并且能够更加结构化地展示任何点云

■八叉树：在一个树结构中每个节点有八个子节点，并且可以用来分割 3D 空间；

■K-d 树：是一颗二叉树，其中节点表示 k 维的点

■**pcl_partitioning.cpp**：使用八叉树搜索和检索一个指定点周围的所有点

```
// 头文件略
class cloudHandler
{
public:
    cloudHandler()
    {
        pcl_sub = nh.subscribe("pcl_downsampled", 10,
&cloudHandler::cloudCB, this);
        pcl_pub =
nh.advertise<sensor_msgs::PointCloud2>("pcl_partitioned", 1);
    }
    void cloudCB(const sensor_msgs::PointCloud2 &input)
    {
        pcl::PointCloud<pcl::PointXYZ> cloud;
        pcl::PointCloud<pcl::PointXYZ> cloud_partitioned;
        sensor_msgs::PointCloud2 output;
        pcl::fromROSMsg(input, cloud);
        float resolution = 128.0f;
      pcl::octree::OctreePointCloudSearch<pcl::PointXYZ>
octree (resolution);
        octree.setInputCloud (cloud.makeShared());
        octree.addPointsFromInputCloud ();
        pcl::PointXYZ center_point;
        center_point.x = 0 ;
        center_point.y = 0.4;
        center_point.z = -1.4;
 float radius = 0.5;
        std::vector<int> radiusIdx;
        std::vector<float> radiusSQDist;
        if (octree.radiusSearch (center_point,
radius, radiusIdx, radiusSQDist) > 0)
        {
            for (size_t i = 0; i < radiusIdx.size (); ++i)
            {
cloud_partitioned.points.push_back(cloud.poin
ts[radiusIdx[i]]);
            }
        }
    pcl::toROSMsg(cloud_partitioned, output);
    output.header.frame_id = "odom";
    pcl_pub.publish(output);
  }

protected:
    ros::NodeHandle nh;
    ros::Subscriber pcl_sub;
    ros::Publisher pcl_pub;
};
// main()函数略pcl_partitioning
```

pcl_partitioning.cpp

使用降采样的主题作为输入

发布

创建八叉树搜索算法，传入分辨率值

定义分区的中心

搜索函数，返回落在半径内点的索引，以及这些点到中心点距离的平方

得到只包含这些点的新点云

将分区点云转化为消息发布

运行点云分区例程： 终端1：roscore

终端2：运行 pcl_read 示例和一个数据源

roscd chapter6_tutorials/data

rosrun chapter6_tutorials pcl_read

终端3：运行滤波例程

rosrun chapter6_tutorials pcl_filter

终端4：运行降采样例程

rosrun chapter6_tutorials pcl_downsampling

终端5：运行分区例程

rosrun chapter6_tutorials pcl_partitioning

6.3 分割

■分割是**将数据组分割为**满足特定标准的**不同数据块**的过程

■有时候也许会涉及从一个点云中基于统计特性提取结构化信息，也有可能仅仅需要提取指定颜色范围内的点

■数据可能满足特定的数学模型，如一个平面、一条直线或一个球体，可以使用模型估计算法计算与数据相匹配的模型参数，有了这些参数就有可能提取出属于这个模型的点并评估他们的匹配度

■例程展示**如何执行一个基于模型的点云分割**，也会采用 **RANSAC** 迭代算法即使存在离群值也能进行准确的估计

```
#include <ros/ros.h>
#include <pcl/point_cloud.h>
#include <pcl_conversions/pcl_conversions.h>
#include <pcl/ModelCoefficients.h>
#include <pcl/sample_consensus/method_types.h>
#include <pcl/sample_consensus/model_types.h>
#include <pcl/segmentation/sac_segmentation.h>
#include <pcl/filters/extract_indices.h>
#include <sensor_msgs/PointCloud2.h>
class cloudHandler
{
public:
    cloudHandler()
    {
        pcl_sub = nh.subscribe("pcl_downsampled", 10, &cloudHandler::cloudCB, this);
        pcl_pub = nh.advertise<sensor_msgs::PointCloud2>("pcl_segmented", 1);
        ind_pub = nh.advertise<pcl_msgs::PointIndices>("point_indices", 1);
        coef_pub = nh.advertise<pcl_msgs::ModelCoefficients>("planar_coef", 1);
    }
 void cloudCB(const sensor_msgs::PointCloud2 &input)
    {
        pcl::PointCloud<pcl::PointXYZ> cloud;
        pcl::PointCloud<pcl::PointXYZ> cloud_segmented;

        pcl::fromROSMsg(input, cloud);

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

存储点云中点的索引

存储一个数学模型的系数

```
// Create the segmentation object
pcl::SACSegmentation<pcl::PointXYZ> segmentation;
segmentation.setModelType(pcl::SACMODEL_PLANE);
segmentation.setMethodType(pcl::SAC_RANSAC);
segmentation.setMaxIterations(1000);
segmentation.setDistanceThreshold(0.01);
segmentation.setInputCloud(cloud.makeShared());
segmentation.segment(*inliers, coefficients);
```

期望匹配的数学模型

用到的算法

迭代最大次数

模型的最大距离

```
// Publish the model coefficients
pcl_msgs::ModelCoefficients ros_coefficients;
pcl_conversions::fromPCL(coefficients, ros_coefficients);
coef_pub.publish(ros_coefficients);
// Publish the Point Indices
pcl_msgs::PointIndices ros_inliers;
pcl_conversions::fromPCL(*inliers, ros_inliers);
ind_pub.publish(ros_inliers);
```

转换并发布内点和模型系数

```
// Create the filtering object
pcl::ExtractIndices<pcl::PointXYZ> extract;
extract.setInputCloud(cloud.makeShared());
extract.setIndices(inliers);
extract.setNegative(false);
extract.filter(cloud_segmented);
//Publish the new cloud
sensor_msgs::PointCloud2 output;
pcl::toROSMsg(cloud_segmented, output);
pcl_pub.publish(output);}
```

创建分割点云

运行：（保持）

```
rosrun chapter6_tutorials pcl_planar_segmentation
```

问题

1.点云数据转换为激光雷达？

http://wiki.ros.org/pointcloud_to_laserscan

2. 激光雷达的数据转化为点云？

http://wiki.ros.org/laser_geometry

参考资料

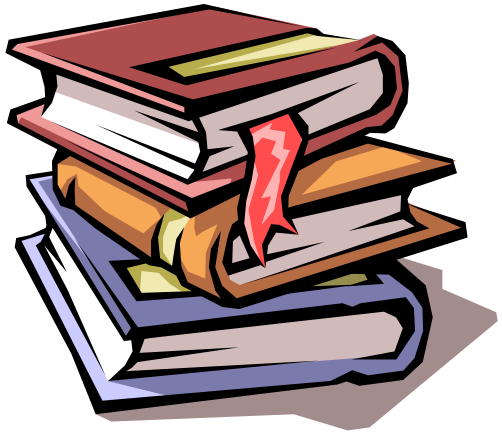
PCL官网：http://www.pointclouds.org/

Using PCL in ROS：http://wiki.ros.org/pcl

可参考书籍

《ROS机器人程序设计》Aaron Martinez Enrique Femandez著刘品杰译

《ROS机器人开发实践》胡春旭 著



Q & A
Thanks !