重庆邮电大学

自动化学院学生实验报告

课程名称：智能车辆定位与导航技术

实验内容：基于NDT或ICP构建点云地图

专业班级：08902102

姓　　名：胡凯新

学　　号：2021212996

成　　绩：

指 导 教 师　岑汝平

学年学期：2023 - 2024 学年 ■春季 □秋季 学期

重庆邮电大学自动化学院制

1、查看 bag 数据包 lidar_imu_rtk_one_lap.bag 雷达点云数据对应的 topic 名称

```
hkx@hkx-virtual-machine:~/ICP_3D$ rosbag info lidar_imu_rtk_one_lap.bag
path:        lidar_imu_rtk_one_lap.bag
version:     2.0
duration:    2:44s (164s)
start:       May 31 2023 17:49:36.05 (1685526576.05)
end:         May 31 2023 17:52:20.36 (1685526740.36)
size:        2.8 GB
messages:    35502
compression: none [1644/1644 chunks]
types:       custom_msgs/NaviData      [7bf1fc8c7c8a4b0662c1e912fa5e0615]
             sensor_msgs/Imu           [6a62c6daae103f4ff57a132d6f95cec2]
             sensor_msgs/MagneticField [2f3b0b43eed0c9501de0fa3ff89a45aa]
             sensor_msgs/PointCloud2   [1158d486dd51d683ce2f1be655c3c181]
topics:      /imu/data               16431 msgs    : sensor_msgs/Imu
             /imu/mag                16430 msgs    : sensor_msgs/MagneticField
             /lslidar_point_cloud     1643 msgs    : sensor_msgs/PointCloud2
             /navi_msg                 499 msgs    : custom_msgs/NaviData
             /raw_imu                  499 msgs    : sensor_msgs/Imu
```

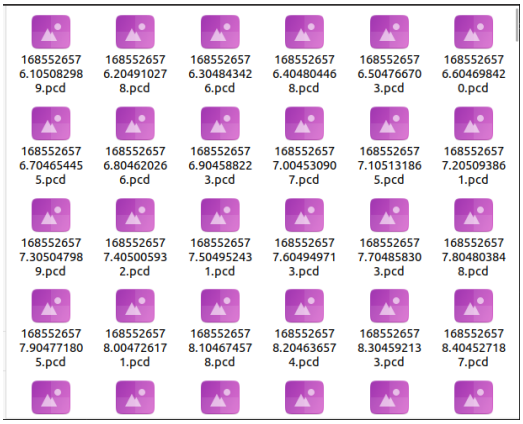
2、读取激光雷达数据：rosrun pcl_ros bag_to_pcd <xxxxxx.bag> <topic> <output_directory>
其中 xxxxxx.bag 指读取的 bag 数据包的名字

topic 指 bag 数据包雷达点云数据对应的 topic 名称

output_directory 指输出过程中所创建的目录名称，得到的 pcd 格式的雷达点云数据放在此目录下。

```
hkx@hkx-virtual-machine:~/ICP_3D$ rosrun pcl_ros bag_to_pcd lidar_imu_rtk_one_lap.bag /lslidar_point_cloud test.pacd
```

得到按时间戳排序的 1643 个雷达点云数据如下：



3、创建工作空间 pairwise_incremental_registration，再在工作空间下创建文件夹 src 用于存放源代码，并在 src 路径下，创建一个 C++代码文件。

其主要思路是使用迭代最近点算法，逐步将一系列点云进行两两匹配，到所有的点云进行变换后，使得都与第一个点云在统一坐标系中。在每个连贯的有重叠的点云之间找到最佳的变换，并累积这些变换到全部的点云。具体代码思路如下：

（1）声明一个结构体，定义 cloud 成员变量指向 PCL 中用于存储 3D 点云数据的类，方便对点云以文件名和点云对象进行成对处理管理，在配准过程中，可以同时接受多个点云文件输入，程序从第一个文件开始，连续的两两配准处理，然后存储配准后的点云文件。

```
27  /* 处理点云的方便的结构定义 */
28  struct PCD
29  {
30      pcl::PointCloud<pcl::PointXYZ>::Ptr cloud;
31      std::string f_name;
32      PCD() : cloud (new pcl::PointCloud<pcl::PointXYZ>) {};
33  };
```

（2）定义一个新的点的表示：定义一个新的类继承自 PCL 库中类，定义基类中成员变量即表示点的维度数为 4，方法中将 PointNormal 类型的点数据复制到浮点数数组中。

```
/* 以< x, y, z, curvature >形式定义一个新的点 */
class MyPointRepresentation : public pcl::PointRepresentation <PointNormalT>
{
    using pcl::PointRepresentation<PointNormalT>::nr_dimensions_;
public:
    MyPointRepresentation ()
    {
        nr_dimensions_ = 4;   //定义尺寸值
    }
    virtual void copyToFloatArray (const PointNormalT &p, float * out) const //覆盖
    {
        out[0] = p.x;
        out[1] = p.y;
        out[2] = p.z;
        out[3] = p.curvature;
    }
};
```

（3）主函数中先检查用户输入，再在点云矢量 data 中加载了用户输入的所有点云数据，建立两个窗口显示点云，左边显示未匹配的源和目标点云，右边显示匹配后的源和目标点云。

再循环为相邻点云找到变换矩阵后，将目标点云变换到源点云坐标系下，并将源点云与变换后的目标点云存储到一点云文件，同时用此次找到的变换矩阵更新全局变换，用于将后续的点云都变换到与第一个输入点云同一坐标系下。

```
int main (int argc, char** argv)
{

    std::vector<PCD, Eigen::aligned_allocator<PCD> > data;
    loadData (argc, argv, data);//加载数据
    if (data.empty ())//检查用户输入，若为空输出错误消息并返回1，不为空则输出已加载的数据集数量
    {
        PCL_ERROR ("Syntax is: %s <source.pcd> <target.pcd> [*]", argv[0]);
        PCL_ERROR ("[*] - multiple files can be added. The registration results of (i, i+1) will be registered against (i+2), etc");
        PCL_INFO ("Example: %s `rospack find pcl`/test/bun0.pcd `rospack find pcl`/test/bun4.pcd", argv[0]);
        return (-1);
    }
    PCL_INFO ("Loaded %d datasets.", (int)data.size ());

    p = new pcl::visualization::PCLVisualizer (argc, argv, "Pairwise Incremental Registration example");//创建一个PCL可视化对象
    p->createViewPort (0.0, 0, 0.5, 1.0, vp_1); //创建两个视图窗口
    p->createViewPort (0.5, 0, 1.0, 1.0, vp_2);
    PointCloud::Ptr result (new PointCloud), source, target;//result用于存储匹配后的点云
    Eigen::Matrix4f GlobalTransform = Eigen::Matrix4f::Identity (), pairTransform;//前为存储全局变换矩阵，后为存储两两匹配时变换矩阵
    for (size_t i = 1; i < data.size (); ++i)//对data中点云进行两两匹配
    {
        source = data[i-1].cloud;//获取源点云与目标点云
        target = data[i].cloud;
        showCloudsLeft(source, target);//左侧窗口显示点云
        PointCloud::Ptr temp (new PointCloud);
        PCL_INFO ("Aligning %s (%d) with %s (%d).\n", data[i-1].f_name.c_str (), source->points.size (), data[i].f_name.c_str (), target->points.size ());
        pairAlign (source, target, temp, pairTransform, true);//匹配并返回匹配后的点云及变换矩阵
        pcl::transformPointCloud (*temp, *result, GlobalTransform); //把当前的两两配对转换到全局变换
        GlobalTransform = pairTransform * GlobalTransform;          //更新全局变换
        std::stringstream ss;
        ss << i << ".pcd";
        pcl::io::savePCDFile (ss.str (), *result, true);//保存配准对，转换到第一个点云框架中
    }
}
```

（4）其中加载数据函数，先检查文件长度是否小于扩展名长度，是则跳过；再看其是否为 pcd 文件，是则加载点云并移除 NAN 点后保存在总体点云列表中。

```
void loadData (int argc, char **argv, std::vector<PCD, Eigen::aligned_allocator<PCD> > &models)
{
    std::string extension (".pcd");
    for (int i = 1; i < argc; i++)
    {
        std::string fname = std::string (argv[i]);
        if (fname.size () <= extension.size ())//检查长度是否小于等于扩展名长度，是则跳过
        {
            continue;
        }
        std::transform (fname.begin (), fname.end (), fname.begin (), (int(*)(int))tolower);//将文件名转为小写
        if (fname.compare (fname.size () - extension.size (), extension.size (), extension) == 0)//检查参数是否为pcd文件
        {
            PCD m;
            m.f_name = argv[i];//将pcd对象m文件名设为当前命令行参数
            pcl::io::loadPCDFile (argv[i], *m.cloud);//加载pcd文件
            std::vector<int> indices;
            pcl::removeNaNFromPointCloud(*m.cloud,*m.cloud, indices); //从点云中移除NAN点
            models.push_back (m);//将清理过的pcd对象m添加回models中
        }
    }
}
```

（5）匹配函数，其中参数有输入一组需要配准的点云，以及是否进行下采样的设置项，其他参数输出配准后的点云以及变换矩阵。

先判断是否要下采样，再计算点云法线，创建 ICP 对象，设置其参数，以需要匹配的两个点云作为输入，根据数据集调整其收敛判断条件和最大距离。

```
131  void pairAlign (const PointCloud::Ptr cloud_src, const PointCloud::Ptr cloud_tgt, PointCloud::Ptr output, Eigen::Matr
165      pcl::IterativeClosestPointNonLinear<PointNormalT, PointNormalT> reg;//创建ICP对象
166      reg.setTransformationEpsilon (1e-6);//设收敛阈值
167      reg.setMaxCorrespondenceDistance (0.1); //设两对应间最大距离为10厘米，需根据数据集大小调整
168      reg.setPointRepresentation (boost::make_shared<const MyPointRepresentation> (point_representation));  //设置点表示
169      reg.setInputCloud (points_with_normals_src);
170      reg.setInputTarget (points_with_normals_tgt);
```

再进入迭代，记录并累积由 ICP 返回的变换，若迭代 N 次找到的变换和迭代 N-1 次中找到的变换之间的差异小于传给 ICP 的变换收敛阈值，选择源与目标之间更靠近的对应点距离阈值来改善配准过程，且在匹配结果窗口对迭代的最新结果进行刷新显示。

```
175      for (int i = 0; i < 30; ++i)//迭代
176      {
177          PCL_INFO ("Iteration Nr. %d.\n", i);
178          points_with_normals_src = reg_result;//更新源点云为上次匹配结果
179          reg.setInputCloud (points_with_normals_src);
180          reg.align (*reg_result);
181          Ti = reg.getFinalTransformation () * Ti;//在每一个迭代之间累积转换
182          //如果这次转换和之前转换之间的差异小于阈值，则通过减小最大对应距离来改善程序
183          if (fabs ((reg.getLastIncrementalTransformation () - prev).sum ()) < reg.getTransformationEpsilon ())
184          {
185              reg.setMaxCorrespondenceDistance (reg.getMaxCorrespondenceDistance () - 0.001);
186              prev = reg.getLastIncrementalTransformation ();
187          }
188          showCloudsRight(points_with_normals_tgt, points_with_normals_src);  //可视化当前状态
189      }
```

一旦找到最优的变换，ICP 返回的变换是从源点云到目标点云的变换矩阵，我们求逆变换得到目标点云到源点云的变化矩阵，并应用到目标点云，变换后的目标点云然后添加到源点云中，并且将源点云和变换矩阵一起返回到主函数。

```
190      targetToSource = Ti.inverse();  //得到目标点云到源点云的变换
191      //把目标点云转换回源框架
192      pcl::transformPointCloud (*cloud_tgt, *output, targetToSource);
193      p->removePointCloud ("source");
194      p->removePointCloud ("target");
195      PointCloudColorHandlerCustom<PointT> cloud_tgt_h (output, 0, 255, 0);
196      PointCloudColorHandlerCustom<PointT> cloud_src_h (cloud_src, 255, 0, 0);
197      p->addPointCloud (output, cloud_tgt_h, "target", vp_2);//添加到右可视器中
198      p->addPointCloud (cloud_src, cloud_src_h, "source", vp_2);
199      PCL_INFO ("Press q to continue the registration.\n");
200      p->spin ();
201      p->removePointCloud ("source");
202      p->removePointCloud ("target");
203      /* 添加源点云到转换目标 */
204      *output += *cloud_src;
205      final_transform = targetToSource;
```

4、再在工作空间根目录下编写 Cmakelists.txt 文件如下：

```
cmake_minimum_required(VERSION 2.6 FATAL_ERROR)

project(pairwise_incremental_registration)

find_package(PCL 1.4 REQUIRED)
include_directories(${PCL_INCLUDE_DIRS})
link_directories(${PCL_LIBRARY_DIRS})
add_definitions(${PCL_DEFINITIONS})

add_executable (${PROJECT_NAME}_node src/pairwise_incremental_registration.cpp)
target_link_libraries (${PROJECT_NAME}_node ${PCL_LIBRARIES})
```

其中先是指定 cmake 的最低版本要求，若低于则会停止且报错；再是定义项目名称；查找 PCL 库且指定最低版本；接着给出 cmake 在编译时包含 PCL 的头文件目录，链接是搜索 PCL 的库文件目录以及预处理器定义；并生成可执行文件。

5、在工作空间根目录下创建一个 build 文件夹，用于存放编译过程中产生的文件，然后执行编译，即执行以下四条命令：

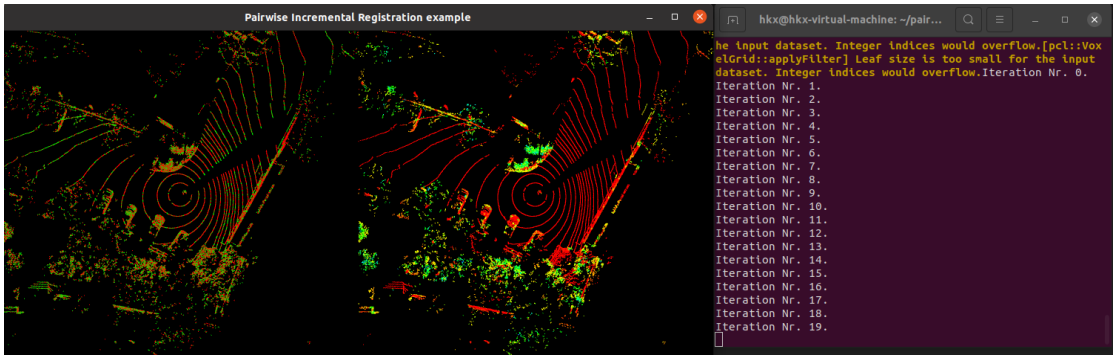
```
mkdir build
cd build
cmake ..
make
```

6、执行完后将在 build 文件夹下生成可执行文件 pairwise_incremental_registration_node，再执行需要匹配的 pcd 文件即可，这里选取读取后的数据中前 6 个进行匹配。

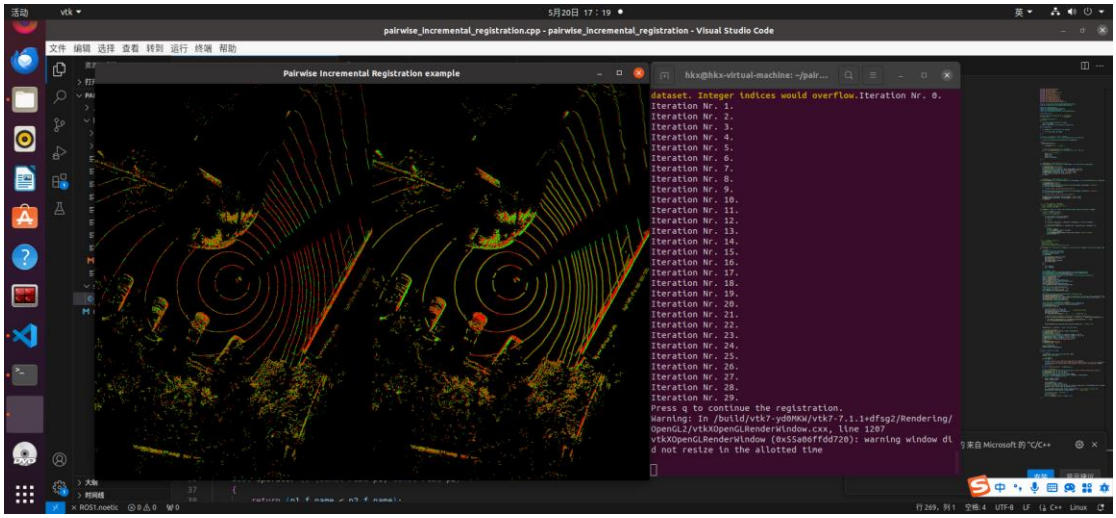
```
hkx@hkx-virtual-machine:~/pairwise_incremental_registration/build$ ./pairwise_increme
ntal_registration_node a.pcd b.pcd c.pcd d.pcd e.pcd f.pcd
```

其中左边为未进行匹配前源点云和目标点云，其中红色为源点云，绿色为目标点云。

命令行提示需要继续执行配准则按键 q，按 q 后，先进行 a.pcd 与 b.pcd 进行匹配，会在窗口右边看到不断调整的点云，其实是配准过程中的迭代中间结果输出，如下图所示。

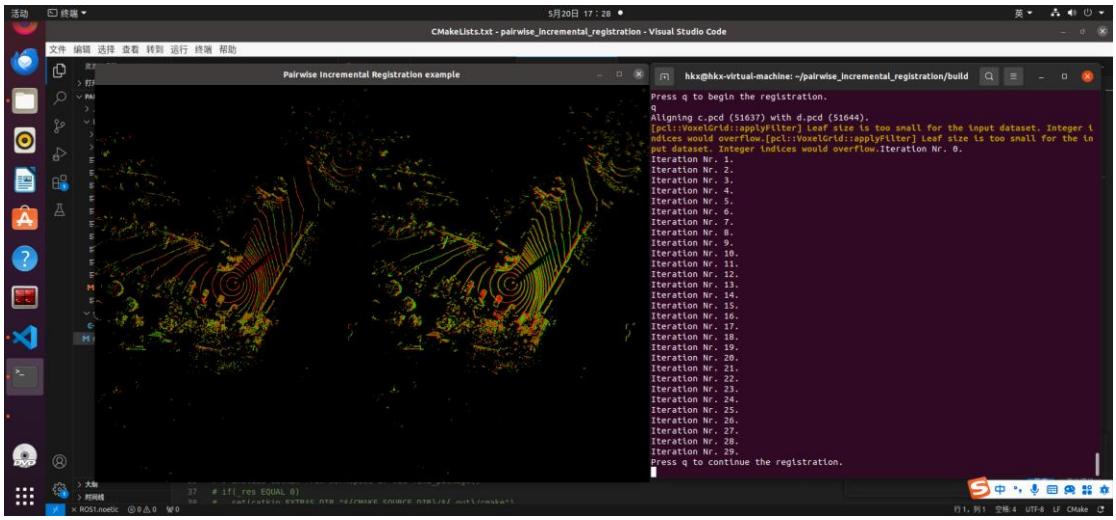


在迭代次数小于设定的次数 30 之前，右边会不断刷新最新的配准结果，直到收敛。如下图所示。图中左边和右边有明显的不同，右边显示配准后的结果，大体上观察其红色部分和绿色部分距离小于左边未配准的点云红绿对应之间的距离。



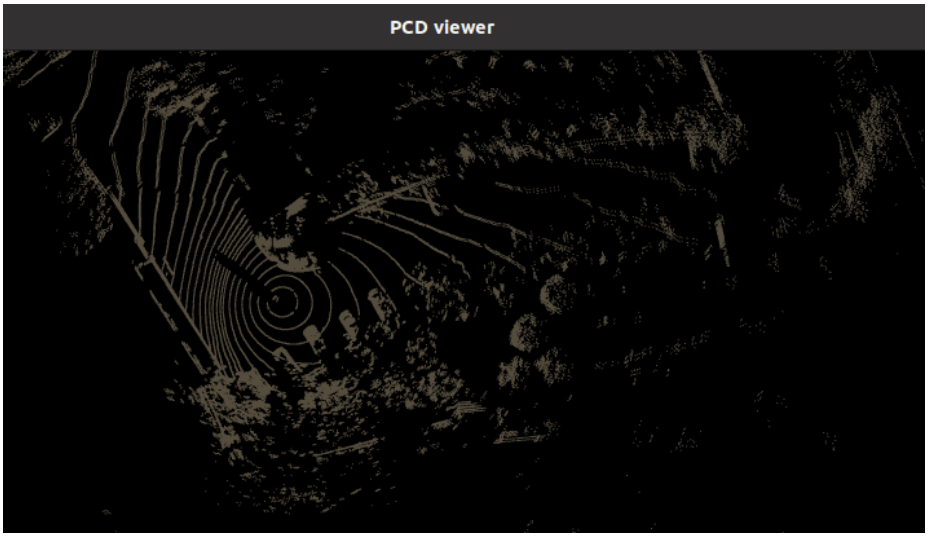
用户再次按键 q 则又会开始 b.pcd 与 c.pcd 间匹配，以此类推，直至相邻两个点云间都匹配完，并将最终的点云都变换到与第一个输入点云同一坐标系下，完成了对所有点云的配准。

同时退出程序，在同一目录下会看到存储的所以输出点云（1.pcd、2.pcd…），如 1.pcd 文件，此文件为第一和第二个输入点云配准后与第一个输入点云在同一坐标系下的点云。



再使用 pcl_viewer 命令查看其中一个生成的 pcd 文件如下：

```
hkx@hkx-virtual-machine:~/pairwise_incremental_registration/build$ pcl_viewer 1.pcd
```



7、需要建立一整张图，即可将读取到的点云数据重命名为 capture[0-1642].pcd，即可直接运行./ pairwise_incremental_registration_node capture[0-1642].pcd，然后等待匹配完成即可得到整张地图。

8、总结

（1）期间所遇问题：运行时显示屏为黑屏，以为代码错了，其实是需要调整相机视角，即使用鼠标滚筒来缩小页面，鼠标移动页面尝试从不同角度查看点云数据；

警告对于输入数据集来说，叶子大小太小，整数索引将溢出，即输入点云尺寸过大但 leaf size 太小，导致其数目不足以记录所有的，导致溢出。通用解决办法一般有若对精度要求不那么高，可将 leaf size 大小设置大一点，或将要采样的点云先分割成几块再做降采样。

（2）主要实现了从数据包中读取点云数据，再使用迭代最近点算法逐步匹配多幅点云，构建整张地图。

9、参考资料

PCL 中点云配准（非常详细，建议收藏）_pcl 点云配准-CSDN 博客

pcl-learning/14registration 配准/2 如何逐步匹配多幅点云 at master ·HuangCongQing/pcl-learning ·GitHub

附代码：

```
#include <boost/make_shared.hpp>
#include <pcl/point_types.h>
#include <pcl/point_cloud.h>
#include <pcl/point_representation.h>
#include <pcl/io/pcd_io.h>
#include <pcl/filters/voxel_grid.h>
#include <pcl/filters/filter.h>
#include <pcl/features/normal_3d.h>
#include <pcl/registration/icp.h>
#include <pcl/registration/icp_nl.h>
#include <pcl/registration/transforms.h>
#include <pcl/visualization/pcl_visualizer.h>
using pcl::visualization::PointCloudColorHandlerGenericField;
using pcl::visualization::PointCloudColorHandlerCustom;
typedef pcl::PointXYZ PointT;
typedef pcl::PointCloud<PointT> PointCloud;
typedef pcl::PointNormal PointNormalT;
typedef pcl::PointCloud<PointNormalT> PointCloudWithNormals;
using namespace std;

pcl::visualization::PCLVisualizer *p; //创建可视化工具
int vp_1, vp_2;                                    //定义左右视点

struct PCD//处理点云的方便的结构定义
{
    pcl::PointCloud<pcl::PointXYZ>::Ptr cloud;
    std::string f_name;
    PCD() : cloud (new pcl::PointCloud<pcl::PointXYZ>) {};
};
struct PCDComparator
{
    bool operator () (const PCD& p1, const PCD& p2)
    {
        return (p1.f_name < p2.f_name);
    }
};

class MyPointRepresentation : public pcl::PointRepresentation <PointNormalT>//以< x, y, z,
curvature >形式定义一个新的点
{
    using pcl::PointRepresentation<PointNormalT>::nr_dimensions_;
public:
    MyPointRepresentation ()
```

```
    {
        nr_dimensions_ = 4;     //定义尺寸值
    }
    virtual void copyToFloatArray (const PointNormalT &p, float * out) const //覆盖
copyToFloatArray 方法来定义我们的特征矢量
    {
        out[0] = p.x;
        out[1] = p.y;
        out[2] = p.z;
        out[3] = p.curvature;
    }
};

void showCloudsLeft(const PointCloud::Ptr cloud_target, const PointCloud::Ptr cloud_source)//窗
口左边显示源点云和目标点云
{
    p->removePointCloud ("vp1_target");
    p->removePointCloud ("vp1_source");
    PointCloudColorHandlerCustom<PointT> tgt_h (cloud_target, 0, 255, 0);
    PointCloudColorHandlerCustom<PointT> src_h (cloud_source, 255, 0, 0);
    p->addPointCloud (cloud_target, tgt_h, "vp1_target", vp_1);
    p->addPointCloud (cloud_source, src_h, "vp1_source", vp_1);
    PCL_INFO ("Press q to begin the registration.\n");
    p-> spin();
}

void showCloudsRight(const PointCloudWithNormals::Ptr cloud_target, const
PointCloudWithNormals::Ptr cloud_source)//窗口右边显示匹配好的点云
{
    p->removePointCloud ("source");
    p->removePointCloud ("target");
    PointCloudColorHandlerGenericField<PointNormalT> tgt_color_handler (cloud_target,
"curvature");
    if (!tgt_color_handler.isCapable ())
    {
        PCL_WARN ("Cannot create curvature color handler!");
    }
    PointCloudColorHandlerGenericField<PointNormalT> src_color_handler (cloud_source,
"curvature");
    if (!src_color_handler.isCapable ())
    {
        PCL_WARN ("Cannot create curvature color handler!");
    }
    p->addPointCloud (cloud_target, tgt_color_handler, "target", vp_2);
```

```
    p->addPointCloud (cloud_source, src_color_handler, "source", vp_2);
    p->spinOnce();
}

void loadData (int argc, char **argv, std::vector<PCD, Eigen::aligned_allocator<PCD> >
&models) //加载一组要匹配在一起的 PCD 文件, argc 是参数的数量, argv 为实际的命令行参
数，models 为点云数据集的合成矢量
{
    std::string extension (".pcd");
    for (int i = 1; i < argc; i++)
    {
        std::string fname = std::string (argv[i]);
        if (fname.size () <= extension.size ())//检查长度是否小于等于扩展名长度，是则跳过
        {
            continue;
        }
        std::transform (fname.begin (), fname.end (), fname.begin (), (int(*)(int))tolower);//将文
件名转为小写
        if (fname.compare (fname.size () - extension.size (), extension.size (), extension) == 0)//
检查参数是否为 pcd 文件
        {
            PCD m;
            m.f_name = argv[i];//将 pcd 对象 m 文件名设为当前命令行参数
            pcl::io::loadPCDFile (argv[i], *m.cloud);//加载 pcd 文件
            std::vector<int> indices;
            pcl::removeNaNFromPointCloud(*m.cloud,*m.cloud, indices); //从点云中移除
NAN 点
            models.push_back (m);//将清理过的 pcd 对象 m 添加回 models 中
        }
    }
}

void pairAlign (const PointCloud::Ptr cloud_src, const PointCloud::Ptr cloud_tgt, PointCloud::Ptr
output, Eigen::Matrix4f &final_transform, bool downsample = false)//匹配一对点云数据集并且
返回结果,cloud_src 和 tgt 为源、目标点云，output 为输出的配准结果的源点云，final_transform
为源点云和目标点云间转换矩阵
{
    PointCloud::Ptr src (new PointCloud);
    PointCloud::Ptr tgt (new PointCloud);
    pcl::VoxelGrid<PointT> grid;//下采样
    if (downsample)//downsample 为真即对 src tgt 下采样以减少点数提高效率
    {
        grid.setLeafSize (0.05, 0.05, 0.05);
        grid.setInputCloud (cloud_src);
```

```
        grid.filter (*src);
        grid.setInputCloud (cloud_tgt);
        grid.filter (*tgt);
    }
    else//为假则直接使用原点云
    {
        src = cloud_src;
        tgt = cloud_tgt;
    }
    PointCloudWithNormals::Ptr points_with_normals_src (new PointCloudWithNormals);//创建用于存储有法线和曲率的点云
    PointCloudWithNormals::Ptr points_with_normals_tgt (new PointCloudWithNormals);
    pcl::NormalEstimation<PointT, PointNormalT> norm_est;// 计算曲面法线和曲率
    pcl::search::KdTree<pcl::PointXYZ>::Ptr tree (new pcl::search::KdTree<pcl::PointXYZ> ());//法线估计时搜索临近点
    norm_est.setSearchMethod (tree);
    norm_est.setKSearch (30);//搜索半径
    norm_est.setInputCloud (src);//输入点云
    norm_est.compute (*points_with_normals_src);//计算
    pcl::copyPointCloud (*src, *points_with_normals_src);//复制
    norm_est.setInputCloud (tgt);//同上
    norm_est.compute (*points_with_normals_tgt);
    pcl::copyPointCloud (*tgt, *points_with_normals_tgt);
    MyPointRepresentation point_representation;
    float alpha[4] = {1.0, 1.0, 1.0, 1.0};//相同权重
    point_representation.setRescaleValues (alpha);//调整'curvature'尺寸权重以便使它和 x, y, z 平衡
    pcl::IterativeClosestPointNonLinear<PointNormalT, PointNormalT> reg;//创建 ICP 对象
    reg.setTransformationEpsilon (1e-6);//设收敛阈值
    reg.setMaxCorrespondenceDistance (0.1); //设两对应间最大距离为 10 厘米，需根据数据集大小调整
    reg.setPointRepresentation (boost::make_shared<const MyPointRepresentation> (point_representation));  //设置点表示
    reg.setInputCloud (points_with_normals_src);
    reg.setInputTarget (points_with_normals_tgt);

    Eigen::Matrix4f Ti = Eigen::Matrix4f::Identity (), prev, targetToSource;//4x4 变换矩阵
    PointCloudWithNormals::Ptr reg_result = points_with_normals_src;//存储匹配结果的点云指针
    reg.setMaximumIterations (2);//设最大迭代次数
    for (int i = 0; i < 30; ++i)//迭代
    {
        PCL_INFO ("Iteration Nr. %d.\n", i);
        points_with_normals_src = reg_result;//更新源点云为上次匹配结果
```

```
            reg.setInputCloud (points_with_normals_src);
            reg.align (*reg_result);
            Ti = reg.getFinalTransformation () * Ti;//在每一个迭代之间累积转换
            //如果这次转换和之前转换之间的差异小于阈值，则通过减小最大对应距离来改善
程序
            if (fabs ((reg.getLastIncrementalTransformation () - prev).sum ()) <
reg.getTransformationEpsilon ())
            {
                reg.setMaxCorrespondenceDistance (reg.getMaxCorrespondenceDistance () -
0.001);
                prev = reg.getLastIncrementalTransformation ();
            }
            showCloudsRight(points_with_normals_tgt, points_with_normals_src);   //可视化当前
状态
        }
        targetToSource = Ti.inverse();   //得到目标点云到源点云的变换
        //把目标点云转换回源框架
        pcl::transformPointCloud (*cloud_tgt, *output, targetToSource);
        p->removePointCloud ("source");
        p->removePointCloud ("target");
        PointCloudColorHandlerCustom<PointT> cloud_tgt_h (output, 0, 255, 0);
        PointCloudColorHandlerCustom<PointT> cloud_src_h (cloud_src, 255, 0, 0);
        p->addPointCloud (output, cloud_tgt_h, "target", vp_2);//添加到右可视器中
        p->addPointCloud (cloud_src, cloud_src_h, "source", vp_2);
        PCL_INFO ("Press q to continue the registration.\n");
        p->spin ();
        p->removePointCloud ("source");
        p->removePointCloud ("target");
        /*  添加源点云到转换目标  */
        *output += *cloud_src;
        final_transform = targetToSource;
}

int main (int argc, char** argv)
{
        std::vector<PCD, Eigen::aligned_allocator<PCD> > data;
        loadData (argc, argv, data);//加载数据
        if (data.empty ())//检查用户输入，若为空输出错误消息并返回 1，不为空则输出已加载
的数据集数量
        {
            PCL_ERROR ("Syntax is: %s <source.pcd> <target.pcd> [*]", argv[0]);
            PCL_ERROR ("[*] - multiple files can be added. The registration results of (i, i+1) will
be registered against (i+2), etc");
            PCL_INFO ("Example: %s `rospack find pcl`/test/bun0.pcd `rospack find
```

```
pcl`/test/bun4.pcd", argv[0]);
        return (-1);
    }
    PCL_INFO ("Loaded %d datasets.", (int)data.size ());

    p = new pcl::visualization::PCLVisualizer (argc, argv, "Pairwise Incremental Registration example");//创建一个 PCL 可视化对象
    p->createViewPort (0.0, 0, 0.5, 1.0, vp_1); //创建两个视图窗口
    p->createViewPort (0.5, 0, 1.0, 1.0, vp_2);
    PointCloud::Ptr result (new PointCloud), source, target;//result 用于存储匹配后的点云
    Eigen::Matrix4f GlobalTransform = Eigen::Matrix4f::Identity (), pairTransform;//前为存储全局变换矩阵，后为存储两两匹配时变换矩阵
    for (size_t i = 1; i < data.size (); ++i)//对 data 中点云进行两两匹配
    {
        source = data[i-1].cloud;//获取源点云与目标点云
        target = data[i].cloud;
        showCloudsLeft(source, target);//左侧窗口显示点云
        PointCloud::Ptr temp (new PointCloud);
        PCL_INFO ("Aligning %s (%d) with %s (%d).\n", data[i-1].f_name.c_str (), source->points.size (), data[i].f_name.c_str (), target->points.size ());
        pairAlign (source, target, temp, pairTransform, true);//匹配并返回匹配后的点云及变换矩阵
        pcl::transformPointCloud (*temp, *result, GlobalTransform); //把当前的两两配对转换到全局变换
        GlobalTransform = pairTransform * GlobalTransform;                //更新全局变换
        std::stringstream ss;
        ss << i << ".pcd";
        pcl::io::savePCDFile (ss.str (), *result, true);//保存配准对，转换到第一个点云框架中
    }
}
```