**激光PointCloud2点云数据的python发送与接收**

[1. 概述 1](#_Toc10910)

[2. 点云发送python版本 1](#_Toc26449)

[3. 点云接收python版本 2](#_Toc8658)

[4. c++完整版本 3](#_Toc20913)

[5. 总结 4](#_Toc20577)

[6. 利用python读取rosbag包中的点云数据 5](#_Toc25620)

[7. 参考文献 5](#_Toc2467)

# 概述

在《动手学ROS（11）：图像传输》中我们已经接触过图片的传输方法，本节我们来关注另一种常用的数据——点云的发送与接收方法。

点云通常是通过深度相机（RGB-D）或激光雷达（lidar）等传感器产生，是空间3D点的集合。点云一般应用在需要知道深度信息（或者说是距离）的场景，例如自动驾驶时需要获取前车的距离。普通的的双目视觉算法也可以计算出深度，但相对复杂且计算量大，通过增加深度传感器即可实时获取深度信息，提高系统的实时性。

在我的另一个系列教程《3D目标检测（6）：点云格式转换(pcd、bin、npy)》中简单介绍过点云的常见格式及转换方法，有需要的读者可进行参考。由于在该文中已经介绍过点云文件（pcd/bin）的读取方法，本文就不再赘述，因此这里就简单手动造一些点，作为示例。

运行环境

Ubuntu 20.04

ROS Noetic

实际调试中，ubuntu18.04对应的版本也可以

# 点云发送python版本

#!/usr/bin/env python3

# -\*- coding: UTF-8 -\*-

import rospy

from sensor\_msgs.msg import PointCloud2

from sensor\_msgs.msg import PointField

import numpy as np

def talker():

pub = rospy.Publisher('pointcloud\_topic', PointCloud2, queue\_size=5)

rospy.init\_node('pointcloud\_publisher\_node', anonymous=True)

rate = rospy.Rate(1)

points=np.array([[225.0, -71.0, 819.8],[237.0, -24.0, 816.0],[254.0, -82.0, 772.3]])

while not rospy.is\_shutdown():

msg = PointCloud2()

msg.header.stamp = rospy.Time().now()

msg.header.frame\_id = "livox\_frame"

if len(points.shape) == 3:

msg.height = points.shape[1]

msg.width = points.shape[0]

else:

msg.height = 1

msg.width = len(points)

msg.fields = [

PointField('x', 0, PointField.FLOAT32, 1),

PointField('y', 4, PointField.FLOAT32, 1),

PointField('z', 8, PointField.FLOAT32, 1)]

msg.is\_bigendian = False

msg.point\_step = 12

msg.row\_step = msg.point\_step \* points.shape[0]

msg.is\_dense = False

msg.data = np.asarray(points, np.float32).tostring()

pub.publish(msg)

print("published...")

rate.sleep()

if \_\_name\_\_ == '\_\_main\_\_':

talker()

# 点云接收python版本

#!/usr/bin/env python3

# -\*- coding: UTF-8 -\*-

import rospy

from sensor\_msgs.msg import PointCloud2

from sensor\_msgs import point\_cloud2

class PointCloudSubscriber(object):

def \_\_init\_\_(self): # -> None:

self.sub = rospy.Subscriber("pointcloud\_topic",

PointCloud2,

self.callback, queue\_size=5)

def callback(self, msg):

assert isinstance(msg, PointCloud2)

# gen=point\_cloud2.read\_points(msg,field\_names=("x","y","z"))

points = point\_cloud2.read\_points\_list(

msg, field\_names=("x", "y", "z"))

print(points)

if \_\_name\_\_ =='\_\_main\_\_':

rospy.init\_node("pointcloud\_subscriber")

PointCloudSubscriber()

rospy.spin()

# c++完整版本

#include <ros/ros.h>

#include <sensor\_msgs/PointCloud2.h>

#include <pcl\_conversions/pcl\_conversions.h>

#include <pcl/point\_types.h>

#include <pcl/PCLPointCloud2.h>

#include <pcl/conversions.h>

class pointcloud\_pub\_sub

{

private:

ros::NodeHandle nh;

ros::Publisher pub;

ros::Subscriber sub;

std::string pointcloud\_topic = "cloudpoint\_topic";

void initializePublishers();

void initializeSubscribers();

void callback(const sensor\_msgs::PointCloud2ConstPtr &cloud);

public:

pointcloud\_pub\_sub(ros::NodeHandle \*node);

~pointcloud\_pub\_sub();

void publish();

};

pointcloud\_pub\_sub::pointcloud\_pub\_sub(ros::NodeHandle \*node) : nh(\*node)

{

initializePublishers();

initializeSubscribers();

}

pointcloud\_pub\_sub::~pointcloud\_pub\_sub()

{

}

void pointcloud\_pub\_sub::initializePublishers()

{

pub = nh.advertise<sensor\_msgs::PointCloud2>(pointcloud\_topic, 10);

}

void pointcloud\_pub\_sub::initializeSubscribers()

{

sub = nh.subscribe(pointcloud\_topic, 10, &pointcloud\_pub\_sub::callback, this);

}

void pointcloud\_pub\_sub::callback(const sensor\_msgs::PointCloud2ConstPtr &cloud)

{

pcl::PointCloud<pcl::PointXYZ>::Ptr temp\_cloud(new pcl::PointCloud<pcl::PointXYZ>);

pcl::fromROSMsg(\*cloud, \*temp\_cloud);

// or

pcl::PCLPointCloud2 pcl\_pc2;

pcl\_conversions::toPCL(\*cloud, pcl\_pc2);

pcl::fromPCLPointCloud2(pcl\_pc2, \*temp\_cloud);

//do stuff

ROS\_INFO("received %ld points", temp\_cloud->points.size());

}

void pointcloud\_pub\_sub::publish()

{

sensor\_msgs::PointCloud2 msg;

pcl::PointCloud<pcl::PointXYZ>::Ptr cloud(new pcl::PointCloud<pcl::PointXYZ>);

for (size\_t i = 0; i < 3; i++)

{

pcl::PointXYZ p(1.0 \* i, 2.0 \* i, 3.0 \* i);

cloud->push\_back(p);

}

pcl::toROSMsg(\*cloud, msg);

// or

pcl::PCLPointCloud2 pcl\_pc2;

pcl::toPCLPointCloud2(\*cloud, pcl\_pc2);

pcl\_conversions::fromPCL(pcl\_pc2, msg);

// publish

pub.publish(msg);

ROS\_INFO("published.");

}

# 总结

pcl库是一个强大的点云处理库，在c++中使用极其方便，而python版本中因为笔者没有安装python-pcl（也是为了追求多样性），代码看起来略显繁杂。

同时，c++版本中也展示了ros中sensor\_msgs::PointCloud2格式与pcl中pcl::PointCloud、pcl::PointCloud的相互转换方法，请读者留意。

# 利用python读取rosbag包中的点云数据

知道了图像的提取方法，提取点云的就简单了，直接上代码：

import rosbag

import numpy as np

import sensor\_msgs.point\_cloud2 as pc2

bag\_file = 'test.bag'

bag = rosbag.Bag(bag\_file, "r")

bag\_data = bag.read\_messages('/PointCloud2')

for topic, msg, t in bag\_data:

lidar = pc2.read\_points(msg)

points = np.array(list(lidar))

# 看你想如何处理这些点云

这里用到了 pc2.read\_points() ，其接受一个 sensor\_msgs.PointCloud2 对象，然后会返回一个迭代器，每次迭代返回点云中的一个点。

rosbag play

回放数据包中的 topic：

rosbag play <bagfile>

开始播放立刻暂停，按空格继续：

rosbag play --pause bag\_name.bag

如果想改变消息的发布速率，可以用下面的命令，-r 后面的数字对应播放速率：

rosbag play -r 2 <bagfile>

如果希望 rosbag 循环播放，可以用命令：

rosbag play -l <bagfile> # -l== --loop

如果只播放感兴趣的 topic ，则用命令：

rostopic list

# 参考文献

原文链接：<https://blog.csdn.net/lemonxiaoxiao/article/details/121693548>

原文链接：<https://blog.csdn.net/MemoryD/article/details/105174348>