

车载软件设计基础

松灵小车任务1



**2023至2024学年第 1 学期**

|  |  |  |
| --- | --- | --- |
| 学号 | 姓名 | 团队作用 |
| E2020196 | 侯加喜 | 组内任务分配、程序编写 |
| E2020017 | 费川川 | 程序编写,问题记录 |
| 任课教师 | 陈恒鑫 | |
| 成 绩 |  | |

|  |  |
| --- | --- |
| 任务书 | |
| 任务内容 | 实验分组进行，每组人数不超过5人。在松灵小车上编写ROS程序，实现如下功能：  （1）开启松灵小车的激光雷达、深度相机等传感器，控制小车移动（遥控器），编写程序，显示移动过程中，各类传感器数据：   * 命令行窗口显示小车的线速度和角速度 * OpenCV显示深度相机的RGB图像和深度图 * PCL显示激光雷达的点云数据   （2）编程实现用里程计（odometry）计算小车移动距离   * 小车靜止不动，读取里程计数据，记为a。控制小车前进n米距离(n=1、2、3均可），读取里程计数据，记为b * 建立小车移动距离与里程计读数a、b之间的关系模型（ 前两步应进行多次以拟合更精确的模型） * 控制小车移动，利用上一步构建的模型，计算小车移动的距离，并将计算结果与实际值进行对比 |
| 程序规范 | （1）所有程序代码采用C++编写，使用git进行源代码管理；  （2）类名、变量名、函数名应符合C++的命名规范，并在代码中前后保持一致；  （3）涉及面向对象的程序，例如自定义的类，应符合面向对象的设计原则；  （4）正确使用头文件和源文件，自定义的头文件应符合头文件的编写原则，例如用条件宏定义确保头文件不被多次引用、不在头文件中进行类和函数的实现（模板除外）；  （5）项目必须是ROS项目，符合ROS的项目的规范，正确编写CmakeLists.txt等文件；  （6）程序能够在松灵小车上运行。 |
| 报告要求 | （1）报告至少应该包括人员分工、需求分析、程序设计、程序效果展示、总结分析4个部分；  （2）人员分工介绍组员各自的工作情况；  （3）需求分析侧重描述程序所需要实现的功能，功能预期的效果；  （4）程序设计描述组成程序的模块、类、函数以及他们之间的相互关系，若有算法，可以描述算法流程；  （5）程序效果展示除了程序运行效果截图之外，应该有必要的文字说明；  （6）总结分析可以分析实现的效果与理想情况的差异，分析导致这些差异的原因，切忌不要写成心得体会；  （7）报告应该格式规范、排版整洁、少语病和错误。 |
| 作业提交 | （1）含有git仓库（有.git目录）的完整源代码；  （2）程序功能演示的讲解视频；  （3）任务报告。 |
| 评分标准 | 按照五级制打分，分为优秀、良好、中等、及格、不及格，各评分项占总成绩的比例为：  （1）任务完成情况占评分的60%；  （2）程序规范占评分的20%；  （3）报告占评分的20%。  评分老师根据各部分的完成情况，直接给出总成绩。 |

1. **人员分工**

1.侯加喜: 担任组长,负责组内工作分配,协调各组员按时完成任务,组织小组会议讨论遇到的困难并及时解决。同时负责OpenCV显示深度相机的RGB图像和深度图,PCL显示激光雷达的点云数据、里程计部分代码编写。

2.费川川: 负责完成命令行窗口显示小车的线速度和角速、负责里程计任务中数据收集,各任务过程记录,重点记录遇到问题及解决方法。

1. **需求分析**

1.任务一：命令行窗口显示小车的线速度和角速度

根据任务要求，要将小车的线速度和角速度显示到命令行窗口，可以直接通过rostopic echo命令打印相关的话题信息即可。当然也可以编写ROS程序接收话题并打印在屏幕上。

2.任务二：OpenCV显示深度相机的RGB图像和深度图

首先，ROS的核心是采用观察者模式，发布者与接收者需要通过独立的topic进行通信。我们平时可

以通过Rviz接收RGB图像和深度图，其中可以看到它们所使用的topic分别为

/camera/color/image\_raw和/camera/depth/image\_rect\_raw。那么只需要编写ROS程序接收这两个

话题信息即可，但是ROS中的格式与opencv所使用的图像格式有差异，需要通过cv\_bridge进行换。

3.任务三：PCL显示激光雷达的点云数据

与任务2类似，通过Rviz可以得知激光雷达的话题为/rslidar\_points，那么只需编写ROS程序订阅该话题信息，并新建窗口进行展示即可。需要注意的是，PCL使用的点云格式为pcl::PCLPointCloud2，而ROS使用的点云数据为sensor\_msgs::PointCloud2，需要先转换为PCL使用的格式后才能加以展示。

4.任务四：编程实现用里程计（odometry）计算小车移动距离

通过打印/odom话题信息可以拿到小车的x，y坐标信息，根据直角三角形可知，小车的移动距离就是该三角形的斜边长,所以在程序中订阅/odom话题信息拿到x，y位置坐标并实时计算距离起始点的距离即可，并将移动的距离打印在屏幕上。除此之外还可以通过速度,时间的积分计算出小车移动端距离。

1. **程序设计**

3.1 任务一：本次任务通过编写ROS节点订阅,打印出小车线速度,角速度以及角位移,角偏转,位姿信息。完整源代码如下:

#include <ros/ros.h>

#include <nav\_msgs/Odometry.h>

void speedCallback(const nav\_msgs::Odometry::ConstPtr &msg)

{

    //读取激光雷达相关的话题/odom，可以获取小车的线速度和角速度

    /\*   靠 里程计信息 计算出来的

    位移  (map坐标系，以现实世界 东南西北为准)

    线位移： position.x

    角位移： orientation.z : 正向0  (左加右减)   (.x/.y 在此无用)

    车朝向： orientation.w : 正向1,反向0

    速度 （base\_link 坐标系，以车朝向为准）

    线速度： linear.x   (前正后负)   (.y/.z 在此无用)

    角速度： angular.z  (左正右负)   (.x/.y 在此无用)

    \*/

    setlocale(LC\_ALL, "");

    ROS\_INFO("角速度： wz=%0.2f ", msg->twist.twist.angular.z);

    ROS\_INFO("线速度： vx=%0.2f \n", msg->twist.twist.linear.x);

    ROS\_INFO("角位移： Wx=%0.2f ", msg->pose.pose.orientation.z);

    ROS\_INFO("角偏转： Wz=%0.2f ", msg->pose.pose.orientation.w);

    ROS\_INFO("线位移： SX=%0.2f  SY=%0.2f \n\n", msg->pose.pose.position.x, msg->pose.pose.position.y);

    return;

}

int main(int argc, char \*\*argv)

{

    ros::init(argc, argv, "nav\_subscriber");

    ros::NodeHandle n;

    //roslaunch scout\_bringup open\_rslidar.launch

    ros::Subscriber speed\_sub = n.subscribe("/odom", 10, speedCallback);

    ros::spin();

    return 0;

}

3.2任务二：先通过opencv中的namedWindow命令新建两个窗口，分别展示RGB彩色图和深度图。再通过Subscriber订阅/camera/depth/image\_rect\_raw和/camera/color/image\_raw两个话题。它们调用各自的回调函数进行处理。在回调函数中，需要通过cv\_bridge将ROS中的图像格式转换为opencv中使用的格式，并指定编码格式来展示RGB图和深度图。最后通过cv::imshow进行窗口展示即可。

完整源码如下：

#include <ros/ros.h>

#include <image\_transport/image\_transport.h>

#include <cv\_bridge/cv\_bridge.h>

#include <sensor\_msgs/image\_encodings.h>

#include <opencv2/imgproc/imgproc.hpp>

#include <opencv2/highgui/highgui.hpp>

static const std::string OPENCV\_WINDOW\_COLOR = "彩色相机";

static const std::string OPENCV\_WINDOW\_DEPTH = "深度相机";

class ImageConverter

{

  ros::NodeHandle nh\_;

  image\_transport::ImageTransport it\_;

  image\_transport::Subscriber image\_sub\_color;

  image\_transport::Subscriber image\_sub\_depth;

public:

  ImageConverter() : it\_(nh\_)

  {

    image\_sub\_color = it\_.subscribe("/camera/color/image\_raw", 1, &ImageConverter::imageCb, this);

    image\_sub\_depth = it\_.subscribe("camera/depth/image\_rect\_raw",1,&ImageConverter::depthCb, this);

    cv::namedWindow(OPENCV\_WINDOW\_COLOR);

    cv::namedWindow(OPENCV\_WINDOW\_DEPTH);

  }

  ~ImageConverter()

  {

    cv::destroyWindow(OPENCV\_WINDOW\_COLOR);

    cv::destroyWindow(OPENCV\_WINDOW\_DEPTH);

  }

  void imageCb(const sensor\_msgs::ImageConstPtr& msg)

  {

    cv\_bridge::CvImagePtr cv\_ptr;

    try

    {

      cv\_ptr = cv\_bridge::toCvCopy(msg, sensor\_msgs::image\_encodings::BGR8);

    }

    catch (cv\_bridge::Exception& e)

    {

      ROS\_ERROR("cv\_bridge exception: %s", e.what());

      return;

    }

    // 刷新窗口

    cv::imshow(OPENCV\_WINDOW\_COLOR, cv\_ptr->image);

    cv::waitKey(3);

    return;

  }

  void depthCb(const sensor\_msgs::ImageConstPtr &msg)

  {

    cv\_bridge::CvImagePtr cv\_ptr;

    try

    {

      cv\_ptr = cv\_bridge::toCvCopy(msg, sensor\_msgs::image\_encodings::TYPE\_32FC1);

    }

    catch (cv\_bridge::Exception& e)

    {

      ROS\_ERROR("cv\_bridge exception: %s", e.what());

      return;

    }

    // 刷新窗口

    cv::imshow(OPENCV\_WINDOW\_DEPTH, cv\_ptr->image);

    cv::waitKey(3);

    return;

  }

};

int main(int argc, char\*\* argv)

{

  ros::init(argc, argv, "image\_subscriber");

  ImageConverter ic;

  ros::spin();

  return 0;

}

3.3任务三：该任务需要将激光雷达的点云数据通过PCL窗口进行展示，其实现思路与任务2大同小异，首先也是通过Subscriber订阅激光雷达的话题/rslidar\_points，并在回调函数中进一步处理：

(1).将传入的ROS格式的sensor\_msgs::PointCloud2点云格式转换为PCL需要的pcl::PCLPointCloud2格式的点云数据。

(2).通过PCLVisualizer声明一个view，设置背景颜色，点云线粗细等参数。

(3).PCL格式点云添加到view中，如果需要颜色信息需要通过PointCloudColorHandlerGenericField进行添加。

(4).通过addPointCloud和updatePointCloud持续添加点云和更新点云信息，其中updatePointCloud函数尤为关键，如果缺失，那么显示的图像只有一帧。而非动态刷新的实时数据。

完整代码如下：

#include <iostream>

#include <ros/ros.h>

#include <sensor\_msgs/PointCloud2.h>

#include <string.h>

#include <pcl/io/pcd\_io.h>

#include <pcl/io/io.h>

#include <pcl/point\_types.h>

#include <pcl/point\_cloud.h>

#include <pcl/visualization/pcl\_visualizer.h>

#include <boost/thread/thread.hpp>

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

using namespace std;

boost::shared\_ptr<pcl::visualization::PCLVisualizer> viewer1(new pcl::visualization::PCLVisualizer("realtime pcl"));

ros::Publisher pub;

typedef pcl::PointCloud<pcl::PointXYZ> PointCloud;

void cloud\_cb (const sensor\_msgs::PointCloud2ConstPtr& input)

{

   // 声明存储原始数据与滤波后的数据的点云的格式

  pcl::PCLPointCloud2\* cloud = new pcl::PCLPointCloud2;    //原始的点云的数据格式

  pcl::PCLPointCloud2ConstPtr cloudPtr(cloud);

  // 转化为PCL中的点云的数据格式

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

  pub.publish (\*cloud);

  pcl::PointCloud<pcl::PointXYZ>::Ptr cloud1;

  cloud1.reset (new pcl::PointCloud<pcl::PointXYZ>);

  pcl::fromROSMsg (\*input, \*cloud1);

  viewer1->removeAllPointClouds();  // 移除当前所有点云

  viewer1->addPointCloud(cloud1, "实时PCL");

  viewer1->updatePointCloud(cloud1, "实时PCL");

  viewer1->spinOnce(0.001);

}

int main (int argc, char\*\* argv)

{

  ros::init (argc, argv, "pcl");

  ros::NodeHandle nh;

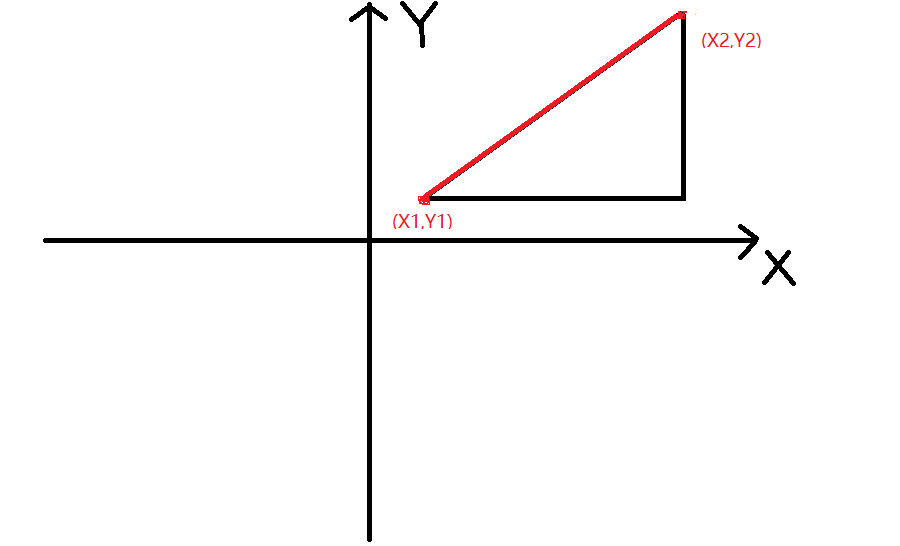
  ros::Subscriber sub = nh.subscribe ("/rslidar\_points", 10, cloud\_cb);

  pub = nh.advertise<pcl::PCLPointCloud2> ("output", 10);

  ros::spin ();

}

3.4任务四：如果小车只做直线运动，那么根据直角三角形可以直观看出：x，y坐标系下小车移动距离就是该直角三角形的斜边长度。因而此任务采用了两种基于map坐标计算和dt,dv积分计算两种方式进行对比。



(1).订阅接收/odom话题，其中包含小车的x，y坐标信息，在小车开始直线运动之前，记录当前的坐标信息保存为小车的起始坐标位置，然后取消订阅。

(2).在ROS程序中通过Subscriber接收/scout\_status话题，其中有小车的线速度、角速度等信息，通过判断线速度的大小可以推断出小车是否运动(注意消抖)。

(3).获取小车当前的坐标信息并与小车的起始坐标进行运算得到小车直线运动的距离。

其中小车距离运算公式为：

**dxy =  sqrt(pow(now\_x-pre\_x,2) + pow(now\_y-pre\_y,2));**

完整代码如下：

#include <ros/ros.h>

#include <nav\_msgs/Odometry.h>

#include <math.h>

#include <ctime>

float displacement = 0.0;   // base\_link位移

float distance = 0.0;       // 起始点开始，行进的路程

// 相对原点

float pre\_x = 0.0;

float pre\_y = 0.0;

float now\_x = 0.0;

float now\_y = 0.0;

float dxy = 0.0;

float dist\_xy = 0.0;

float fit\_dist\_xy = 0.0;

float loss = 0.03;

// 时间

time\_t pre\_time = time(nullptr);

time\_t now\_time = time(nullptr);

bool flag = true;

void doMsg(const nav\_msgs::Odometry::ConstPtr& msg)

{

    float wz = msg->twist.twist.angular.z;    // 角速度 wz

    if(wz<0.02 && wz>-0.02)

    {

        wz = 0.0;

    }

    float wx = msg->pose.pose.orientation.z;  // 角位移 wx

    float ww = msg->pose.pose.orientation.w;  // 角偏转 ww

    float vx = msg->twist.twist.linear.x;     // 线速度 vx

    if(vx<0.01 && vx>-0.01)

    {

        vx = 0.0;

    }

    now\_time = time(nullptr);;

    float dt = difftime(now\_time, pre\_time);

    float dx = dt\*vx;                        // 位移

    float ds = (dx>0 ? dx:0.0-dx);            // 距离

    displacement += dx;                       // base\_link 坐标系下正反向行进路程

    distance += (dx>0 ? dx:0.0-dx);           // 起始点开始，行进的路程

    now\_x = msg->pose.pose.position.x;

    now\_y = msg->pose.pose.position.y;

    dxy =  sqrt(pow(now\_x-pre\_x,2) + pow(now\_y-pre\_y,2));

    if(dxy > loss)

    {

        pre\_x = now\_x;

        pre\_y = now\_y;

        if(flag == true)

        {

            dist\_xy = 0.0;

            fit\_dist\_xy = 0.0;

        }

        else

        {

            dist\_xy += dxy;

            fit\_dist\_xy = 1.0183 \* dist\_xy + 0.0059;

        }

        flag = false;

    }

    ROS\_INFO("角速度： wz=%0.3f   角位移： wx=%0.3f   角偏转ww=%0.3f  ", wz, wx, ww);

    ROS\_INFO("线速度： vx=%0.3f", vx);

    ROS\_INFO("base\_link 位移：  disp=%0.3f  ", displacement);

    ROS\_INFO("积分vx 移动距离:  Sadd=%0.3f  ", distance);

    ROS\_INFO("Map坐标移动距离:  Smap=%0.3f  \n\n\n", fit\_dist\_xy);

    pre\_time = time(nullptr);

    return;

}

int main(int argc, char  \*argv[])

{

    setlocale(LC\_ALL,"");

    ros::init(argc, argv, "odom\_sub");

    ros::NodeHandle nh;

    pre\_time = time(nullptr);

    ros::Subscriber sub = nh.subscribe("/odom",1000,doMsg);

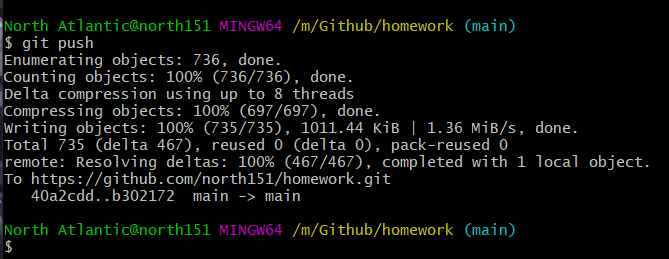
    ros::spin();

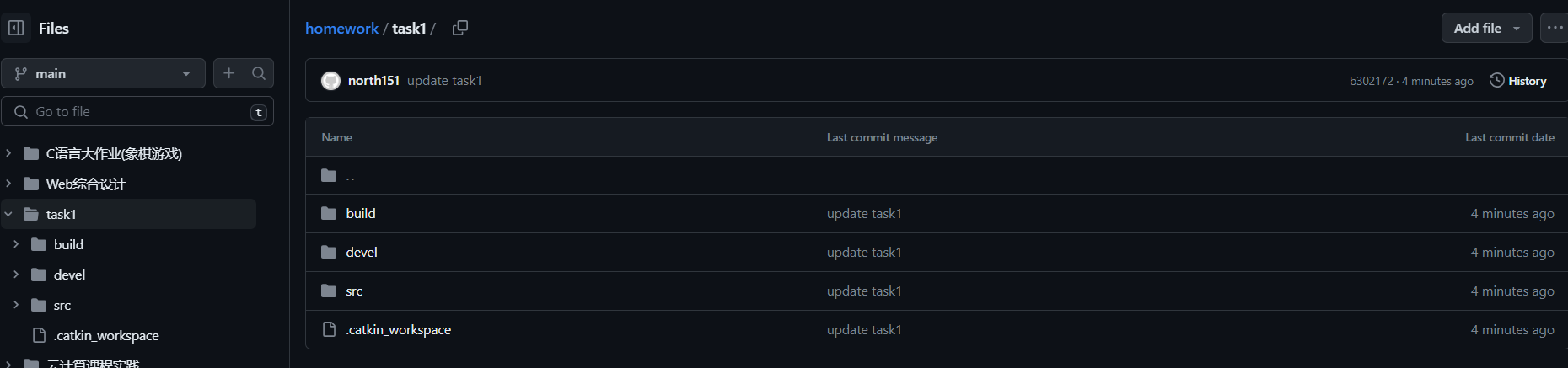
    return 0;

}

1. **程序效果展示**

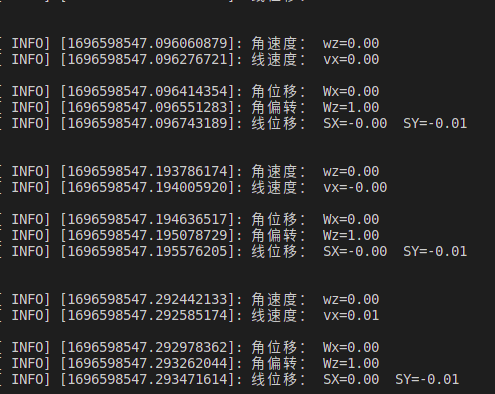
git版本控制(上传Github): https://github.com/north151/homework.git





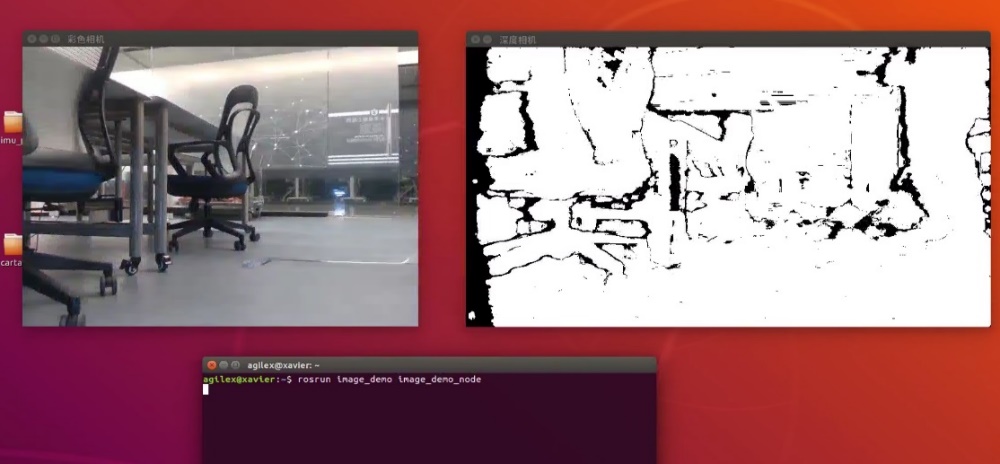
任务1：

通过优化使得静止时角速度、线速度为0。

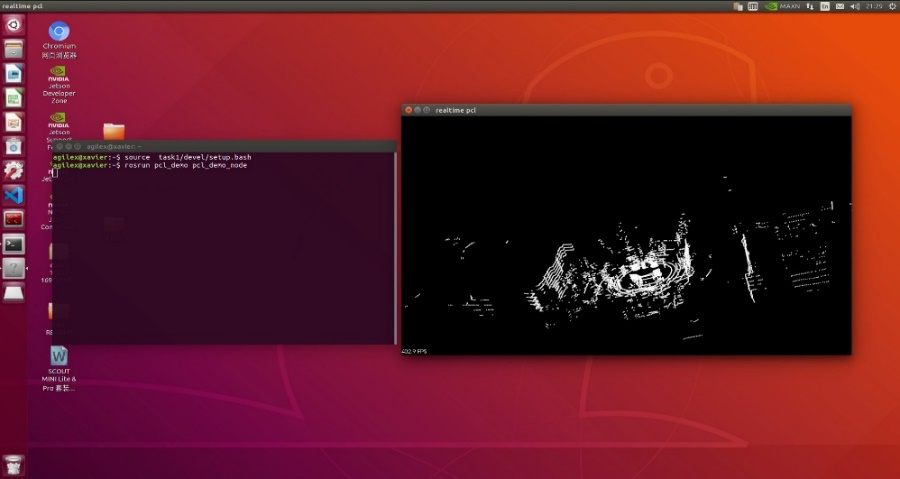


任务2：

左框为彩色相机显示,右框为深度相机显示

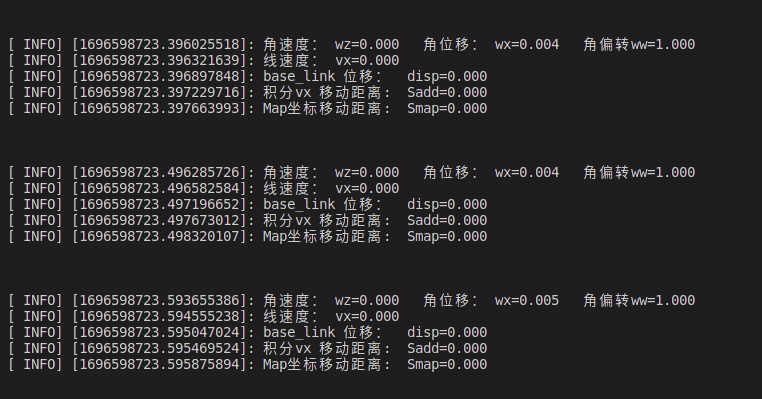


任务3：

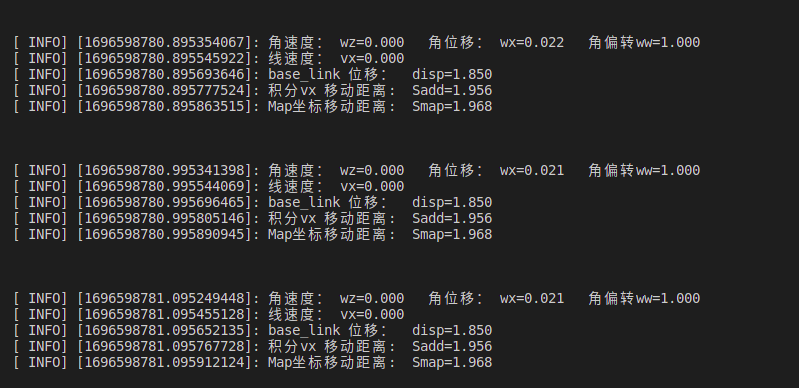


任务4：

小车运动前:



小车运动后:



1. **总结分析**

[Github](https://github.com/north151/carTask): https://github.com/north151/carTask

[Gitee](https://gitee.com/north151/car-task1): https://gitee.com/north151/car-task1

1.任务一通过话题/scout\_status读取角速度、线速度时，首先要启动底盘运动节点。在读取话题/odom获取小车坐标信息时还要启动深度相机和激光雷达，因为小车的x,y坐标是通过IMU和激光雷达进行计算的，所以通过/odom话题读取的坐标信息是在不断变化的，虽然程度很小，但这就是因为激光雷达扫描到的数据在不断变化造成的。

2.任务二中可以通过订阅相应的话题来获取RGB彩图和深度图，但是opencv中使用的图像格式与ROS中不同，所以要通过cv\_bridge进行转换后才能通过opencv展示。

3.任务三与任务2类似，要将ROS中的点云格式转换为PCL中的点云格式才能进行展示。此外，由于激光雷达原生扫描到的数据是没有颜色信息的，所以要想展示出来的点云是彩色的还需要通过pcl::visualization::PointCloudColorHandlerGenericField颜色渲染。

4.任务四计算小车直线运动距离时，要理解小车是在base\_link坐标系下运动，此时不涉及高度，所以不需要考虑Z坐标系。故就是计算直角三角形的斜边长。难点在于如何判断小车是运动状态还是静止状态，可以通过小车线速度的突变进行判断, 除此之外还要注意消抖。通过对比发现,小车移动2.04m的距离,速度对时间积分得到的结果为1.956m,坐标计算结果为1.968m。二者的差距分别为8.4cm和7.2cm以内,考虑到硬件因素和控制急停,此误差在可接受范围内。

1. **参考文献**

[1]如何在ROS中使用PCL.http://t.zoukankan.com/li-yao7758258-p-6651326.html,2020-11-21.

[2]openCV调用USB摄像头并发布topic.https://blog.csdn.net/Leslie\_92672,2022-03-01.

[3]PCL库学习笔记（为点云上色的几种方法详解）.http://t.csdn.cn/gRBO2,2021-06-21.

[4]使用ROS\_BRIDGE获取深度数据.https://www.freesion.com/article/5782869347/,2019-05-18.

[5]PCL点云可视化.https://cdn.modb.pro/db/488053,2022-08-17.

[6]在ROS中使用OpenCV进行简单的图像处理---代码实现篇.http://t.csdn.cn/9wmfg,2022-01-20.

[7][ROS机器人应用查看里程计、IMU 话题信息](http://www.wtld.cn/a/75093.html).http://www.wtld.cn/a/75093.html,2020-07-11.

[8]ROS学习（七）：三维可视化工具（RViz）.http://t.csdn.cn/RJEcV,2022-05-21.

[9]C/C++编写命名规范.http://t.csdn.cn/VzFcz,2018-04-28.