

车载软件设计基础

松灵小车任务3



**2022至2023学年第 1 学期**

|  |  |  |
| --- | --- | --- |
| 学号 | 姓名 | 团队作用 |
| E2021201 | 宋智鑫 | 程序封装和程序编写 |
| E2021146 | 王定彬 | 小车运动实现及程序编写 |
| E2021206 | 熊健霖 | 程序编写和问题分析 |
| E2021198 | 陈干 | 需求分析及程序编写 |
| 任课教师 | 刘骥 | |
| 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）使用（agilexrobotics）提供的SDK（https://github.com/agilexrobotics/ugv\_sdk）读取松灵小车的车灯（light）、电池（battery）、电机（motor）和运动（motion）状态，并显示；  （2）使用（agilexrobotics）提供的SDK操作松灵小车前进、后退和旋转；  （3）在（1）（2）的基础上，将程序封装为Ros程序，采用Topic通信模式，构建显示小车状态、控制小车运动的Ros程序（参考https://github.com/agilexrobotics/scout\_ros ）；  （4）通过scout\_teleop\_keyboard程序控制小车运动，在运动过程中用Rviz显示小车位置。 |
| 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陈干：负责需求分析、程序设计中关于读取小车状态的模块，并编写相应的C++代码。
* 2王定彬：负责实现控制小车运动的功能，并将其封装为ROS节点，同时负责相关的C++代码编写。
* 3宋智鑫：负责将所有功能整合成完整的ROS程序，包括Topic通信模式的设计与实现，以及Rviz显示小车位置的配置。
* 4熊健霖：负责文档撰写，包括报告中的人员分工、需求分析、程序设计、程序效果展示和总结分析。此外，还需负责git仓库的管理，确保代码提交规范。

1. **需求分析**

**任务1：**使用（agilexrobotics）提供的SDK（https://github.com/agilexrobotics/ugv\_sdk）读取松灵小车的车灯（light）、电池（battery）、电机（motor）和运动（motion）状态，并显示。

根据分析可知：只需要调用官方提供的ugv\_sdk中的demo\_scout\_robot 可执行文件即可。

1.从官网下载ugv\_sdk压缩包，解压到自己的ROS工作目录下，然后编译即可。我们需要执行的是demo目录下的scout\_robot\_demo，但由于CMakeLists.txt规定只有通过cmake编译时才会生成demo目录下的可执行文件，所以通过catkin\_make编译时要注释掉图中标注的两行。 这样才能生成我们需要使用的可执行文件。执行can使能命令并设置波特率为500000。然后执行 rosrun ugv\_sdk demo\_scout\_robot can0 命令来读取车辆的状态信息。结果如下，可以看到小车的线速度、角速度、电机温度、转速等信息。

图形用户界面, 文本

描述已自动生成

**任务2：**使用（agilexrobotics）提供的SDK操作松灵小车前进、后退和旋转；

**任务3**：在（1）（2）的基础上，将程序封装为Ros程序，采用Topic通信模式，构建显示小车状态、控制小车运动的Ros程序；

由于任务2与任务3高度相似，所以此处着重介绍任务3的需求分析。根据要求可知，如果ROS程序想操作小车，那么肯定要依赖于ugv\_sdk这个包。并且话题发布者持续监听用户键盘输入的线速度和角速度，话题接收者读取到用户输入的线速度和角速度，在回调函数中使用ugv\_sdk提供的方法**SetMotionCommand** 来控制小车。

图形用户界面, 文本

描述已自动生成

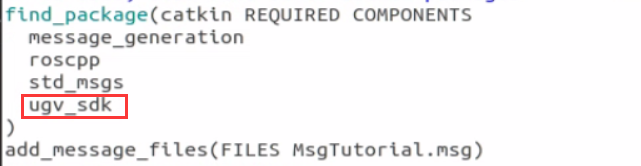
**任务4：**通过scout\_teleop\_keyboard程序控制小车运动，在运动过程中用Rviz显示小车位置。

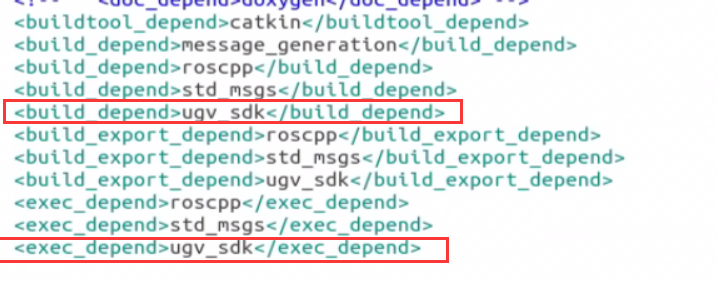
图形用户界面, 应用程序

描述已自动生成

**三、程序设计**

由于需要通过ROS程序控制小车，那么在CMakeList.txt文件中必须引入对应的依赖。





自定义需要使用的msg消息，只有两个变量：一个表示线速度、一个表示角速度

然后开始编写topic的发布方程序，该程序主要功能为持续监听用户输入的线速度、角速度然后发布

对于话题的接收方，功能为初始化小车状态，首先要能保证正确连接至小车。然后订阅之前发布的topic，读取msg消息，获取到里面包含的线速度、角速度。在对应的回调函数中调用ugv\_sdk中的相应方法来控制小车线速度、角速度（SetMotionCommand命令），以及小车的状态信息。

#include <unistd.h>

#include "ros/ros.h" // ROS的默认头文件

#include "ugv\_sdk/MsgTutorial.h" // MsgTutorial消息头文件（构建后自动生成）

#include <memory>

#include <iostream>

#include "ugv\_sdk/mobile\_robot/scout\_robot.hpp"

#include "ugv\_sdk/utilities/protocol\_detector.hpp"

using namespace westonrobot;

std::unique\_ptr<ScoutRobot> scout;

int count = 0;

void msgCallback(const ugv\_sdk::MsgTutorial::ConstPtr& msg)

{

//运动控制

std::cout << "linear\_velocity: " << msg->linear << " twist\_velocity: " << msg->twist << std::endl;

scout->SetMotionCommand(msg->linear, msg->twist);//设置线速度、角速度

// 获取小车状态

auto state = scout->GetRobotState();

std::cout << "-------------------------------" << std::endl;

std::cout << "count: " << count << std::endl;

std::cout << "control mode: "

<< static\_cast<int>(state.system\_state.control\_mode)

<< " , vehicle state: "

<< static\_cast<int>(state.system\_state.vehicle\_state)

<< " , error code: " << std::hex << state.system\_state.error\_code

<< std::dec

<< ", battery voltage: " << state.system\_state.battery\_voltage

<< std::endl;

std::cout << "velocity (linear, angular): "

<< state.motion\_state.linear\_velocity << ", "

<< state.motion\_state.angular\_velocity << std::endl;

std::cout << "core state age (ms): "

<< std::chrono::duration\_cast<std::chrono::milliseconds>(

AgxMsgRefClock::now() - state.time\_stamp)

.count()

<< std::endl;

//获取电机状态

auto actuator = scout->GetActuatorState();

if (scout->GetParserProtocolVersion() == ProtocolVersion::AGX\_V1) {

for (int i = 0; i < 4; ++i) {

printf("motor %d: current %f, rpm %d, driver temp %f, motor temp %f\n",

actuator.actuator\_state[i].motor\_id,

actuator.actuator\_state[i].current,

actuator.actuator\_state[i].rpm,

actuator.actuator\_state[i].driver\_temp,

actuator.actuator\_state[i].motor\_temp);

}

std::cout << "actuator state age (ms): "

<< std::chrono::duration\_cast<std::chrono::milliseconds>(

AgxMsgRefClock::now() - actuator.time\_stamp)

.count()

<< std::endl;

}

else {

for (int i = 0; i < 4; ++i) {

printf("motor %d: current %f, rpm %d, driver temp %f, motor temp %f\n",

actuator.actuator\_hs\_state[i].motor\_id,

actuator.actuator\_hs\_state[i].current,

actuator.actuator\_hs\_state[i].rpm,

actuator.actuator\_ls\_state[i].driver\_temp,

actuator.actuator\_ls\_state[i].motor\_temp);

}

std::cout << "actuator state age (ms): "

<< std::chrono::duration\_cast<std::chrono::milliseconds>(

AgxMsgRefClock::now() - actuator.time\_stamp)

.count()

<< std::endl;

}

std::cout << "-------------------------------" << std::endl;

//挂起20000微秒

usleep(20000);

++count;

}

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

{

std::string device\_name;//例如 can0

std::string robot\_subtype;//例如 mini

if (argc == 2) {

device\_name = { argv[1] };

std::cout << "Selected interface " << device\_name << ", robot type: scout"

<< std::endl;

}

else if (argc == 3) {

robot\_subtype = { argv[1] };

device\_name = { argv[2] };

std::cout << "Selected interface " << device\_name

<< ", robot type: " << robot\_subtype << std::endl;

}

else {

std::cout << "Usage: demo\_scout\_robot [<robot-subtype>] <interface>"

<< std::endl

<< "Example 1: ./demo\_scout\_robot can0" << std::endl

<< "\t <robot-subtype>: mini" << std::endl;

return -1;

}

//看来robot\_subtype必须为mini

bool is\_scout\_mini = false;

if (robot\_subtype == "mini") {

is\_scout\_mini = true;

}

else if (!robot\_subtype.empty() && robot\_subtype != "scout") {

std::cout

<< "Unkonwn robot subtype. Supported subtypes: \"scout\" or \"mini\""

<< std::endl;

}

//unique\_ptr是一种定义在<memory>中的智能指针(smart pointer)。它持有对对象的独有权——两个unique\_ptr不能指向一个对象，不能进行复制操作只能进行移动操作。

ProtocolDectctor detector;

if (detector.Connect(device\_name)) {

//auto可以在声明变量的时候根据变量初始值的类型自动为此变量选择匹配的类型。

auto proto = detector.DetectProtocolVersion(5);

//如果为AGX\_V1协议

if (proto == ProtocolVersion::AGX\_V1) {

std::cout << "Detected protocol: AGX\_V1" << std::endl;

scout = std::unique\_ptr<ScoutRobot>(

new ScoutRobot(ProtocolVersion::AGX\_V1, is\_scout\_mini));

//如果为AGX\_V2协议

}

else if (proto == ProtocolVersion::AGX\_V2) {

std::cout << "Detected protocol: AGX\_V2" << std::endl;

scout = std::unique\_ptr<ScoutRobot>(

new ScoutRobot(ProtocolVersion::AGX\_V2, is\_scout\_mini));

}

else {

std::cout << "Detected protocol: UNKONWN" << std::endl;

return -1;

}

}

else {

return -1;

}

if (scout == nullptr)

std::cout << "Failed to create robot object" << std::endl;

//连接can0

scout->Connect(device\_name);

if (scout->GetParserProtocolVersion() == ProtocolVersion::AGX\_V2) {

scout->EnableCommandedMode();

}

}

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

{

initCar(argc, argv);

ros::init(argc, argv, "carwork3sub"); // 初始化节点名称

ros::NodeHandle nh; // 声明用于ROS系统和通信的节点句柄

ros::Subscriber ros\_tutorial\_sub = nh.subscribe("/ros\_tutorial\_msg", 100, msgCallback);

ros::spin();

return 0;

}

**四、程序效果展示**

图形用户界面, 文本

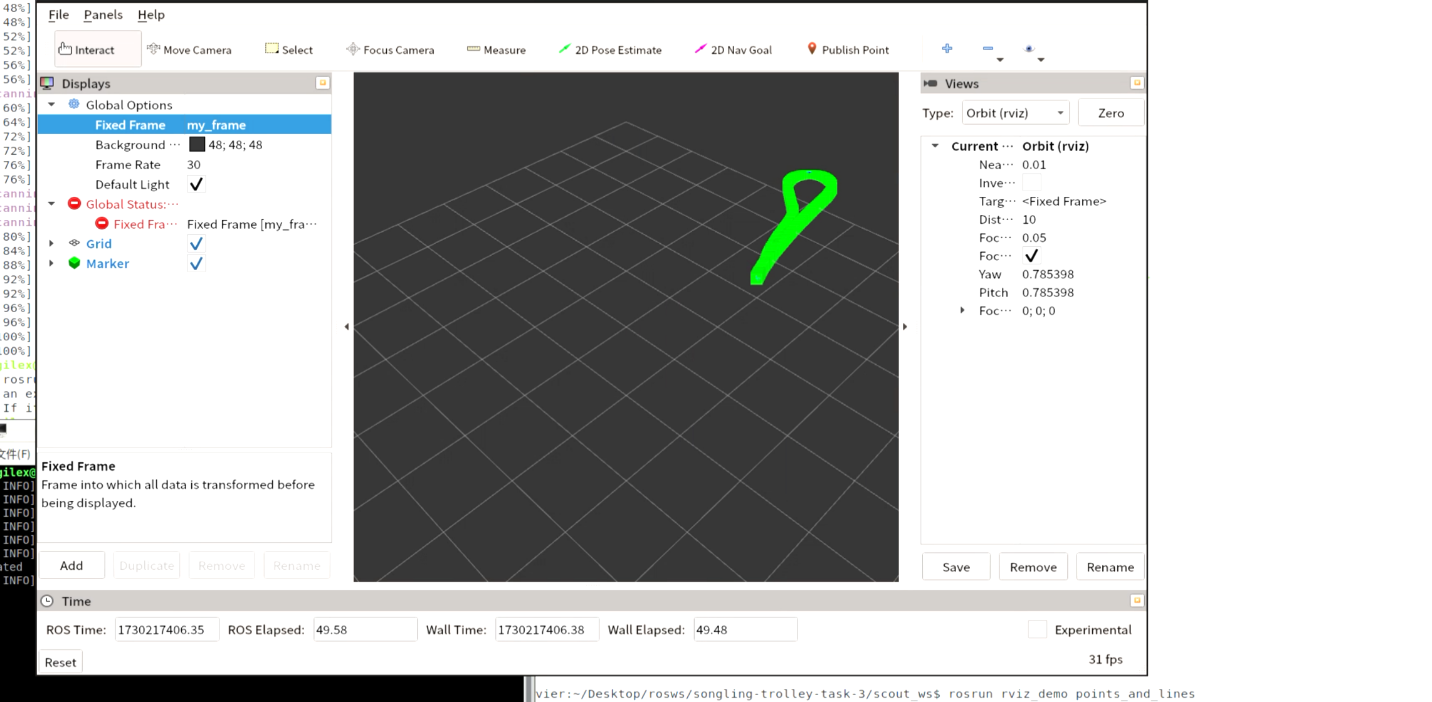
描述已自动生成

封装的ROS程序来控制小车并展示小车状态：

图形用户界面, 文本

描述已自动生成

Rviz中展示小车运行轨迹：



**五、总结和分析**

此次使用基本达到了预期的效果，虽然难度不大，但也遇到了不少问题。例如，松灵小车官方提供的 ugv\_sdk 支持通过 cmake 和 catkin\_make 两种方式进行编译。使用 cmake 编译可以生成 demo 目录下的可执行文件，可以直接运行。然而，使用 catkin\_make 方式却无法生成这些可执行文件。起初怀疑是编译环境的问题，排查耗费了不少时间，最终通过仔细阅读 CMakeLists.txt 发现：catkin\_make 编译时会忽略 demo 目录，因此不会生成相应的可执行文件。

当通过 ugv\_sdk 控制小车运动时，需要启动 CAN 使能和波特率设置两条命令。无论何时通过程序而非遥控器来操作小车，都需要启动这两条命令。此外，在 rviz 可视化程序中，由于需要订阅 /odom 话题，因此还需要启动底盘基础节点和激光雷达节点，否则 /odom 话题中不会有数据。

在 topic\_publisher.cpp 程序中，需要持续监听用户的键盘输入并将消息发送到对应的话题。这要求使用两个 while 循环：一个用于监听用户输入，另一个用于持续发送消息。如果这两个 while 循环都放在 main 函数中，会导致阻塞。这时可以通过引入多线程来解决：一个线程专门监听用户的键盘输入，另一个线程负责持续发送消息，从而使得两个 while 循环可以并发执行而不相互影响。

**六、参考文献**

[1] ROS的三种基本通信机制. http://t.csdn.cn/5KJ2z,2022-05-29.

[2] 松灵机器人Scout mini 使用Ros例程控制.zhuanlan.zhihu.com/p/550920775,2022-09-17.

[3] ROS订阅odom里程计消息，并在RVIZ中显示轨迹. http://t.csdn.cn/psy6w,2021-07-15.

[4] [美]Stephen Prata.C++ Primer plus[M].张海龙 袁国忠.北京:人民邮电出版社,2015:96-183.

[5] ROS学习（四）：消息格式. http://t.csdn.cn/hzVyi,2019-03-04.

**七、参考视频**

<https://www.bilibili.com/video/BV1dvSVYcEGV/?share_source=copy_web&vd_source=09354cc6a7a4648b50bc2816b3712cae>

源码：

