

ROS小车项目报告

- ROS小车项目报告
 - 成果展示
 - 项目目标
 - 项目设备及软件介绍
 - ROS1和ROS2介绍
 - 硬件介绍
 - 部署工具Docker(沙盒，小型虚拟机)介绍
 - 完成项目过程
 - 1.容器搭建
 - 2.任务一：创建工作空间
 - 3.任务二：基于topic通讯
 - 4.任务三：基于service通讯
 - 5.任务四、五、六：避障、循线、里程计

成果展示

1. 博客地址 [点击查看](#)
2. B站链接 [点击查看](#)

项目目标

1. 学会使用ROS。
2. 完成项目指导书前6项任务中至少4个。

项目设备及软件介绍

首先介绍本次项目所使用到的工具和设备。

ROS1和ROS2介绍

ROS(Robot Operating System)相关介绍wiki：
[ROS相关wiki 点击查看](#)

- ROS1:
[ROS1官网\(en\) 点击查看](#)
[ROS1官网\(cn\) 点击查看](#)
- ROS2:
[ROS2文档\(en\) 点击查看](#)
[ROS2文档\(cn\) 点击查看](#)

版本介绍：

[ROS1版本 点击查看](#) [ROS2版本 点击查看](#)

对比一下可以看出来，ROS1 只有Noetic支持到2025年，ROS2 Humble支持到2027年。
从维护上来说，使用ROS2肯定是更好的。

[更详细的ROS1和ROS2对比 点击查看](#)

优先使用ROS2 Humble

硬件介绍

本项目使用的小车，硬件有：

除去电机车轮，显示屏等基础设施

1. 激光雷达：思岚科技 [RPLIDAR A2 点击查看官网](#)
2. 相机：乐视三合一体感摄像头(没找到官网，博客仅供参考)
3. 运算平台：[jetson tx2 点击查看官网](#)

总的来说，硬件设施还是相对不错的。

控制相关的设备，是STM32作为下位机通过串口和上位机tx2通讯。

部署工具Docker(沙盒，小型虚拟机)介绍

同样地，相关wiki：

[Docker相关wiki 点击查看](#)

为什么要介绍Docker呢？因为我们发现，运算平台tx2上的系统是ubuntu18...，
ROS2 Humble 不支持这个版本，
而重装系统显然又不太靠谱。所以最好的办法是，使用**容器部署**。

容器部署会遇到很多正常情况下遇不到的事，后面会详细介绍

[Docker官网 点击查看](#)

然后就是安装docker

```
# ubuntu 安装docker
sudo apt update
sudo apt install docker.io docker docker-compose
# 更新docker用户组权限，防止使用docker需要sudo
sudo gpasswd -a "$USER" docker
sudo newgrp docker
# 启动docker并设置开机服务自启
sudo systemctl start docker
sudo systemctl enable docker
```

完成项目过程

接下来具体介绍，如何完成本次项目。

1.容器搭建

ps:第一次使用的小车拓展坞坏了，因为连不了鼠标键盘，系统都进不去。后面把车换了

先给小车连上校园网。（或者手机热点，保证电脑可以和小车运算平台互ping）

使用ssh连接小车。首先安装docker，发现车上早就装了。

然后从[Docker Hub](#)上拉取镜像。

最开始是直接拉取非官方[ROS2](#)镜像

一开始拉了半天镜像都拉不下来，一查发现是系统空间不够。

```
# 查询磁盘空间，发现最大的一块空间是挂载上去的磁盘
# 而docker文件默认存放的位置在/var/lib/docker下
df -h
# 编辑daemon文件，修改docker文件存放位置
# 修改/添加"data-root":"<docker文件存放的文件夹>"
# 这里使用vim修改，使用其他编辑器也行
vim /etc/docker/daemon.json
# 修改后重启docker服务
sudo service docker restart
# 查看修改后的docker文件存放位置是否正确
docker info | grep Dir
```

修改后拉取镜像

```
# 这个镜像的架构是amd64的
docker pull osrf/ros:humble-desktop
```

然后在漫长的镜像拉取等待之后发现镜像run不起来，tx2貌似是aarch架构（本质上是arm64架构）

拉的这个镜像不能用。查了一下Docker Hub的其他ROS镜像，貌似都不是arm架构。

只能老老实实写DockerFile或者拉取ubuntu22的镜像手动安装ros2之后导出了。

一顿操作安装ROS2

```
# 小鱼安装ros脚本，使用bash执行不要用zsh
wget http://fishros.com/install -O fishros && . fishros
```

导出成镜像然后运行。

```
# 生成容器
docker run --privileged \
  --network=bridge -p 8888:22 -p 8765:8765 \
  --cap-add=SYS_PTRACE --security-opt seccomp=unconfined \
  -v "$HOME":"$HOME" \
  -e DISPLAY="$DISPLAY" -v /tmp/.X11-unix:/tmp/.X11-unix \
  --name=<生成容器的名字> -it <镜像名>:<版本> sh
```

解释一下生成容器的一些参数解析：

- `-privileged` 是给容器最高权限的，这样就可以在容器里访问宿主机的硬件了。

一般为了隔离，是用`-device`挂载需要的硬件就好了，这里咱也不知道之后要用几个硬件，就偷懒了。

- `-network=bridge -p 8888:22 -p 8765:8765`

使用桥接模式连接宿主机，并把外部的8888端口转发到内部的22端口，8765同理

也可以使用`-network=host`这样的话就和宿主机共享所有端口 [相关博客](#)

- `-cap-add=SYS_PTRACE --security-opt seccomp=unconfined`
gdb调试用的，容器搭建完毕之后可以不用

- `-v "$HOME":"$HOME"`

把宿主机的"*HOME*"挂载到容器内的"*HOME*"目录，比如小车外面叫nvidia，那就会把外面的/home/nvidia挂载到里面的/home/nvidia目录

- `-e DISPLAY="$DISPLAY" -v /tmp/.X11-unix:/tmp/.X11-unix`
x11转发用的，**不使用问题也不是很大**，如果遇到转发不了，需要在x11转发目标机器上执行 `xhost +`

2.任务一：创建工作空间

1. 先连接小车容器

```
# ssh连接小车
ssh -X -p 22 nvidia@<小车ip>
# 先启动并进入容器
docker start <容器名> && docker exec -it <容器名> zsh
# 容器内开启ssh服务
service ssh start
# ps:开启服务后，可在自己电脑上直接连接内部容器
ssh -X -p 8888 root@<小车ip>
```

2. 使用ros2 cli命令创建工作空间

```
# 创建工作空间和src文件夹
mkdir -p /root/workspace/src
# 到工作空间
cd /root/workspace
# 构建
colcon build
# 如果找不到colcon命令
# 环境变量更新
echo 'source /opt/ros/humble/setup.zsh' >> ~/.zshrc
source ~/.zshrc
# 再构建
colcon build
```

任务一就完成了。

3.任务二：基于topic通讯

先看看ROS2自带的例子

```
# cpp例子
# 发布者节点
ros2 run examples_rclcpp_minimal_publisher publisher_member_function
# 订阅者节点
ros2 run examples_rclcpp_minimal_subscriber subscriber_member_function
# python例子
# 发布者节点
ros2 run examples_rclpy_minimal_publisher publisher_member_function
# 订阅者节点
ros2 run examples_rclpy_minimal_subscriber subscriber_member_function
```

在workspace/src下创建包然后编写代码即可

发布者大致代码见本地文件

(./code/workspace/src/mypkg/pubsub/include/pubsub/pub.hpp) 订阅者类似

```

class Pub : public rclcpp::Node {
public:
    explicit Pub(const std::string& name) : Node(name) {
        RCLCPP_INFO(this->get_logger(), "%s开始运行", name.c_str());

        pub_ptr_ = this->create_publisher<std_msgs::msg::String>("/topic1");

        auto callback_f = std::bind(&Pub::sayHelloCallBack, this);
        timer_ptr_ = this->create_wall_timer(500ms, callback_f);
    }

    ~Pub() = default;

private:
    void sayHelloCallBack() {
        std_msgs::msg::String msg;
        msg.data = "hello " + std::to_string(++count_);
        if (count_ == 10) {
            msg.data = "kill";
            pub_ptr_->publish(msg);
            RCLCPP_INFO(this->get_logger(), "发布消息:'%s' ", msg.data.c_str());
            exit(0);
        }
        pub_ptr_->publish(msg);
        RCLCPP_INFO(this->get_logger(), "发布消息:'%s' ", msg.data.c_str());
        return;
    }

    rclcpp::Publisher<std_msgs::msg::String>::SharedPtr pub_ptr_;
    rclcpp::TimerBase::SharedPtr timer_ptr_;
    size_t count_ = 0;
};

```

代码架构大致如下

文件树如下:

```

. src
├─ mypkg                                #存放自己编写的所有包
│   └─ pubsub                          #topic通讯的包
│       ├── CMakeLists.txt            #包对应的CMakeLists.txt
│       ├── include                   #头文件
│       │   └─ pubsub
│       │       ├── pub.hpp
│       │       └─ sub.hpp
│       ├── launch                    #launch文件
│       │   └─ pubsub_run.launch.py
│       ├── package.xml
│       └─ src                        #源文件
│           └─ pub.cpp

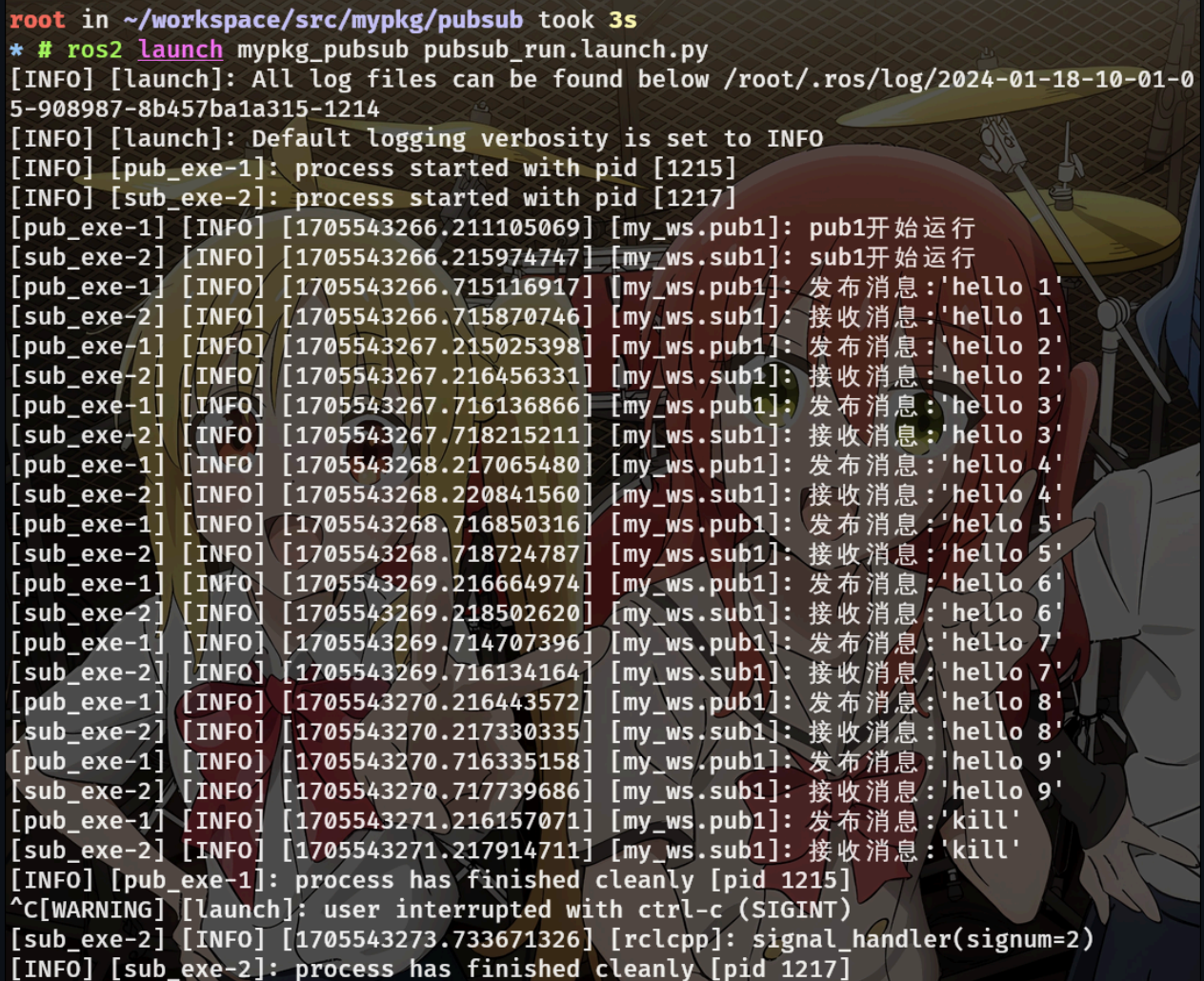
```



```
-- sub.cpp

# 构建
cd ~/workspace && colcon build --symlink-install
# 更新环境变量
source ~/workspace/install/setup.zsh
# 运行
ros2 launch mypkg_pubsub pubsub_run.launch.py
```

运行效果：



```
root in ~/workspace/src/mypkg/pubsub took 3s
* # ros2 launch mypkg_pubsub pubsub_run.launch.py
[INFO] [launch]: All log files can be found below /root/.ros/log/2024-01-18-10-01-05-908987-8b457ba1a315-1214
[INFO] [launch]: Default logging verbosity is set to INFO
[INFO] [pub_exe-1]: process started with pid [1215]
[INFO] [sub_exe-2]: process started with pid [1217]
[pub_exe-1] [INFO] [1705543266.211105069] [my_ws.pub1]: pub1开始运行
[sub_exe-2] [INFO] [1705543266.215974747] [my_ws.sub1]: sub1开始运行
[pub_exe-1] [INFO] [1705543266.715116917] [my_ws.pub1]: 发布消息:'hello 1'
[sub_exe-2] [INFO] [1705543266.715870746] [my_ws.sub1]: 接收消息:'hello 1'
[pub_exe-1] [INFO] [1705543267.215025398] [my_ws.pub1]: 发布消息:'hello 2'
[sub_exe-2] [INFO] [1705543267.216456331] [my_ws.sub1]: 接收消息:'hello 2'
[pub_exe-1] [INFO] [1705543267.716136866] [my_ws.pub1]: 发布消息:'hello 3'
[sub_exe-2] [INFO] [1705543267.718215211] [my_ws.sub1]: 接收消息:'hello 3'
[pub_exe-1] [INFO] [1705543268.217065480] [my_ws.pub1]: 发布消息:'hello 4'
[sub_exe-2] [INFO] [1705543268.220841560] [my_ws.sub1]: 接收消息:'hello 4'
[pub_exe-1] [INFO] [1705543268.716850316] [my_ws.pub1]: 发布消息:'hello 5'
[sub_exe-2] [INFO] [1705543268.718724787] [my_ws.sub1]: 接收消息:'hello 5'
[pub_exe-1] [INFO] [1705543269.216664974] [my_ws.pub1]: 发布消息:'hello 6'
[sub_exe-2] [INFO] [1705543269.218502620] [my_ws.sub1]: 接收消息:'hello 6'
[pub_exe-1] [INFO] [1705543269.714707396] [my_ws.pub1]: 发布消息:'hello 7'
[sub_exe-2] [INFO] [1705543269.716134164] [my_ws.sub1]: 接收消息:'hello 7'
[pub_exe-1] [INFO] [1705543270.216443572] [my_ws.pub1]: 发布消息:'hello 8'
[sub_exe-2] [INFO] [1705543270.217330335] [my_ws.sub1]: 接收消息:'hello 8'
[pub_exe-1] [INFO] [1705543270.716335158] [my_ws.pub1]: 发布消息:'hello 9'
[sub_exe-2] [INFO] [1705543270.717739686] [my_ws.sub1]: 接收消息:'hello 9'
[pub_exe-1] [INFO] [1705543271.216157071] [my_ws.pub1]: 发布消息:'kill'
[sub_exe-2] [INFO] [1705543271.217914711] [my_ws.sub1]: 接收消息:'kill'
[INFO] [pub_exe-1]: process has finished cleanly [pid 1215]
^C[WARNING] [launch]: user interrupted with ctrl-c (SIGINT)
[sub_exe-2] [INFO] [1705543273.733671326] [rclcpp]: signal_handler(signum=2)
[INFO] [sub_exe-2]: process has finished cleanly [pid 1217]
```

任务二完成。

4.任务三：基于service通讯

类似任务二：ROS2 自带例子：

```
# 仅展示cpp例子
# 服务端
ros2 run examples_rclcpp_minimal_service service_main
# 客户端
ros2 run examples_rclcpp_minimal_client client_main
```

服务端代码来自本地文件(./code/workspace/src/mypkg/pubsub/include/pubsub/sub.hpp)

```
// ros2
#include <rclcpp/rclcpp.hpp>
// std
#include <memory> //NOLINT
// interfaces
#include <example_interfaces/srv/add_two_ints.hpp>

int main(int argc, char **argv) {
    rclcpp::init(argc, argv);
    auto node = rclcpp::Node::make_shared("add_two_ints_service");
    auto add =
        [&node](
            const std::shared_ptr<example_interfaces::srv::AddTwoInts::Request>
                request,
            std::shared_ptr<example_interfaces::srv::AddTwoInts::Response>
                response) {
                response->sum = request->a + request->b;
                RCLCPP_INFO(node->get_logger(), "正在获取数据请求\na: %ld b: %ld",
                    request->a, request->b);
                RCLCPP_INFO(node->get_logger(), "返回响应: [%ld]", response->sum);
            };
    auto service = node->create_service<example_interfaces::srv::AddTwoInts>(
        "add_two_ints", add);
    RCLCPP_INFO(node->get_logger(), "服务端初始化完毕，可以开始相加");
    rclcpp::spin(node);
    rclcpp::shutdown();
} // NOLINT
```

类似地，执行代码

```
# 构建
cd ~/workspace && colcon build --symlink-install
# 更新环境变量
source ~/workspace/install/setup.zsh
# 后台运行服务端
ros2 run mypkg_service server &
# 运行客户端
ros2 run mypkg_service client 114000 514
```


效果如下

```
root in ~
* # ros2 run mypkg_service server 8
[1] 1723

root in ~
* # [INFO] [1705545390.884265816] [add_two_ints_service]: 服务端初始化完毕，可以开始相加
ros2 run mypkg_service client 114000 514
[INFO] [1705545407.489990016] [add_two_ints_service]: 正在获取数据请求
a: 114000 b: 514
[INFO] [1705545407.490117460] [add_two_ints_service]: 返回响应: [114514]
[INFO] [1705545407.490882701] [add_two_ints_client]: 服务返回结果: 114514

root in ~
* #
```

任务三完成。

5.任务四、五、六：避障、循线、里程计

前面的任务都是没有实物的任务，后面三个任务都是实现具体的某项功能。

那么**首要任务**就是实现通过容器内的ros2与外部硬件交互。

首先，雷达是有ros2相应的包的[雷达ros2 github 点击查看](#)。

这让我们的工作量得以减轻，但是控制小车底盘的代码，由厂商提供，是没有直接的ROS2代码的。这实在是令人悲伤，我们不得已去修改上位机通讯的代码。

详细代码见本地文件(./code/workspace/src/jubot/jubot_driver/src/jubot_driver.cpp)

我们将原本使用ROS1编写的底盘控制节点改写为ROS2的写法，这个过程倒不是很困难，但是由于原本的代码使用了一些奇怪的写法，比如使用了boost的线程库而非c++标准库，导致修改的过程有一点曲折，不过好在时间比较充裕，代码得以修改完成。

然后是

- 避障代码和底盘控制的通讯，详见本地文件夹(./code/workspace/src/mypkg/obstacle_avoidance/) 避障策略为，判断前方一定角度内距离最近的障碍是否大于给定值，大于则前进，小于则同时求两边障碍距离平均值，判断向哪个方向转动。
- 循线代码和底盘控制的通讯，详见本地文件夹(./code/workspace/src/mypkg/line_follow/) 循线使用opencv提取对应HSV或RGB颜色，进行形态学操作后计算其距离摄像头中心的距离，识别到线后判断是向左/右前方前进（左右偏移的速度由检测到的线中心距离中心的偏移程度决定），还是直行，未识别到则自转。
- 里程计代码，详见(./code/workspace/src/jubot/jubot_driver/)

我们选择远程调试的工具为foxglove-studio 详见[github](#)
最后结果见视频

任务四五六完成。