

# 结项报告

RT&ROS智能小车

do\_our\_best

LOGO

# 目录

Here is your Content

>

<

1

项目分工

2

选题目的

3

设计实现

4

成果展示

5

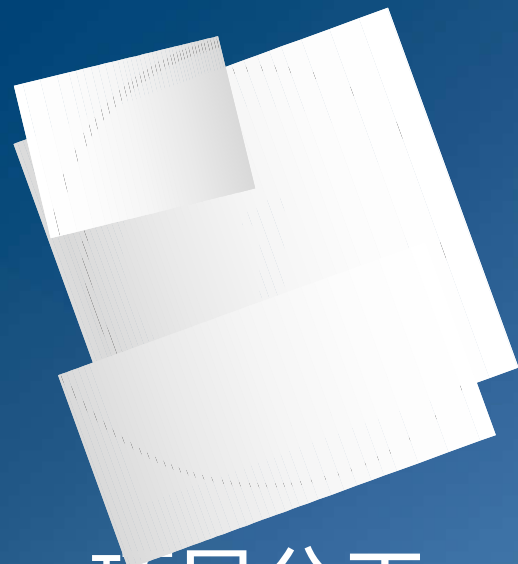
应用场景

6

失败尝试

7

总结与展望



# 项目分工



黄与进：ros的资料查找、安装、学习，ros的环境配置以及控制和操作，代码编写。

刘津畅：RT-thread的资料查找、学习，调试小车硬件，协助小车代码与硬件之间的适配。

陆子睦：RT-thread的资料查找、学习，RT-thread在STM32上的移植，RT-thread与ros的结合，代码编写，报告及ppt制作。

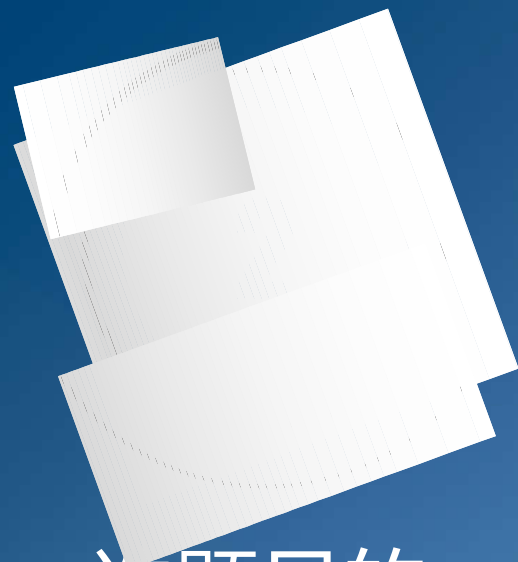
唐星：ros的资料查找、安装、学习，ros的环境配置以及操作，代码编写，报告及ppt制作。

杨涛：rtthread的资料查找、学习，RT-thread Studio使用的研究，RT-thread和ros代码的编写，se14上调用Linux虚拟机方案的尝试

周四周日准时碰头



电四楼221



# 选题目的





## 为什么选择RT-Thread+ROS+智能小车？

随着智能硬件、物联网行业的迅猛发展, 嵌入式系统在各个领域都得到了广泛的应用。嵌入式操作系统可以帮助嵌入式设备更好地完成任务的调度, 从而更高效地完成任务。目前, 嵌入式系统正在向着功能日趋复杂, 多机联合, 分布式等方面发展。这些复杂的应用需要有较强的运算量, 而传统的单片机是无法完成这些复杂的计算的。同时, 要想实现复杂的功能, 裸机编程也会因为过于困难且容易出错而难以实现。

# RT-Thread

一个面向物联网设备的嵌入式实时多线程操作系统



跨芯片平台



实时操作系统内核



快速启动



开放平台





RT-Thread 主要采用 C 语言编写，浅显易懂，方便移植、架构清晰、系统模块化并且可裁剪性非常好。对于资源丰富的物联网设备，RT-Thread 能使用在线的软件包管理工具，配合系统配置工具实现直观快速的模块化裁剪，无缝地导入丰富的软件功能包。

相较于 Linux 操作系统，RT-Thread 体积小，成本低，功耗低、启动快速，除此以外 RT-Thread 还具有实时性高、占用资源小等特点，非常适用于各种资源受限（如成本、功耗限制等）的场合。

### 跨芯片平台

支持所有主流微控制器，  
解决设备碎片化问题。

01

02

### 实时操作系统内核

完全由中国团队自主开发，硬实时，精致，高效，高度可定制。

### 快速启动

上电即启动，毫秒级启动时间，  
真正零等待开机。

03

04

### 开放平台

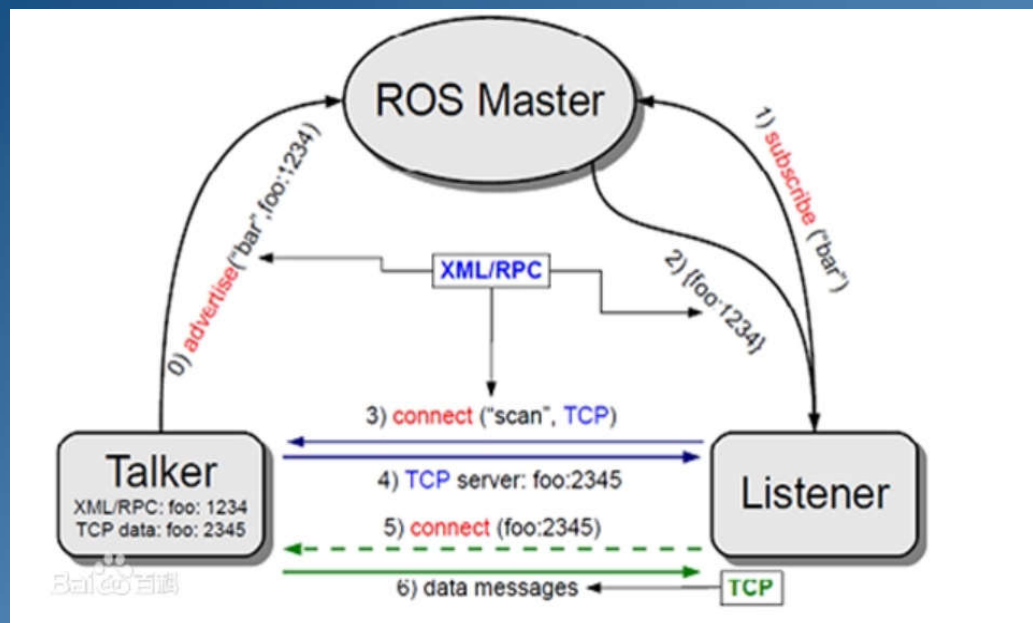
独立第三方开放平台，支持各类，  
第三方软件包和SDK，扩展系统  
功能。

# ROS

用于编写机器人软件程序的一种具有高度灵活性的软件架构

ROS是用于编写机器人软件程序的一种具有高度灵活性的软件架构。

ROS的核心是一个可以同步或者异步传递消息的中间件框架，即使是不同机器人的进程和线程也可以通过它进行交流和传输数据。



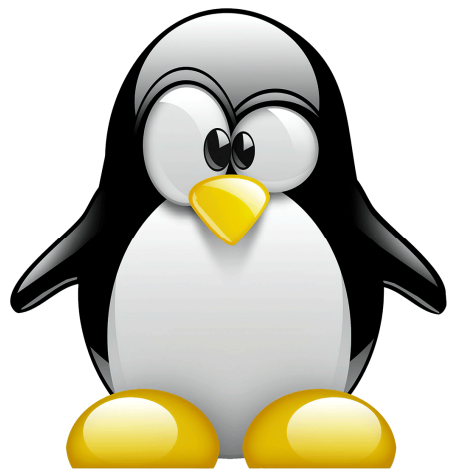
# RT-Thread + ROS

---

虽然名叫机器人操作系统，但ROS其实并不是一个操作系统，而只是一些软件包，它的实时性完全是由它运行在其上的操作系统实现的，而传统上ros一般运行在Linux系统上或是运行在裸机上，然而Linux系统并不是专门的实时性操作系统，它的实时性并不是很好，而裸机编程又过于复杂不适合实现丰富的。

所以我们决定在实时性操作系统RT-thread上运行ros，通过ros来与主节点上的ros通信，并可以把需要进行复杂的计算的信息发送给运行在PC端的主节点上的ros，在PC端进行运算，再通过运行结果来决定嵌入式系统的一些操作。

实时性：优于Linux + Ros



Linux：并非实时操作系统，无硬实时性



RT-Thread：实时操作系统，有优越的实时性

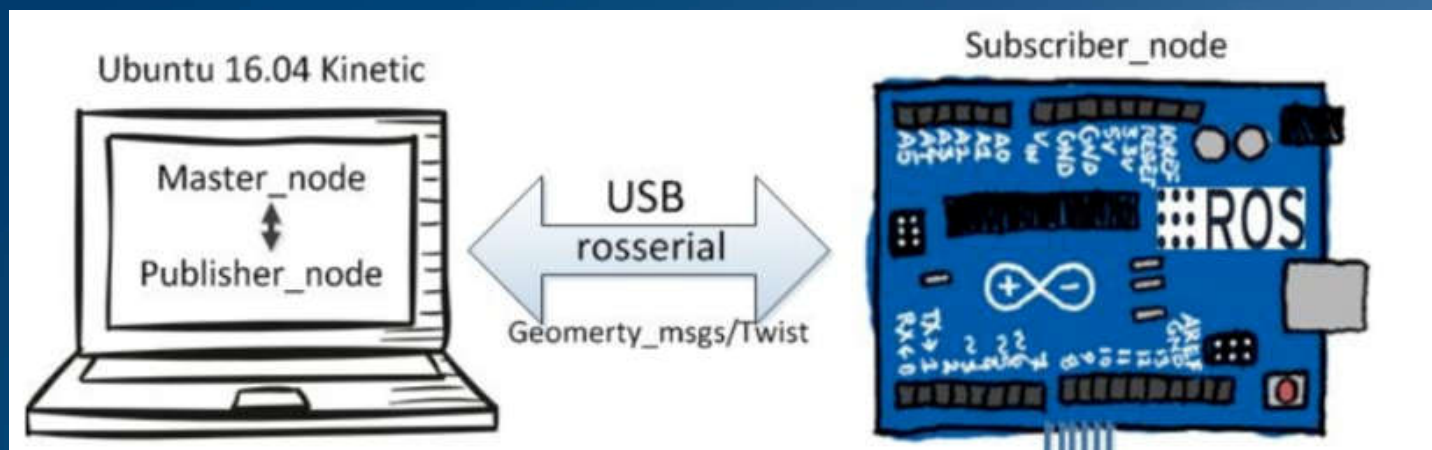
RT-Thread-Ros > Linux-Ros

可编程性：优于裸机编程

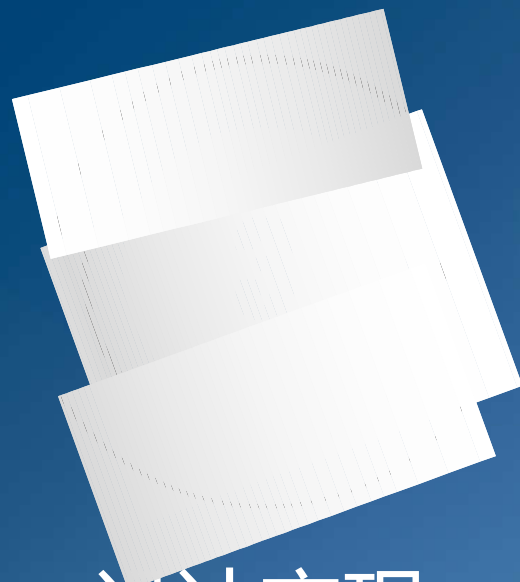


RT-Thread-ros作为操作系统，比裸机上的ROS节点更适合于完成复杂任务

计算性能：能够完成复杂计算任务



RT-Thread-Ros把数据传回电脑端Ros，在电脑端实现复杂计算任务



# 设计实现







---

## RT-Thread-Ros与电脑端 互联

---

通过串口转usb与电脑端互  
联，验证RT-Thread-Ros与  
电脑端Ros成功建立信道

---

## 单车调度

---

通过PC端键盘输入，  
控制小车移动。

---

## 双车调度

---

A车巡逻侦测，向PC端发送待侦测位置；  
PC端处理数据；  
发送控制信号，指示B车沿最短路径经  
过待侦测点。

完成环境搭建



20%

配置TCP通信模块



50%

编写控制代码



100%


## 项目阶段


1. 调研阶段：第一版方案1-4周，废弃；第二版方案，4-6周，前期走了较多弯路；
2. 确定具体路线：7-9周，不断试错、讨论、修改；
3. 环境搭建：10-11周。
4. TCP通信模块：12-13周，硬件、软件的适配过程。
5. 控制代码：13-15周。
6. 收尾工作：15-17周，整理资料，完善细节，编写报告。


# 环境搭建




zRT-Thread工程中添加roserial软件包，编译  
出RT-Thread-Ros系统

▼  roserial-noetic-latest


>  doc

>  port

>  src

 LICENSE

 README.md

 SConscript

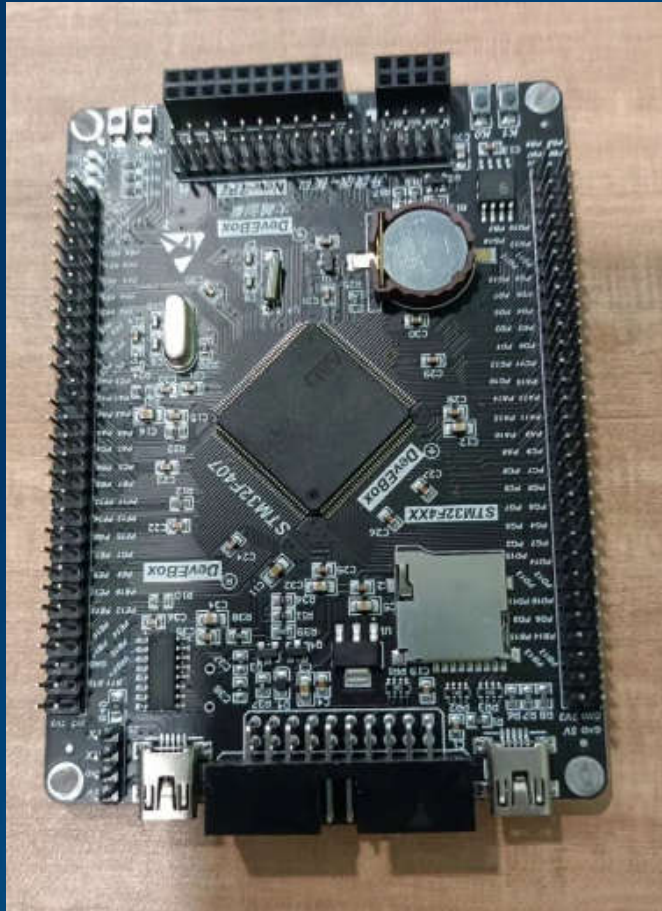


RT-Thread

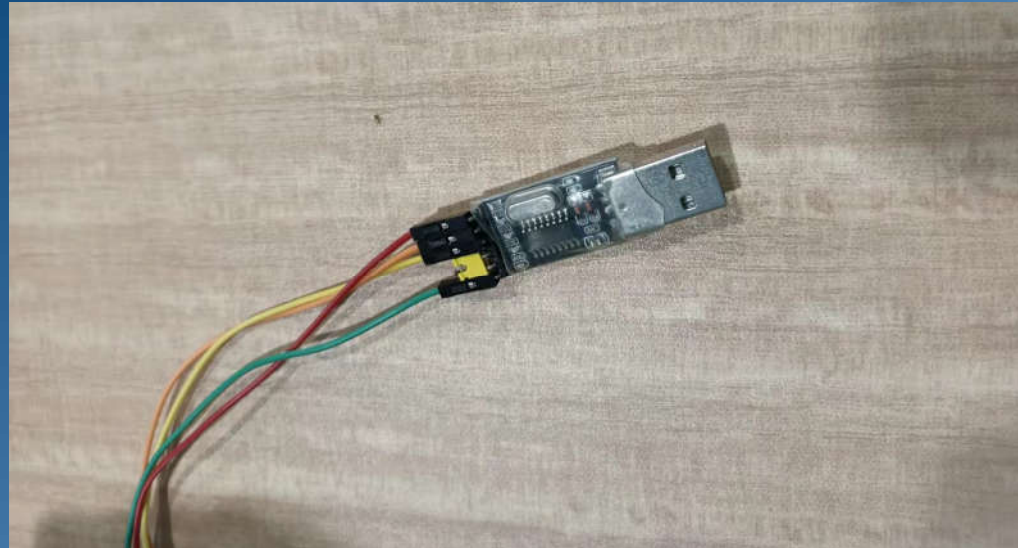
## RT-Thread打开串口

```
74  
75 #define BSP_USING_UART2  
76 #define BSP_UART2_TX_PIN    "PA2"  
77 #define BSP_UART2_RX_PIN    "PA3"  
78
```

# STM32开发板

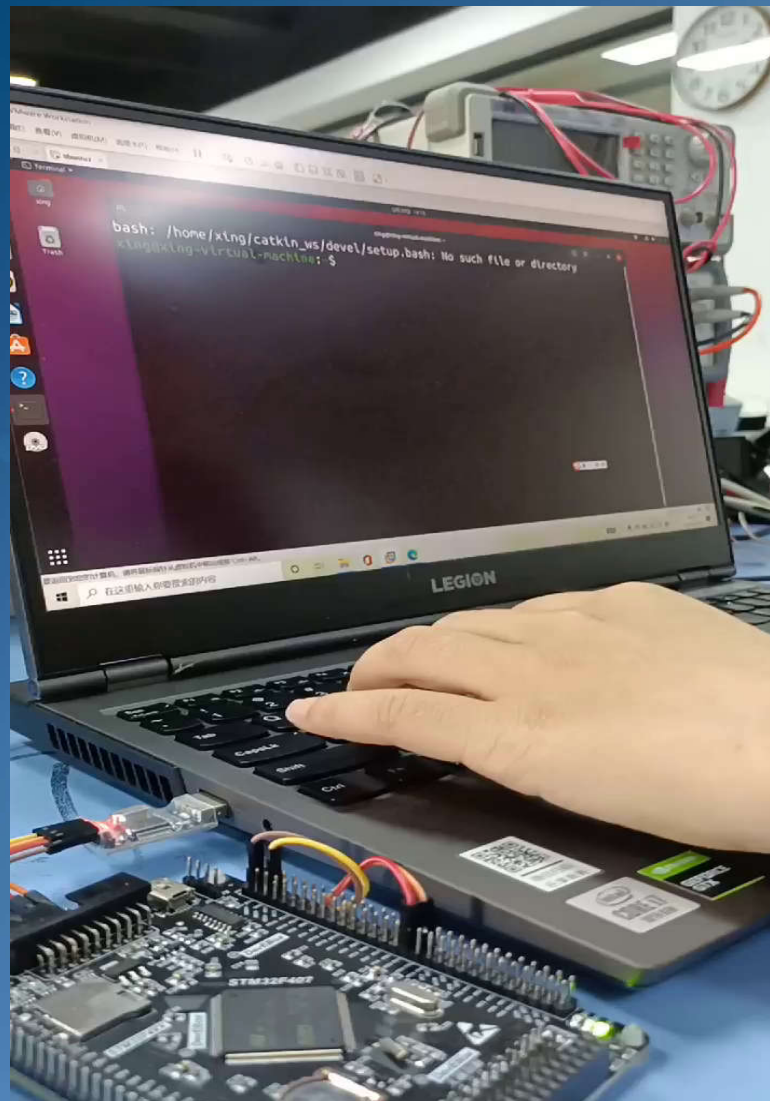


# 串口转USB模块

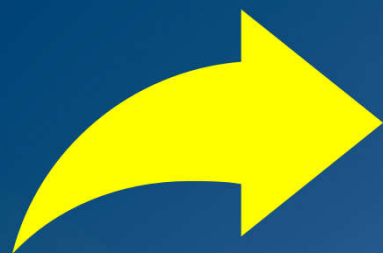


# RT-Thread-Ros与电脑端互联

通过串口转usb与电脑端互联，验证RT-Thread-Ros与电脑端Ros成功建立信道



系统搭建  
完成！



应用：  
RT-Thread-Ros小车





### 主要思路:

ROS运行在PC端，RT-thread项目运行在开发板上，二者通过rosserial软件包建立通信；  
ROS端对小车的运动通过无线通信方式进行控制，RT-Thread端实时管理小车的运动



## 系统搭建逻辑：RT-Thread--ROs + ESP8266 WIFI模块 + PC



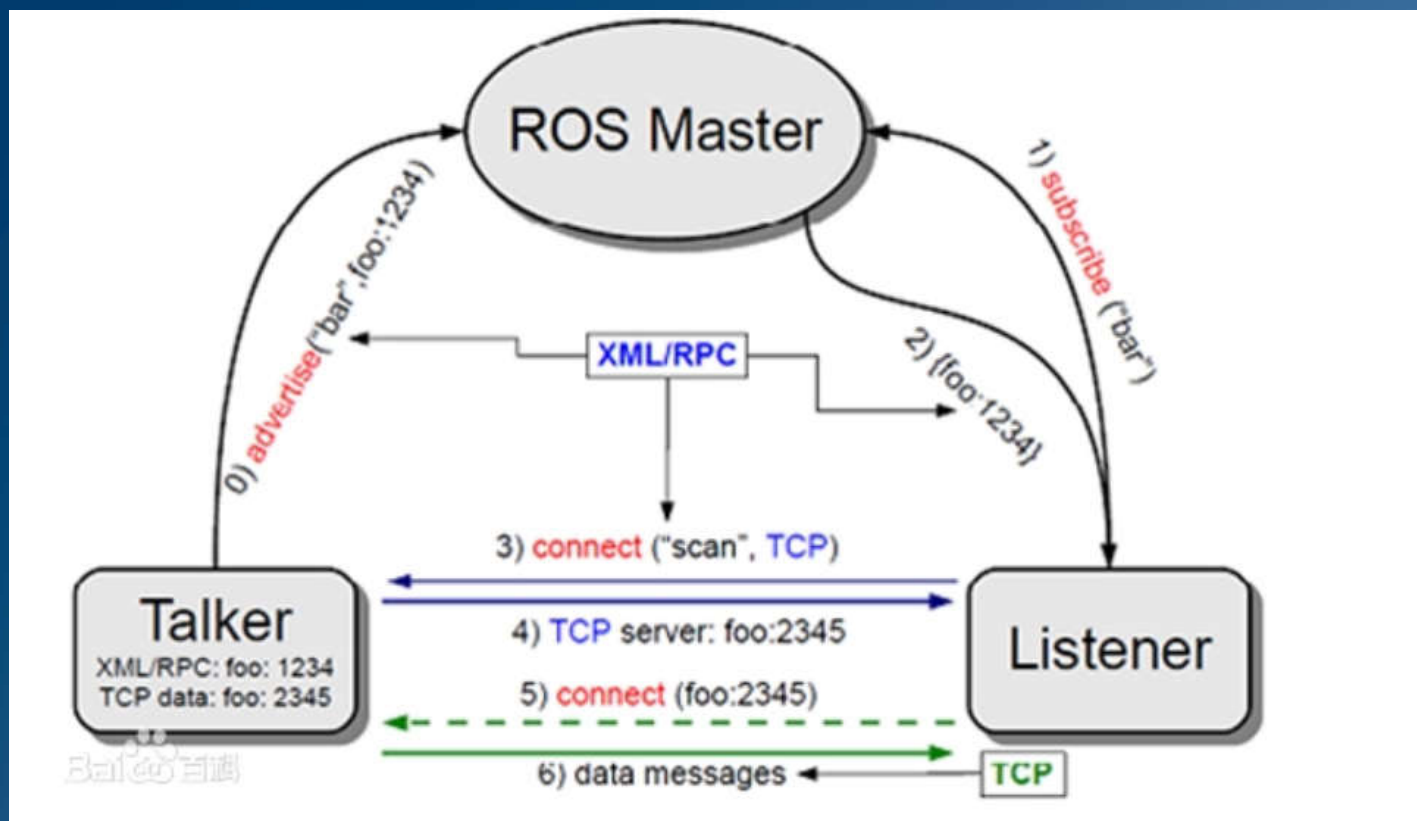
# TCP通信模块

ROS端与RT-Thread端通信实现



ROS端和RT-thread端互发消息无线通信，ros端控制小车的速度和旋转角速度

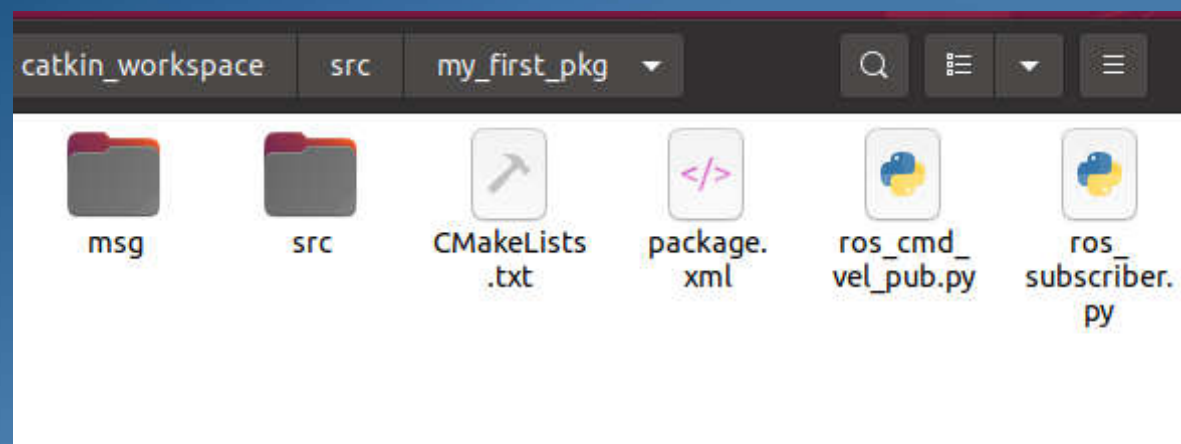
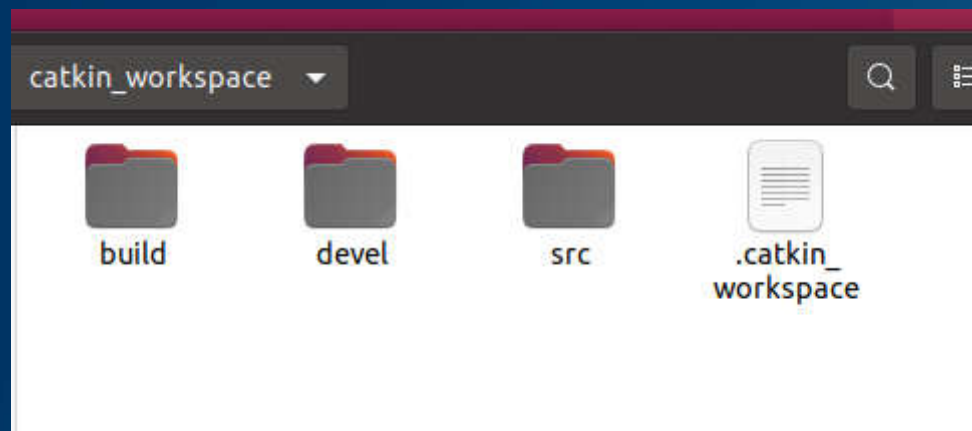
# ROS实现小车与PC通信



# 控制代码编写



# ROS软件包代码编写



# RT-Thread小车控制代码编写

名称	修改日期	类型
ros_multi_car_package	2022/6/10 21:50	文件夹
ros_single_car_package	2022/6/9 18:39	文件夹
rtthread_ros	2022/6/9 18:39	文件夹
rtthread_ros_multi_cars	2022/6/10 21:50	文件夹
rtthread_ros_multi_cars_A	2022/6/9 18:39	文件夹
rtthread_ros_multi_cars_B	2022/6/9 18:42	文件夹
rtthread_ros_single_car	2022/6/6 9:44	文件夹

```
//Publisher
ros::Publisher<std_msgs::Float64MultiArray> locations("locations", 1000);

std_msgs::Float64 velX_tmp;
std_msgs::Float64 turnBias_tmp;
std_msgs::Float64 begin_tmp;
ros::Publisher xv("vel_x", &velX_tmp);
ros::Publisher xt("turn_bias", &turnBias_tmp);

clock_t start_time, test_time;

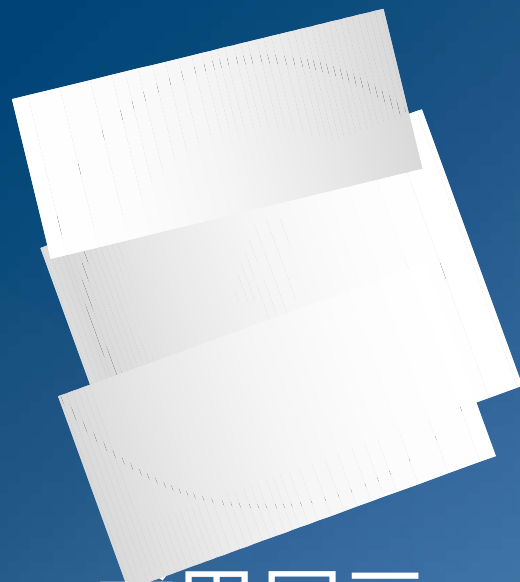
static void rosserial_thread_entry(void *parameter)
{
    //Init motors, specify the respective motor pins
    mtr.initMotors();

    // 设置 ROS 的 IP 端口号
    nh.getHardware()->setConnection("192.168.125.21", 11411);

    //Init node>
    nh.initNode();

    // 发布了一个话题 /vel_x 告诉 ROS 小车速度
    nh.advertise(xv);

    // 发布了一个话题 /turn_bias 告诉 ROS 小车的旋转角速度
    nh.advertise(xt);
```

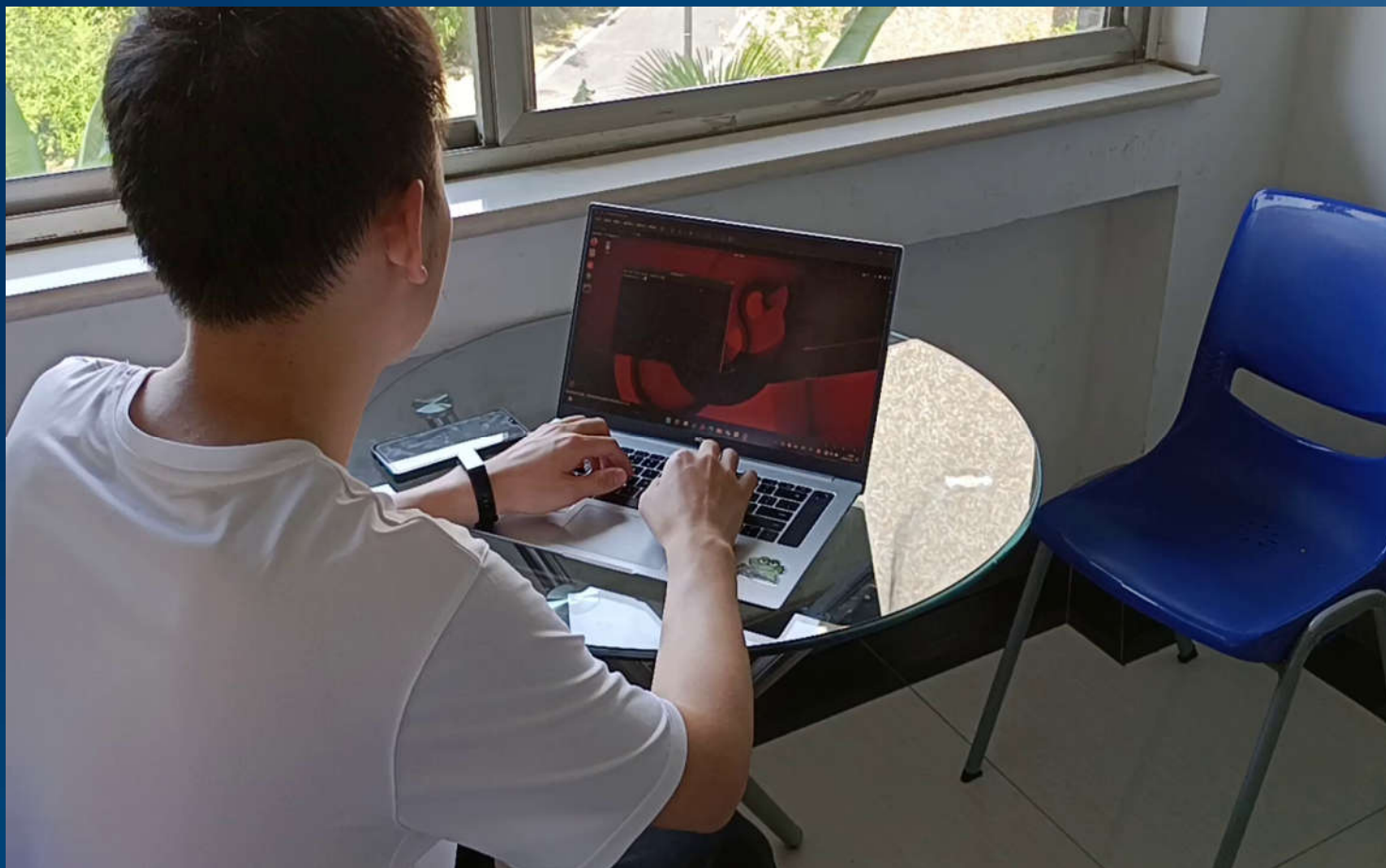


## 成果展示





# 单车调度



通过电脑端w,  
s, a, d控制  
小车前进、  
后退、左转、  
右转

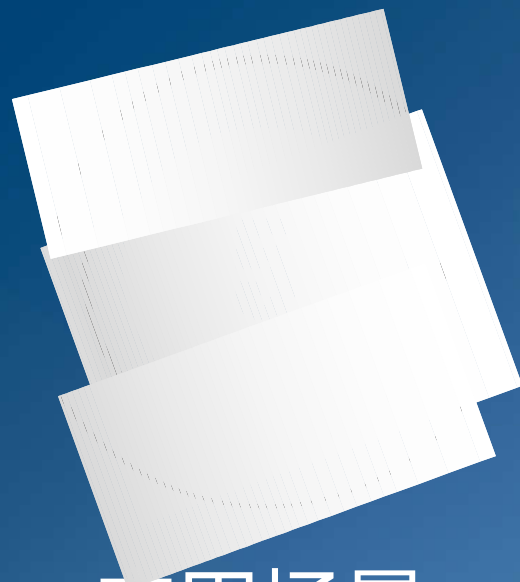
# 双车调度



A车:  
巡逻

B车:  
踩点





## 应用场景



# 智能小车应用场景

## 智能园区

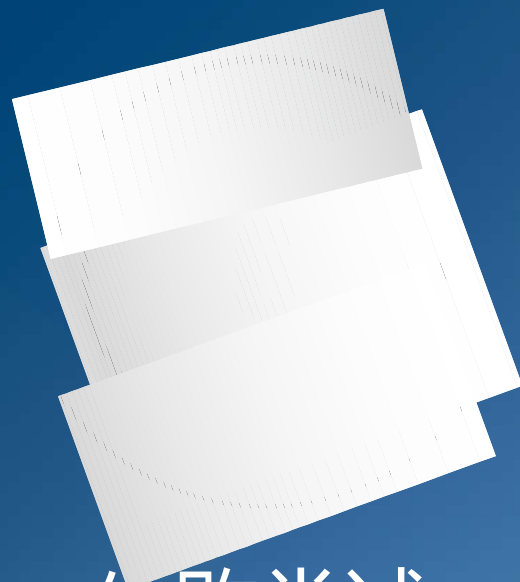
已有科技园区投入使用智能小车完成送餐工作；不少高校开始使用智能车进行快递配送。

## 病毒消杀

疫情期间，使用智能车代替人工进行消杀工作，更加高效安全

## 信息监控采集

地理环境信息采集、农作物生长监控等等。



失败尝试



# 混合内核

本小组一开始是希望能够用混合内核的方式，使用一个实时性较好的微内核调用Linux，从而实现实时性和技术复杂性的兼顾，我们也尝试了这种方案，不过由于一些瓶颈没有实现。

选择seL4微内核，其带有已经带有虚拟机组件，使用 CAMkES 来创建、配置和构建虚拟机从而运行 guest Linux

编写简单测试程序并加入内核编译链，并且搭建相关的编译环境。



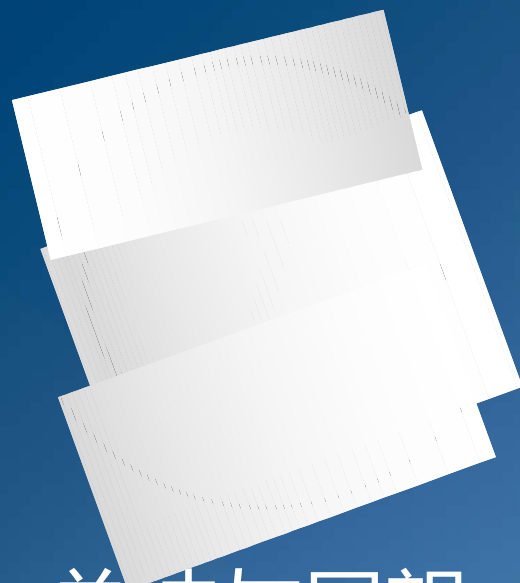
```
ubuntu@VM3274-test: /home/ubuntu/code/fry/sel4-tutorials-manifest/hello-world_build
文件(F) 编辑(E) 视图(V) 搜索(S) 终端(T) 帮助(H)
Hello, World!
Second hello
Caught cap fault in send phase at address 0
while trying to handle:
vm fault on data at address 0 with status 0x4
in thread 0xffffffff801fe08400 "rootserver" at address 0
With stack:
0x41ce98: 0x40179d
```

# 混合内核

本小组一开始是希望能够用混合内核的方式，使用一个实时性较好的微内核调用Linux，从而实现实时性和技术复杂性的兼顾，我们也尝试了这种方案，不过由于一些瓶颈没有实现。

进行VMM组件的搭建与linux的运行，在内核编译完成之后使用qemu进行模拟，但在尝试模拟的过程之中出现了如图的错误

```
$ qemu-system-x86_64 -cpu Nehalem,+vme,+pdpe1gb,-xsave,-xsaveopt,-xsavc,-  
fsgsbase,-invpcid,\  
+syscall,+lm,enforce -nographic -serial mon:stdio -m size=512M -kernel  
images/kernel-x86_64-pc99 \  
-initrd images/capdl-loader-image-x86_64-pc99  
qemu-system-x86_64: warning: TCG doesn't support requested feature:  
CPUID.01H:EDX.vme [bit 1]  
qemu-system-x86_64: TCG doesn't support requested features
```



# 总结与展望





## RT-Thread-Ros

优势:

实时操作系统

实时性

处理计算复杂型任务能力

项目不足:

小车控制比较粗糙，路径有误差

未能实现实时内核+Linux的混合内核

展望：

在这个项目基础上实现分布式车辆调度

单个主节点变为分布式集群

小车上激光、雷达、摄像头应用

.....

谢谢！