1 ROS及sp_control基本介绍

0、前提

预装VMWare且已经预装好Ubuntu20.04+Noetic+CMake+Opencv（与视觉组保持一致）

1、ROS介绍

1.1 何为ROS

ROS全称为 Robot Operating System(机器人操作系统)，顾名思义，他是应用于机器人的一套集成开发环境。对于一台机器人，我们往往从算法、嵌入式、机械三个层面来分析，这也是我们机甲大师对抗赛大都分为机械、电控、视觉三个组的原因。机器人设计包含了机械加工、机械结构设计、硬件设计、嵌入式软件设计、上层软件设计....是各种硬件与软件集成，可以说机器人系统是当今工业体系的一种集大成者。

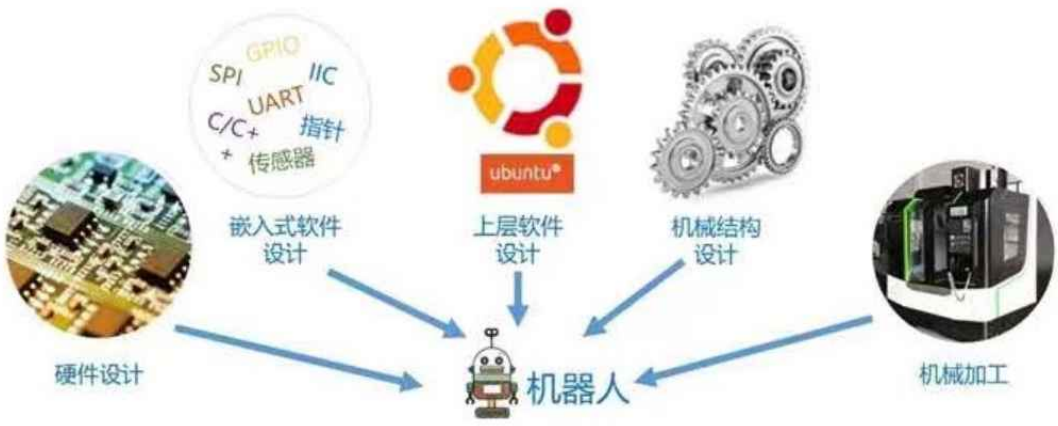


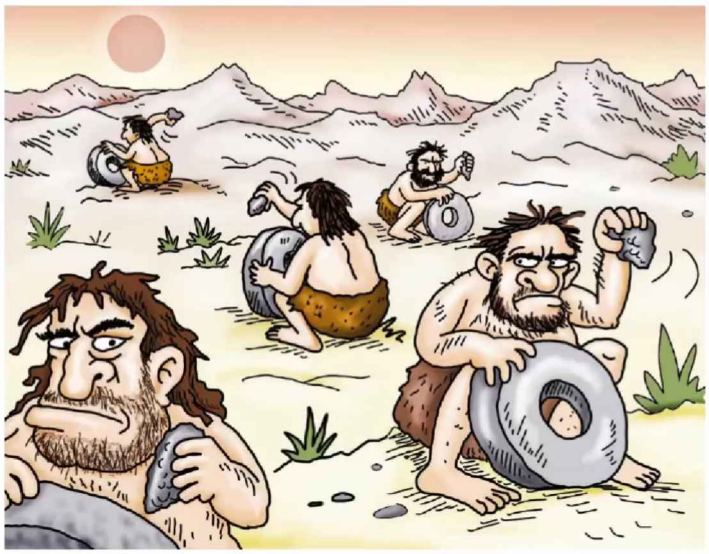
Fig.1. ROS的集成性质

首先记住一句话，对于我们比赛而言：

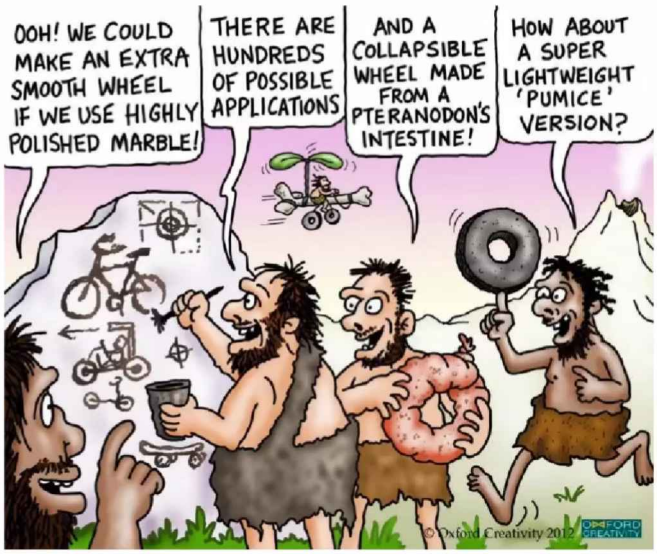
> 机械是骨肉，硬件是心脏，电控是大脑，视觉是灵魂！

*机器人体系是相当庞大的，其复杂度之高，以至于没有任何个人、组织甚至公司能够独立完成系统性的机器人研发工作。一种更合适的策略是：让机器人研发者专注于自己擅长的领域，其他模块则直接复用相关领域更专业研发团队的实现，当然自身的研究也可以被他人继续复用。这种基于"复用"的分工协作，遵循了**不重复发明轮子**的原则，显然是可以大大提高机器人的研发效率的，尤其是随着机器人硬件越来越丰富，软件库越来越庞大，这种复用性和模块化开发需求也愈发强烈。*

提高机器人研发中的软件复用率



传统模式



现代模式

*一家名为 **柳树车库（Willow Garage）**的机器人公司发布了 **ROS**(机器人操作系统)，ROS是一套机器人通用软件框架，可以提升功能模块的复用性，并且随着该系统的不断迭代与完善，如今 ROS 已经成为机器人领域的事实标准。*

既然他这么强大，那他具有哪些特点呢？首先来看下面一张图

Fig.2. ROS的组成

- **代码复用:**ROS的目标不是成为具有最多功能的框架，ROS的主要目标是支持机器人技术研发中的代码*重用*。
- **分布式:**ROS是进程（也称为*Nodes*）的分布式框架,ROS中的进程可分布于不同主机，不同主机协同工作，从而分散计算压力

- **语言独立性：**包括Java，C++，Python等。为了支持更多应用开发和移植，ROS设计为一种语言弱相关的框架结构，使用简洁，中立的定义语言描述模块间的消息接口，在编译中再产生所使用语言的目标文件，为消息交互提供支持，同时允许消息接口的嵌套使用
- **松耦合:**ROS中功能模块封装于独立的功能包或元功能包，便于分享，功能包内的模块以节点为单位运行，以ROS标准的IO作为接口，开发者不需要关注模块内部实现，只要了解接口规则就能实现复用,实现了模块间点对点的松耦合连接
- **工具包丰富：**ROS具有很多常用的可以用来仿真调试的软件功能包，例如可以建立具有物理参数虚拟世界的Gazebo，三维可视化平台rviz，可以实时绘制参数值的rqt_plot等。
- **免费且开源：**开发者众多，功能包多 盈利固然是一个实实在在的目的，但是人类正是通过学习他人已有的知识才得以不断加速发展，许多开源社区的目的为了降低大家学习的门槛，使得入门的学者更有方向性，能够站在前人的肩膀上快速成长。

1.2 ROS常用功能

ROS的应用，如图：

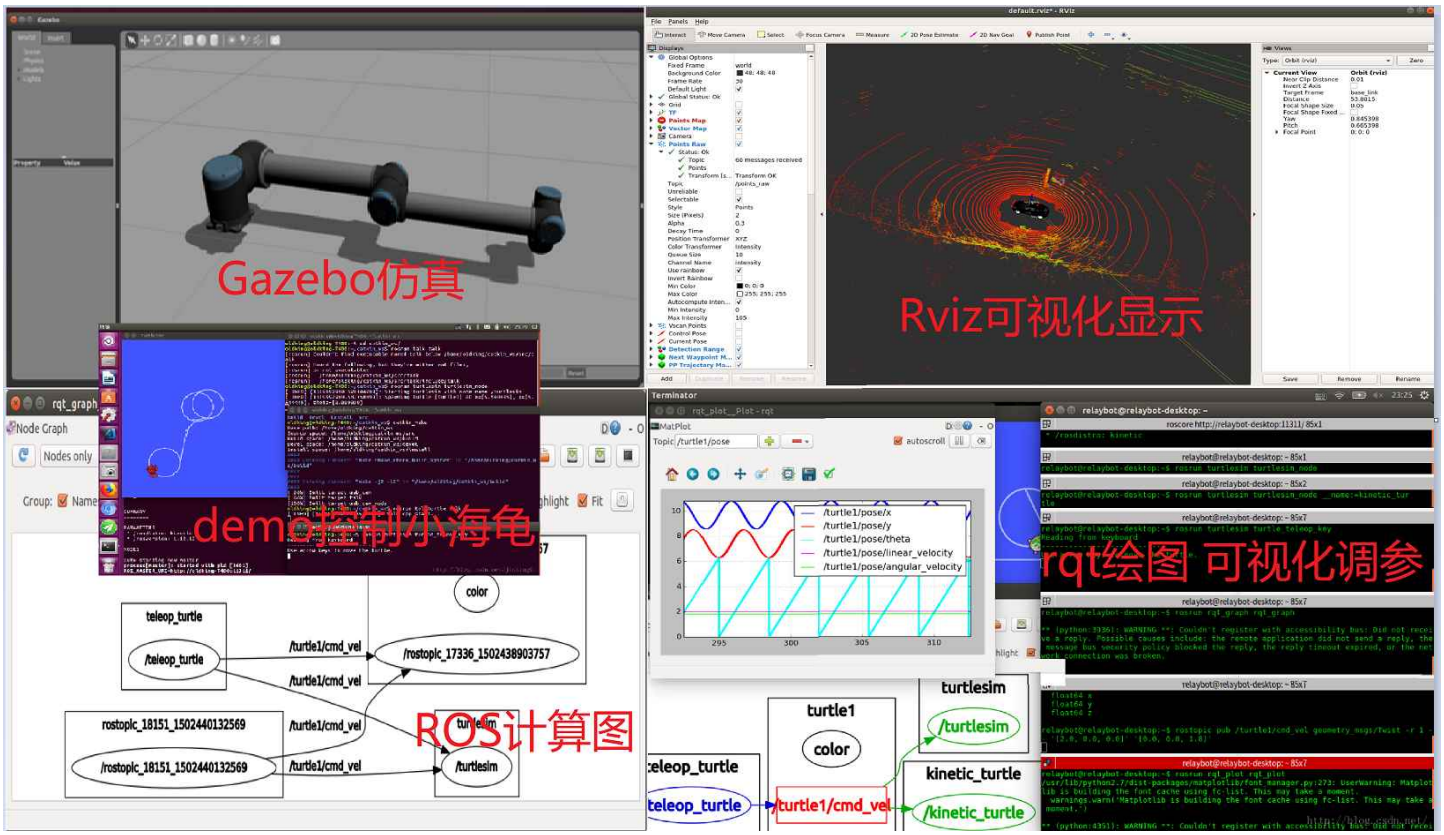


Fig.3. ROS常用功能

2、ROS组定位

ROS集成了大量工具、库、协议。我们队伍ROS小组的名称定为ROS的原因是还有一层Rise of SuperPower，即“SuperPower战队崛起”的双关含义。

可能会有同学有疑问了，“机械是骨肉，硬件是心脏，电控是大脑，视觉是灵魂！”，怎么没有ROS组呢。机械是基石，他们用SolidWorks搓出了一台台机器人，赋予机器人以骨肉；电控重在对关节电机的控制，并对操作手的意图作出响应，而本赛季我们正在研发的平衡步兵则需要对双足机器人进行平衡控制，这分别对应了一个人控制运动逻辑的大脑和控制身体平衡的小脑；视觉强调传感器信息的获取与计算，计算敌人的运动轨迹，对机器人进行定位和导航，也可以说是大脑，但我更倾向于将其解读为给一台机器人赋予真正的灵魂；硬件组研制的超级电容，对于机器人的运行功率上限有极大的提升效果，就像小宇宙爆发一样，因此我将其比喻为心脏。

说了这么多，那么ROS相较于其他组别的定位又是什么呢，我个人的理解是血液。ROS的核心功能之一是他突出的通信机制和分布式框架，一方面，ROS可以多线程并行运行多个程序，通过roslaunch这一指令将他们集成起来，非常方便管理，易于将功能模块化封装；另一方面，ROS通过发布topic话题，来实现不同程序之间的通信，这样电控和视觉的感知和控制程序之间就可以实现很好的沟通。这就好比人体内的血液，是实现人体内器官实现物质交换的重要溶剂，ROS组也将作为信息沟通的重要桥梁。

本赛季ROSP的定位是使用 `SP_Control` 完成工程机器人和哨兵机器人的控制；制定并优化测试流程；并深化工程机器人的自动取矿和自定义控制模块、哨兵机器人的导航和决策模块。此外，在11月初开始对基于ROS2的 `SP2_Control` 进行测试，在寒假集训期间完成 `SP2_Control` 在步兵机器人上的部署。在赛季中期形成基于 `SP_Control` 的工程机器人、哨兵机器人完整控制框架，完成基于 `SP2_Control` 的步兵机器人部署。最终，在赛季末开始基于 `SP_Control` 的机器人向 `SP2_Control` 的迁移。

ROSP的最终目的是优化本队RM机器人的整体开发流程，并将更多的研发时间投入到算法层面的开发中，而非重复造轮子上。

3、ROS下实现"Hello World"

3.1 C++版本

ROS中可以使用C++和Python进行编程，ROSP的工作由于牵涉到比较多的数据结构和底层IO操作，因此使用C++作为开发语言。本节向大家展示如何在ROS下使用C++编写经典的“Hello World”程序。

> 💻 C++运行效率高但是开发效率低，而Python则反之，基于二者互补的特点，ROS设计者分别设计了roscpp和rospy库， 前者旨在成为ROS的高性能库，而后者一般用于对性能无要求的场景，旨在提高开发效率。

我们假设大家已经创建了ROS工作空间，那么为了完成“Hello World”程序，我们还需顺序进行以下的工作：

1. 创建新的功能包

```
// 在你的ROS工作空间下执行以下命令行
cd src
catkin_create_pkg hello_world roscpp std_msgs
```

上述命令，会在工作空间下生成一个名称为 `hello_world` (即catkin_create_pkg第一个参数)的功能包，该功能包依赖于roscpp、std_msgs，其中roscpp是使用C++实现的库，std_msgs是标准消息库，创建ROS功能包时，一般都会依赖于这两个库实现。

2. 新建并编辑cpp文件

```
cd hello_world/src
touch main.cpp
```

然后对新建的main.cpp进行编辑。

```
#include "ros/ros.h"

int main(int argc,char *argv[])
{
    //执行ros节点初始化
    ros::init(argc,argv,"Hello_World")//Hello_World是节点名称
    //创建ros节点句柄(非必须)
    ros::NodeHandle n;
    //控制台输出，Hello World
    ROS_INFO("Hello World");

    return 0;
}
```

3. 编辑CMakeLists.txt文件

```
add_executable(hello_world_node
  src/main.cpp
)
target_link_libraries(hello_world_node
  ${catkin_LIBRARIES}
)
```

`add_executable` 用于告诉编译器我们需要使用哪些**cpp文件**来编译形成**可执行文件**。在本例中，我们需要使用源文件src/main.cpp生成名为hello_world_node的可执行文件。该可执行文件的名称在ROS下也即是默认节点名称。

4. 编译

```
cd 自定义的工作空间
catkin_make
```

5. 执行

```
cd 自定义的工作空间
source ./devel/setup.bash
// rosrun 功能包名 节点名称（与add_executable第一个参数一致）
rosrun hello_world hello_world_node
```

💻 每一次打开新的终端，都必须执行一次 `source ./devel/setup.bash` 配置环境参数，这个过程实在是太讨人厌了。但我们可通过更改 `bash` 文件来避免这一重复劳动。在任意终端内输入下述命令（工作空间替换为自己的ROS工作空间名）：

`echo “source ~/工作空间/devel/setup.bash”>>~/.bashrc`

如此操作后，下一次执行该工作空间内的功能包节点就无需再输入 `source ./devel/setup.bash` 指令了。

3.2 Python版本

根据你已经创建的C++版本的ROS的工作空间和功能包，接下来可以直接进入核心步骤，使用Python编写程序实现：

1.进入 ros 包添加 scripts 目录并编辑 python 文件

```
cd ros包
mkdir scripts
```

新建 python 文件: (文件名自定义)

```
#! /usr/bin/env python"""
    Python 版 HelloWorld
```

```
3
4 """import rospy
5
6 if __name__ == "__main__":
7     rospy.init_node("Hello")
8     rospy.loginfo("Hello World!!!!")
```

2.为 python 文件添加可执行权限

```
1 chmod +x 自定义文件名.py
2 Copy
```

3.编辑 ros 包下的 CamkeList.txt 文件

```
1 catkin_install_python(PROGRAMS scripts/自定义文件名.py
2   DESTINATION ${CATKIN_PACKAGE_BIN_DESTINATION}
3 )
```

4.进入工作空间目录并编译

```
1 cd 自定义空间名称
2 catkin_make
```

5.进入工作空间目录并执行

先启动命令行1：

```
1 roscore
```

再启动命令行2：

```
1 cd 工作空间
2 source ./devel/setup.bash
3 rosrun 包名 自定义文件名.py
```

输出结果: `Hello World!!!!`

4、ROS架构

4.1 ROS文件系统

ROS文件系统级指的是在硬盘上ROS源代码的组织形式，其结构大致可以如下图所示：

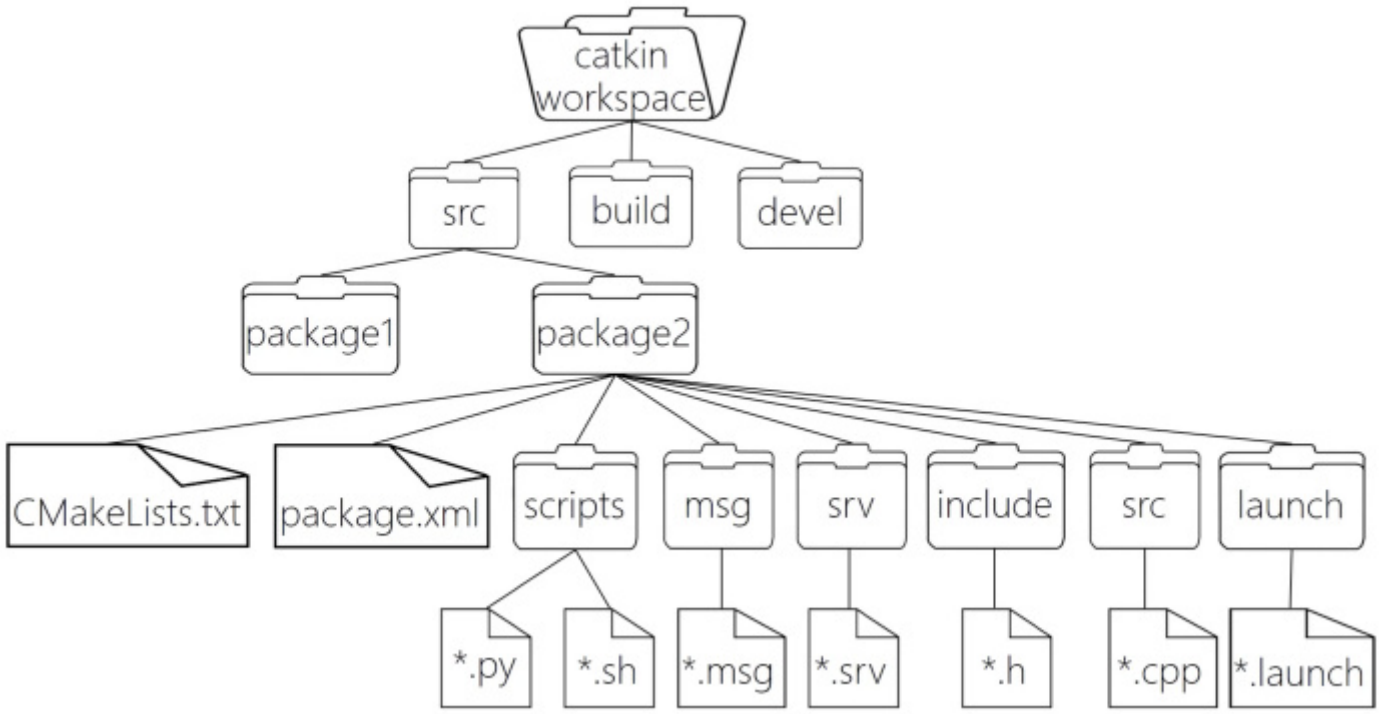


Fig.4. ROS工作空间下的文件结构

```
WorkSpace --- 自定义的工作空间

    |--- build:编译空间，用于存放CMake和catkin的缓存信息、配置信息和其他中间文件。

    |--- devel:开发空间，用于存放编译后生成的目标文件，包括头文件、动态&静态链接库、可执行

    |--- src: 源码

        |-- package：功能包(ROS基本单元)包含多个节点、库与配置文件，包名所有字母小写，只

            |-- CMakeLists.txt 配置编译规则，比如源文件、依赖项、目标文件

            |-- package.xml 包信息，比如:包名、版本、作者、依赖项...(以前版本是 manife

            |-- scripts 存储python文件

            |-- src 存储C++源文件

            |-- include 头文件
```

```
20
21              |-- msg 消息通信格式文件
22
23              |-- srv 服务通信格式文件
24
25              |-- action 动作格式文件
26
27              |-- launch 可一次性运行多个节点
28
29              |-- config 配置信息
30
31          |-- CMakeLists.txt: 编译的基本配置
```

ROS 文件系统中部分目录和文件前面编程中已经有所涉及，比如功能包的创建、src目录下cpp文件的编写、scripts目录下python文件的编写、launch目录下launch文件的编写，并且也配置了 package.xml 与 CMakeLists.txt 文件。其他目录下的内容后面教程将会再行介绍，当前我们主要介绍: package.xml 与 CMakeLists.txt 这两个配置文件。

4.2 ROS计算图

4.2.1 计算图简介

前面介绍的是ROS文件结构，是磁盘上 ROS 程序的存储结构，是静态的，而 ros 程序运行之后，不同的节点之间是错综复杂的，ROS 中提供了一个实用的工具:rqt_graph。

rqt_graph能够创建一个显示当前系统运行情况的动态图形。ROS 分布式系统中不同进程需要进行数据交互，计算图可以以点对点的网络形式表现数据交互过程。rqt_graph是rqt程序包中的一部分。

4.2.2 计算图安装

如果前期把所有的功能包（package）都已经安装完成，则直接在终端窗口中输入

rosrun rqt_graph rqt_graph

如果未安装则在终端（terminal）中输入

```
1 $ sudo apt install ros-<distro>-rqt
2 $ sudo apt install ros-<distro>-rqt-common-plugins
```

请使用你的ROS版本名称（比如:kinetic、melodic、Noetic等）来替换掉<distro>。

例如当前版本是 Noetic,就在终端窗口中输入

```
1 $ sudo apt install ros-noetic-rqt
2 $ sudo apt install ros-noetic-rqt-common-plugins
```

3.计算图演示

接下来以 ROS 内置的小乌龟案例来演示计算图

首先，按照前面所示，运行案例

然后，启动新终端，键入: rqt_graph 或 rosrun rqt_graph rqt_graph，可以看到类似下图的网络拓扑图，该图可以显示不同节点之间的关系。

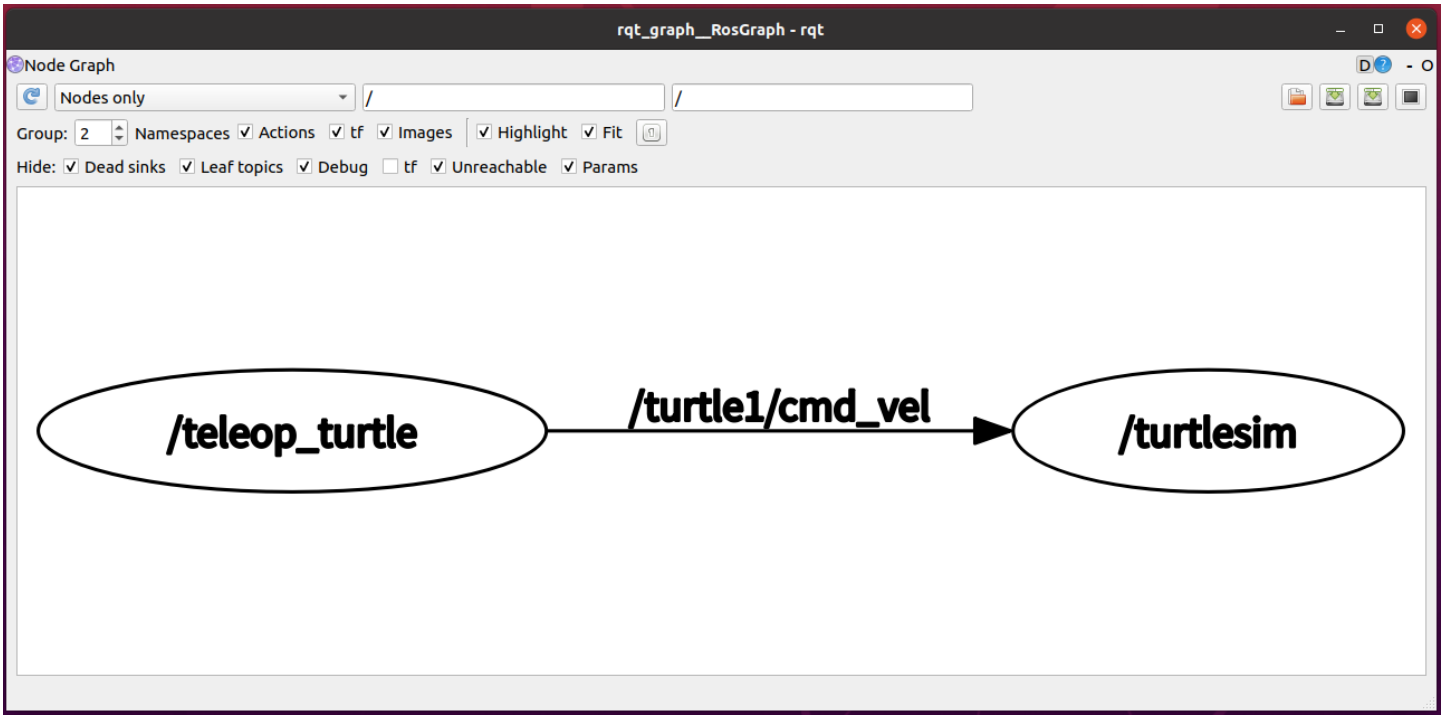


Fig.5. ROS计算图

5、SP_Control

5.1 The Introduction of SP_Control

1. 该项目受启发于广东工业大学的rm-control项目
2. RM传统的嵌入式代码在本节特指基于官方C板代码框架的嵌入式代码

`SP_Control` 是SuperPower战队开发的基于ROS1的无下位机机器人通用控制框架。通过对外部设备进行合理的抽象，使得任何一种支持CAN、UART的设备，只需通过 `hw_config` 文件声明就可以快速接入ROS体系中，并使用ROS提供的其他功能包进行开发，降低了开发部署的难度。

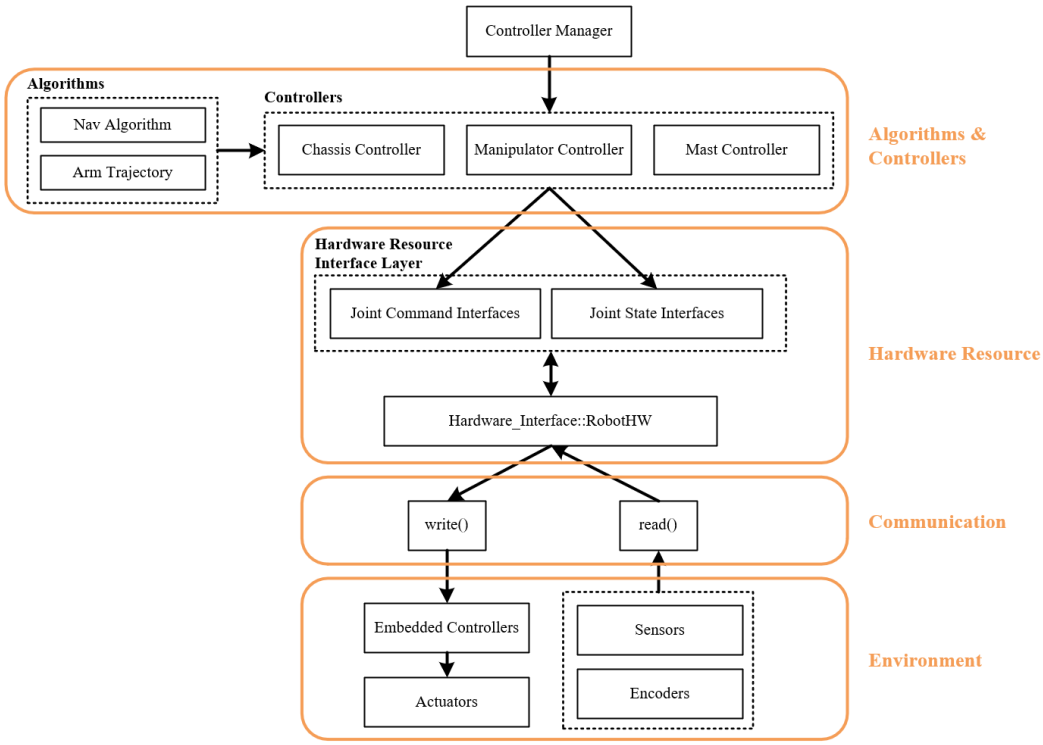


Fig.6. SP_Control抽象框架图

从抽象框架图上来看，架构将整个机器人控制系统分成四层，从顶向下分别为：顶层算法（Algorithms & Controller）、硬件资源（Hardware Resource）、通讯层（Communication）、环境（Environment）。

- 顶层算法是指机器人控制的相关算法，包括运动规划算法、运动控制算法、电机控制算法（如PID）等；
- 硬件资源是完成硬件抽象的部分，其包括硬件资源池（Hardware Resource Interface Layer）和抽象硬件层（Hardware Interface）。抽象硬件层负责加工外设的数据并完成抽象化（比如将不同协议电机的数据处理为相同的数据结构），并将处理后的数据上载至硬件资源池中；
- 通讯层是上位机与下层设备（如电机、传感器、外设）建立通讯的部分，`SP_Control`使用了SocketCAN和虚拟串口完成通讯层的搭建；
- 环境是指机器人运行的环境，包括仿真环境与真实环境，受益于`ROS-Control`的外接接口，在保证上层算法不进行修改的情况下，通过改写launch文件参数，可快速地使用Gazebo、UE4在内的多种包括物理引擎的软件进行仿真，这也保证了该框架具有Sim2Real的潜力。

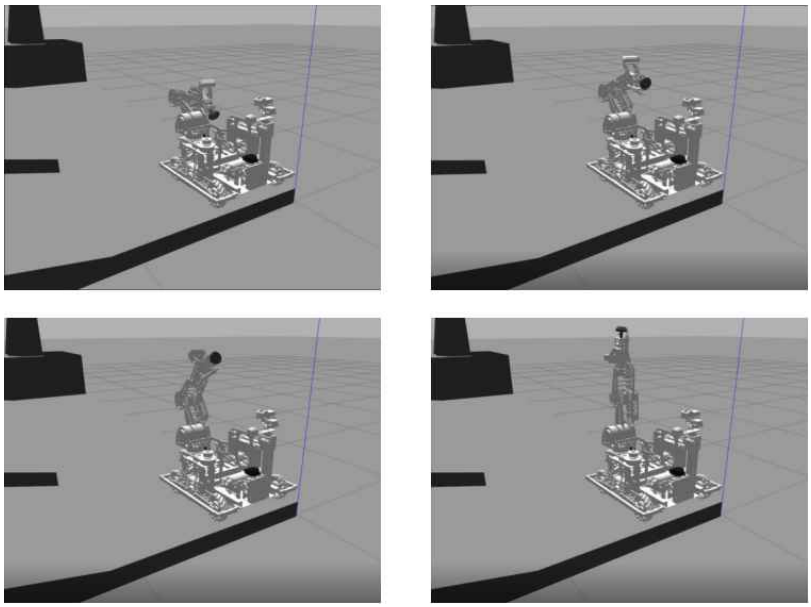


Fig.7. Gazebo仿真环境下的工程机器人

5.2 SP_Control的优势

对比RM赛场上传统使用的嵌入式开发框架，`SP_Control`有若干实用的特性：

1. 顶层算法与底层驱动解耦合

在嵌入式开发流程中，底层驱动与顶层算法往往有不同程度的耦合。这就迫使队员在研发中需要同时对顶层算法和底层驱动有开发的能力，这无疑提升了研发门槛，且这种耦合也会导致代码开发管理更为困难。

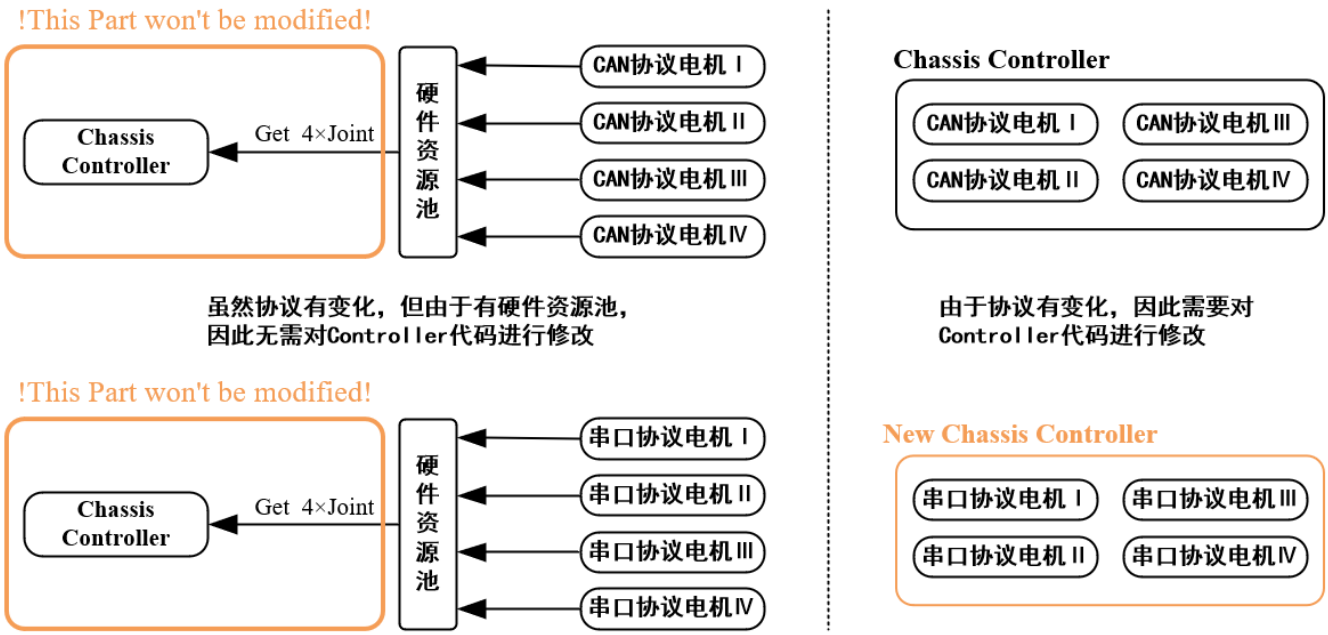


Fig.8. 两种开发流程更换电机协议的示意图

举例来讲，若我们需要将底盘四个使用`CAN`协议的电机更换为基于`RS485`的电机，那么在RM传统嵌入式开发流程中，须在其`Chassis Controller`中考虑修改现有的数据结构以及发送和接收流程。也就是说`Chassis Controller`的编码工作与电机使用的具体协议有关，或者

说底盘控制任务代码与电机协议存在一一对应的关系。而`SP_Control`架构下，将所有电机都抽象为了`Actuator`(`Joint`)类型，因此更换电机协议也无需更改底盘控制任务代码。

在`SP_Control`项目中，借助于ROS包管理机制，我们设计了两个基础功能包：`sp_hw`，`sp_controller`。前者负责底层驱动的所有任务，后者负责机器人的具体控制，两者互不依赖、互不干扰。如此便实现了代码的模块化开发和解耦合。

> 💻 解耦合可以降低开发的难度，并且能够形成“术业有专攻”的开发分工。

2. 更简便的调试与可视化流程

这一点显而易见，由于`SP_Control`所有的代码都运行于上位机平台中，因此所有的数据都可以直接获得，调试也可以借助成熟的工具（例如GDB，或者自己手动看程序报错）。而RM传统嵌入式开发流程中可视化需要借助串口或者外设（例如JLink或者正点原子的无线烧录调试器）将数据转发至上位机可视化，Debug一些复杂的问题也难以直接获得堆栈信息。

在`SP_Control`中，我们常常直接使用Rviz对程序中使用到的数据进行可视化，且在大多数情况下，无需编写额外可视化代码或者外接额外转发设备。

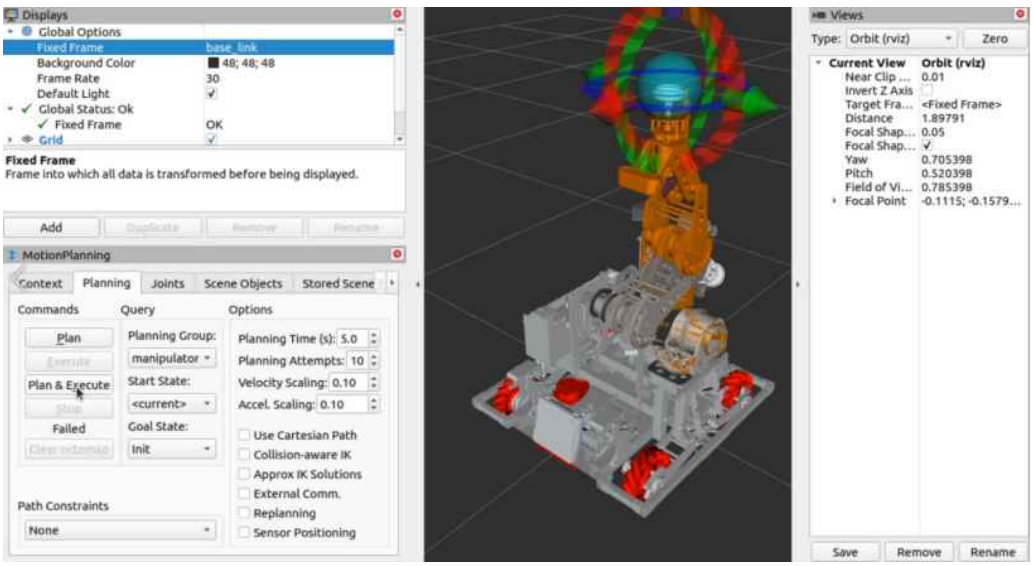


Fig.9. 借助Rviz对工程机器人进行可视化

3. 高代码复用率

`SP_Control`在设计时认为RM机器人是不同任务模块的集合。例如可以将工程机器人分解为**麦轮底盘模块、机械臂模块、升降桅杆模块**；可以将步兵机器人分解为**麦轮底盘模块、云台模块、发射模块、裁判系统模块**。容易注意到两者都具有麦轮底盘模块，从控制代码角度上来说（暂时不考虑功率限制），工程和步兵麦轮底盘模块的差异点仅是尺寸的不同。除开底盘模块，不同兵种间还存在很多重复的模块，那么这些重复的模块是否可以通过某种设计避免重复开发呢？

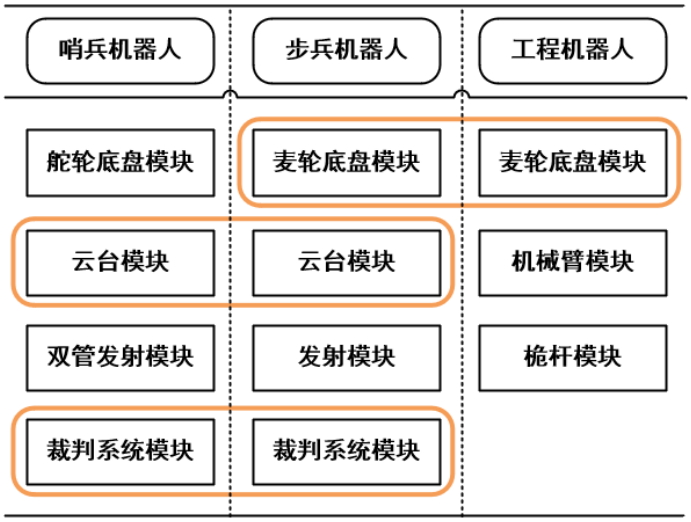


Fig.10. 任务模块划分及重复模块示意

答案是肯定的。我们通过ROS的 `Plugin` 和 `Parameter Server` 机制实现了任务模块插件化。因此不同机器人麦轮底盘模块运行的代码实际上完全一致，不同的参数则通过 `Parameter Server` 在运行时提供给任务模块代码。

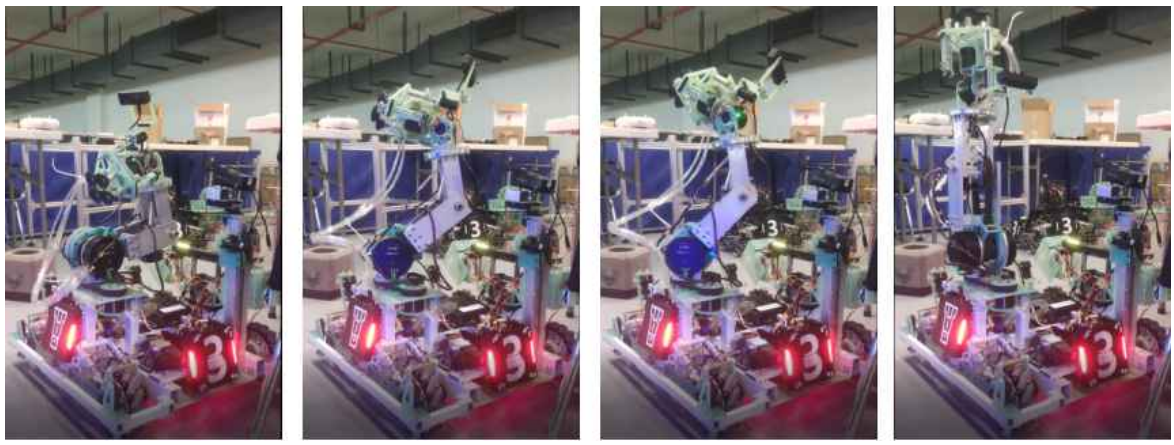


Fig. 11. 通过SP_Control控制的工程机器人（机械臂）

作业

熟悉今天上课所讲的流程，通过VScode集成实现C++版本及Python版本的helloworld

http://www.autolabor.com.cn/book/ROSTutorials/chapter1/14-ros-ji-cheng-kai-fa-huan-jing-da-jian/142-an-zhuang-vscode.html

下次课我们要讲的内容是常用电机简介，电机协议，使用can_bus 读写电机数据，电机参数与电机数据转换，时间为9月19日晚上19：00-20：30