AMOVLAB
阿|木|实|验|室

阿木实验室

Mavros进阶课程

讲师：凯迪

AMOVLAB
阿木实验室

目录

一、基础知识

1、工具链的安装
2、创建节点与编译
3、ROS编程基础
4、mavros消息订阅与发布

二、编程实践

1、mavros位置跟踪
2、mavros姿态跟踪
3、订阅传感器数据
4、mavros与slam
5、mavros与PWM

三、总结引申

1、代码架构分析
2、课程总结

0、课程简介

- 本课程偏重于工程实践，通过一个个实践项目，帮助大家熟悉mavros的使用。所涉及的面比较广，基本上涵盖了mavros常用的应用场景。

- 本课程对控制理论的讲解较少，偏重于工程应用，重点讲解ROS系统的应用，对于提高编程水平，构建软件架构是用帮助的。

- 本课程适用于有一定无人机开发基础的初级开发者，可作为学习教程，也可作为开发过程中的参考资料。

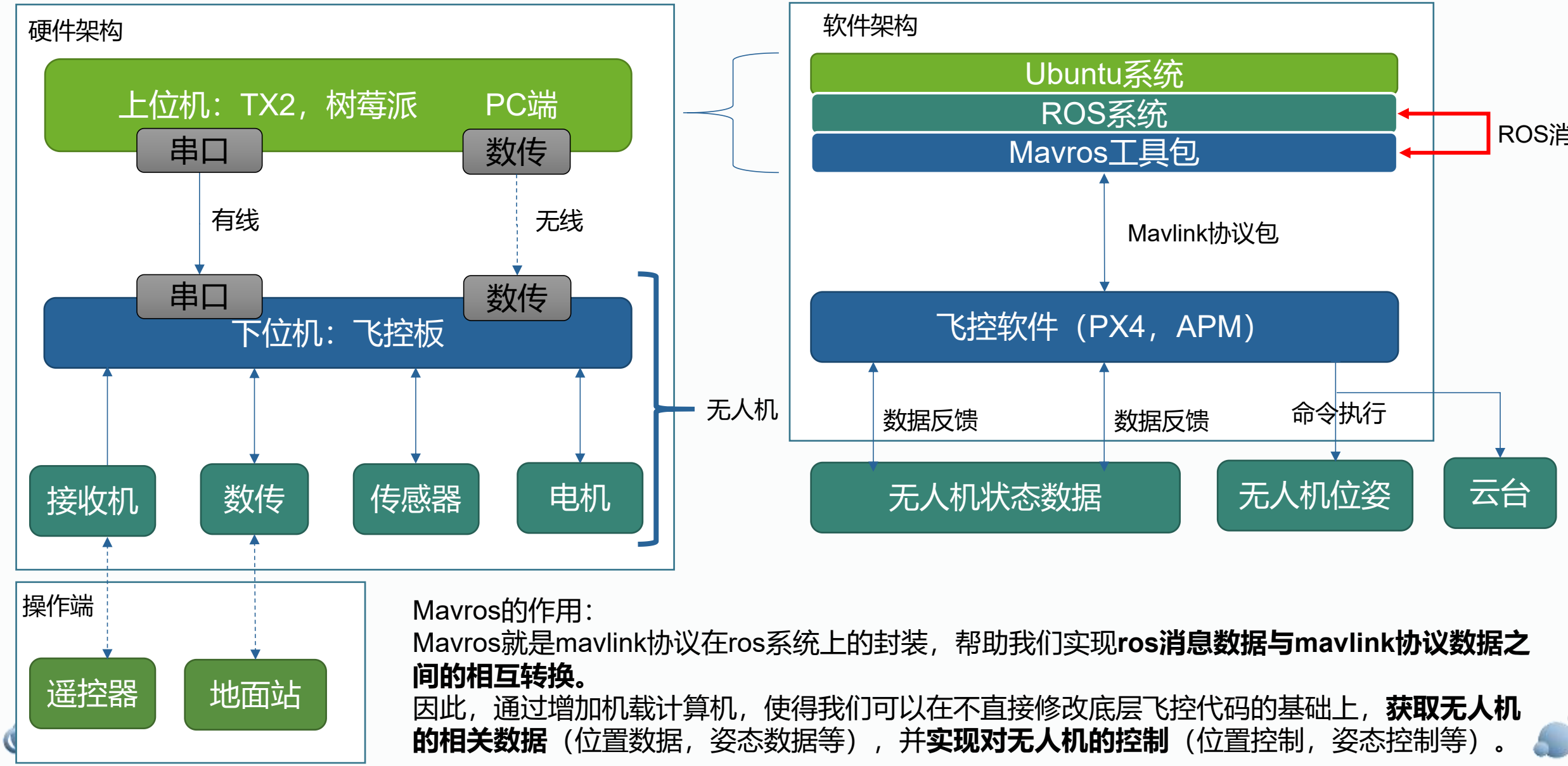
AMOVLAB
阿木实验室

01

基础知识

工欲善其事，必先利其器

应用mavros操作无人机的软硬件架构



Mavros的作用：

Mavros就是mavlink协议在ros系统上的封装，帮助我们实现**ros消息数据与mavlink协议数据之间的相互转换。**

因此，通过增加机载计算机，使得我们可以在不直接修改底层飞控代码的基础上，**获取无人机的相关数据**（位置数据，姿态数据等），并**实现对无人机的控制**（位置控制，姿态控制等）。

AMOVLAB
阿木实验室

410轴距开源无人机

1、工具链的安装

- 工具链：VMware虚拟机 + Ubuntu 18.04操作系统 + ROS Melodic + mavros包 + gazebo仿真工具
- 虚拟机的安装

  https://www.vmware.com/cn/products/workstation-pro/workstation-pro-evaluation.html
- 虚拟机安装Ubuntu 18.04操作系统
  - Ubuntu 18.04系统下载: http://59.80.44.44/old-releases.ubuntu.com/202106232130/893764253980CECE9AF66AA844A72923/releases/18.04.4/ubuntu-18.04.4-desktop-amd64.iso
  - 参考教程: https://blog.csdn.net/qq_37618797/article/details/81192091
- 安装ROS系统
  - 换源，国内源：中科大源，阿里源，清华源

    编辑(vim 或者 gedit) /etc/apt/sources.list，更换为国内源，然后保存文件
    然后执行以下两行命令：
    sudo apt-get update
    sudo apt-get upgrade
  - 安装ROS系统步骤:http://wiki.ros.org/melodic/Installation/Ubuntu

1、工具链的安装

- 安装mavros步骤

  https://docs.px4.io/master/en/ros/mavros_installation.html

  - 二进制安装

  二进制包里面包括了已经编译完成,可以直接运行的程序。你通过sudo apt-get install来进行下载和解包(安装),执行完该指令后就可以马上使用了。因此这种方式简单快捷，适合比较固定、无需改动的程序。

  - 源码安装

  源代码包里是程序的原始代码,在你的计算机上必须经过编译,生成了可执行的二进制文件,方可运行。一些个人开发的程序、第三方修改或者你希望修改的程序都应当通过源代码包的来编译安装。

  - **推荐用源码安装的方式安装mavros**

- 安装gazebo步骤

  https://docs.px4.io/master/en/dev_setup/dev_env_linux_ubuntu.html#rosgazebo

Ubuntu 18.04 国内替换源

中科大源

```
deb https://mirrors.ustc.edu.cn/ubuntu/ bionic main restricted universe multiverse
deb-src https://mirrors.ustc.edu.cn/ubuntu/ bionic main restricted universe multiverse
deb https://mirrors.ustc.edu.cn/ubuntu/ bionic-updates main restricted universe multiverse
deb-src https://mirrors.ustc.edu.cn/ubuntu/ bionic-updates main restricted universe multiverse
deb https://mirrors.ustc.edu.cn/ubuntu/ bionic-backports main restricted universe multiverse
deb-src https://mirrors.ustc.edu.cn/ubuntu/ bionic-backports main restricted universe multiverse
deb https://mirrors.ustc.edu.cn/ubuntu/ bionic-security main restricted universe multiverse
deb-src https://mirrors.ustc.edu.cn/ubuntu/ bionic-security main restricted universe multiverse
deb https://mirrors.ustc.edu.cn/ubuntu/ bionic-proposed main restricted universe multiverse
deb-src https://mirrors.ustc.edu.cn/ubuntu/ bionic-proposed main restricted universe multiverse
```

Ubuntu 18.04 国内替换源

阿里源

```
deb http://mirrors.aliyun.com/ubuntu/ bionic main restricted universe multiverse
deb http://mirrors.aliyun.com/ubuntu/ bionic-security main restricted universe multiverse
deb http://mirrors.aliyun.com/ubuntu/ bionic-updates main restricted universe multiverse
deb http://mirrors.aliyun.com/ubuntu/ bionic-proposed main restricted universe multiverse
deb http://mirrors.aliyun.com/ubuntu/ bionic-backports main restricted universe multiverse
deb-src http://mirrors.aliyun.com/ubuntu/ bionic main restricted universe multiverse
deb-src http://mirrors.aliyun.com/ubuntu/ bionic-security main restricted universe multiverse
deb-src http://mirrors.aliyun.com/ubuntu/ bionic-updates main restricted universe multiverse
deb-src http://mirrors.aliyun.com/ubuntu/ bionic-proposed main restricted universe multiverse
deb-src http://mirrors.aliyun.com/ubuntu/ bionic-backports main restricted universe multiverse
```

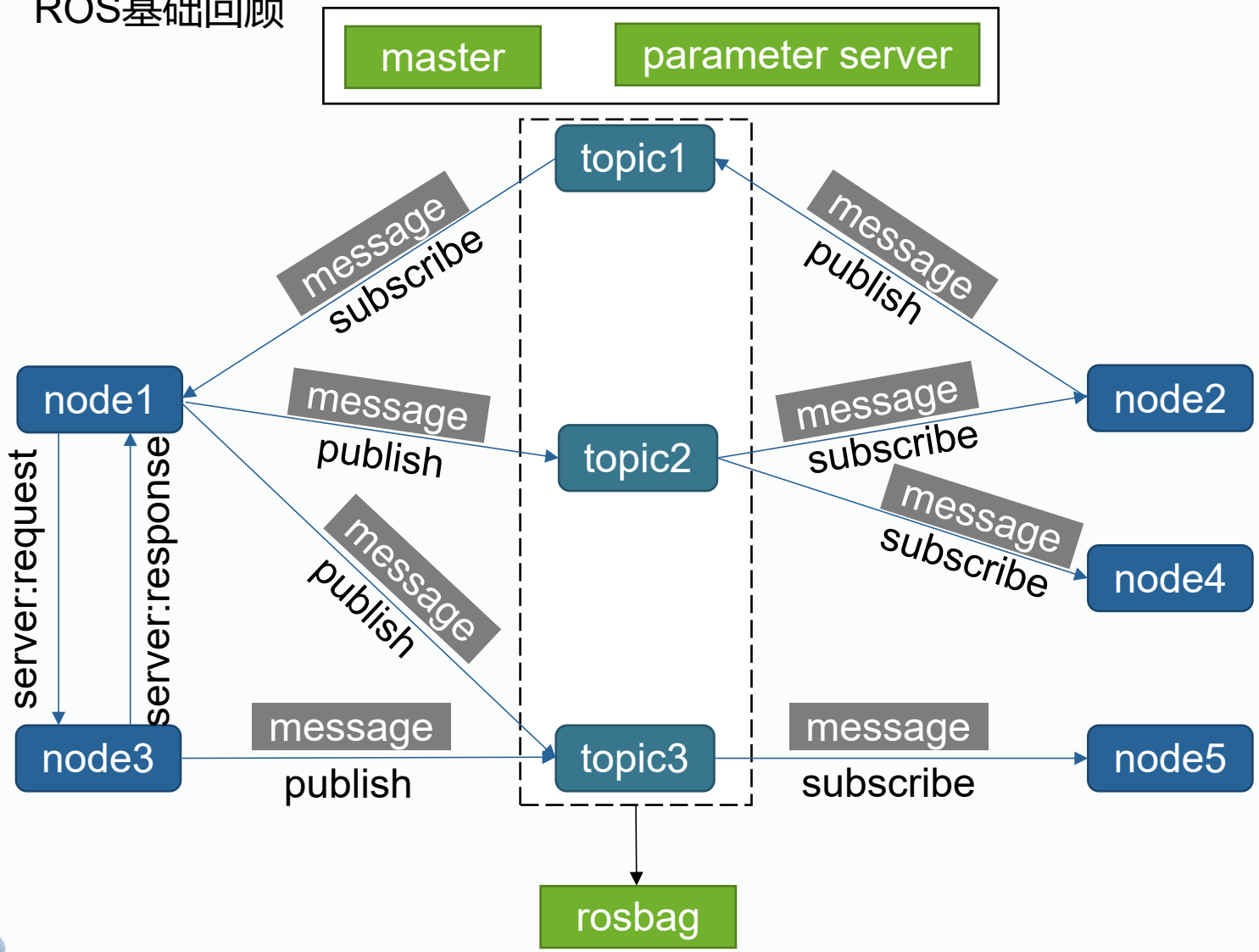
Ubuntu 18.04 国内替换源

清华源：

```
deb https://mirrors.tuna.tsinghua.edu.cn/ubuntu/ bionic main restricted universe multiverse
deb-src https://mirrors.tuna.tsinghua.edu.cn/ubuntu/ bionic main restricted universe multiverse
deb https://mirrors.tuna.tsinghua.edu.cn/ubuntu/ bionic-updates main restricted universe multiverse
deb-src https://mirrors.tuna.tsinghua.edu.cn/ubuntu/ bionic-updates main restricted universe multiverse
deb https://mirrors.tuna.tsinghua.edu.cn/ubuntu/ bionic-backports main restricted universe multiverse
deb-src https://mirrors.tuna.tsinghua.edu.cn/ubuntu/ bionic-backports main restricted universe multiverse
deb https://mirrors.tuna.tsinghua.edu.cn/ubuntu/ bionic-security main restricted universe multiverse
deb-src https://mirrors.tuna.tsinghua.edu.cn/ubuntu/ bionic-security main restricted universe multiverse
deb https://mirrors.tuna.tsinghua.edu.cn/ubuntu/ bionic-proposed main restricted universe multiverse
deb-src https://mirrors.tuna.tsinghua.edu.cn/ubuntu/ bionic-proposed main restricted universe multiverse
```

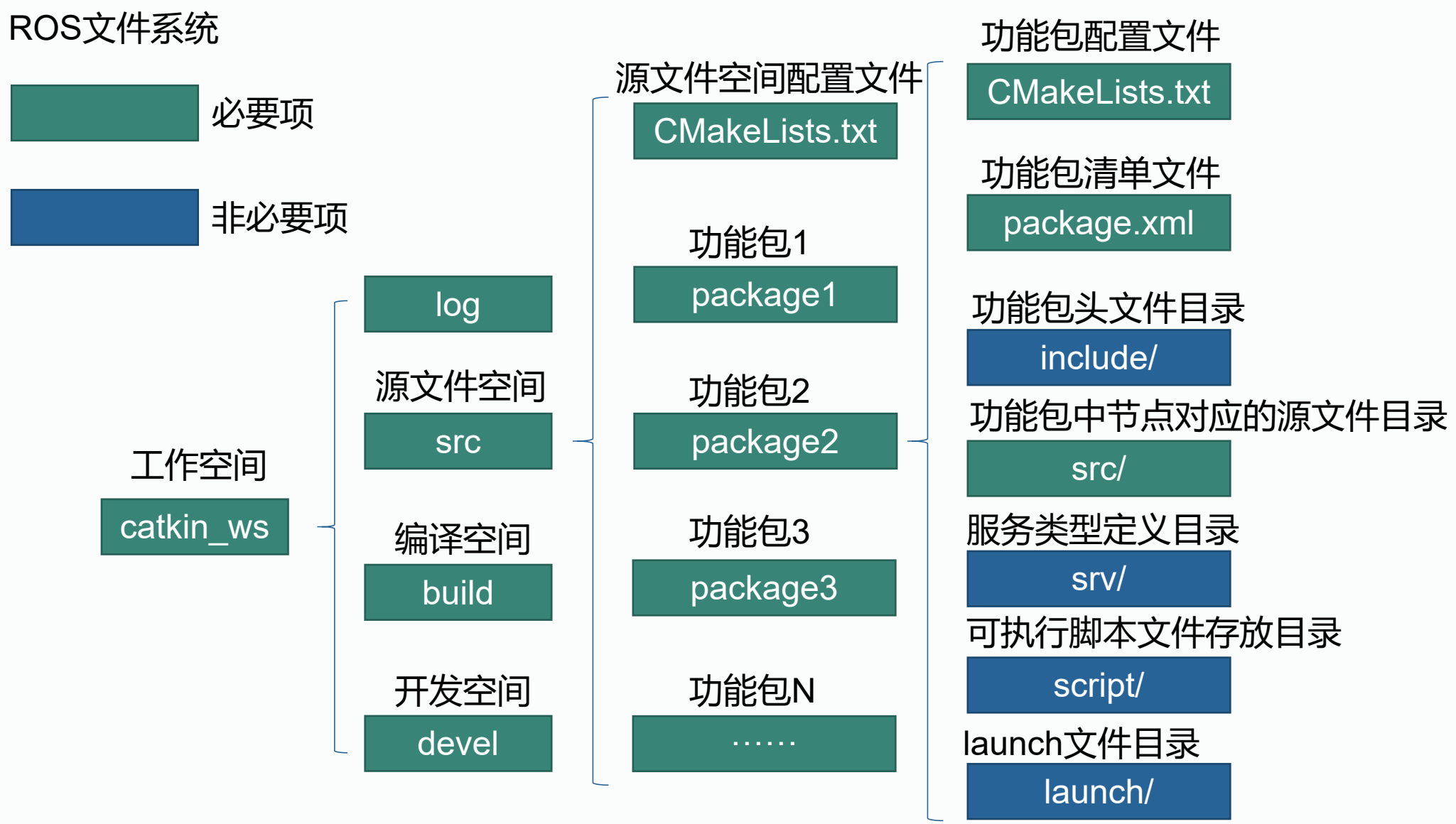
2、ROS创建节点与编译

ROS基础回顾



- 节点（node）：各自**独立的可执行**文件，能够通过**话题**（topic）、**服务**（server）或**参数服务器**（parameter server）与其他节点进行通信
- 节点可以独立于其他节点启动（**rosrun**），多个节点也可以以任何顺序启动（**roslaunch**）
- 节点可以运行在**同一台计算机**中，或者分布于一个**计算机网络**中（ROS多机通信）
- 一个节点既可以是订阅器也可以是发布器

2、ROS创建节点与编译



2、ROS创建节点与编译

src源文件空间：这个文件夹放置各个功能包和一个用于这些功能包的CMake配置文件CMakeLists.txt。这里做一下说明，由于ROS中的源码采用catkin工具进行编译，而catkin工具又是基于cmake技术的，所以我们会在src源文件空间和各个功能包中都会见到一个文件CMakeLists.txt，这个文件就是起编译配置的作用。
build编译空间：这个文件夹放置CMake和catkin编译功能包时产生的缓存、配置、中间文件等。
devel开发空间：这个文件夹放置编译好的可执行程序，这些可执行程序是不需要安装就能直接运行的。一旦功能包源码编译和测试通过后，可以将这些编译好的可执行文件直接导出与其他开发人员分享。
CMakeLists.txt功能包配置文件：用于这个功能包cmake编译时的配置文件。
package.xml功能包清单文件：用xml的标签格式标记这个功能包的各类相关信息，比如包的名称、依赖关系等。主要作用是为了更容易的安装和分发功能包。
include/<package_name>功能包头文件目录：你可以把你的功能包程序包含的*.h头文件放在这里，include下之所以还要加一级路径<package_name>是为了更好的区分自己定义的头文件和系统标准头文件，<package_name>用实际功能包的名称替代。不过这个文件夹不是必要项，比如有些程序没有头文件的情况。
msg非标准消息定义目录：消息是ROS中一个进程（节点）发送到其他进程（节点）的信息，消息类型是消息的数据结构，ROS系统提供了很多标准类型的消息可以直接使用，如果你要使用一些非标准类型的消息，就需要自己来定义该类型的消息，并把定义的文件放在这里。不过这个文件夹不是必要项，比如程序中只使用标准类型的消息的情况。
srv服务类型定义目录：服务是ROS中进程（节点）间的请求/响应通信过程，服务类型是服务请求/响应的数据结构，服务类型的定义放在这里。如果要调用此服务，你需要使用该功能包名称和服务名称。不过这个文件夹不是必要项，比如程序中不使用服务的情况。
scripts可执行脚本文件存放目录：这里用于存放bash、python或其他脚本的可执行文件。不过这个文件夹不是必要项，比如程序中不使用可执行脚本的情况。
launch文件目录：这里用于存放*.launch文件，*.launch文件用于启动ROS功能包中的一个或多个节点，在含有多个节点启动的大型项目中很有用。不过这个文件夹不是必要项，节点也可以不通过launch文件启动。
src功能包中节点源文件存放目录：一个功能包中可以有多个进程（节点）程序来完成不同的功能，每个进程（节点）程序都是可以单独运行的，这里用于存放这些进程（节点）程序的源文件，你可以在这里再创建文件夹和文件来按你的需求组织源文件，源文件可以用c++、python等来书写。

2、ROS创建节点与编译

ROS文件系统相关的常用命令
http://wiki.ros.org/ROS/Tutorials/NavigatingTheFilesystem

- rospack
  - 允许您获取有关包的信息。在本教程中，我们将只介绍 find 选项，它返回包的路径。

```
rospack    find    [package_name]
```

- roscd
  - 将目录 (cd) 直接更改为包或堆栈

```
roscd    <package-or-stack>[/subdir]
```

- rosls
  - 允许直接在ls包中的文件，按package包的名称而不是绝对路径

```
 rosls    <package-or-stack>[/subdir]
```

2、ROS创建节点与编译

1）ROS创建工作空间和ROS程序包

· 创建工作空间目录

```
mkdir  -p  catkin_ws/src
```

· 创建ROS程序包

```
cd  catkin_ws/src
catkin_create_pkg   my_minimal_nodes  roscpp  std_msgs
```

```
kd@ubuntu:~/catkin_ws_1/src$ catkin_create_pkg   my_minimal_nodes  roscpp  std_m
sgs

Created file my_minimal_nodes/package.xml
Created file my_minimal_nodes/CMakeLists.txt
Created folder my_minimal_nodes/include/my_minimal_nodes
Created folder my_minimal_nodes/src
Successfully created files in /home/kd/catkin_ws_1/src/my_minimal_nodes. Please
adjust the values in package.xml.
```

```
kd@ubuntu:~/catkin_ws_1/src$ cd my_minimal_nodes/
kd@ubuntu:~/catkin_ws_1/src/my_minimal_nodes$ ls
CMakeLists.txt  include  package.xml  src
```

```
<?xml version="1.0"?>
<package format="2">
  <name>my_minimal_nodes</name>
  <version>0.0.0</version>
  <description>The my_minimal_nodes package</description>

  <!-- One maintainer tag required, multiple allowed, one person per tag -->
  <!-- Example:  -->
  <!-- <maintainer email="jane.doe@example.com">Jane Doe</maintainer> -->
  <maintainer email="kd@todo.todo">kd</maintainer>


  <!-- One license tag required, multiple allowed, one license per tag -->
  <!-- Commonly used license strings: -->
  <!--   BSD, MIT, Boost Software License, GPLv2, GPLv3, LGPLv2.1, LGPLv3 -->
  <license>TODO</license>


  <!-- Url tags are optional, but multiple are allowed, one per tag -->
  <!-- Optional attribute type can be: website, bugtracker, or repository -->
  <!-- Example: -->
  <!-- <url type="website">http://wiki.ros.org/my_minimal_nodes</url> -->


  <!-- Author tags are optional, multiple are allowed, one per tag -->
  <!-- Authors do not have to be maintainers, but could be -->
  <!-- Example: -->
  <!-- <author email="jane.doe@example.com">Jane Doe</author> -->


  <!-- The *depend tags are used to specify dependencies -->
  <!-- Dependencies can be catkin packages or system dependencies -->
  <!-- Examples: -->
  <!-- Use depend as a shortcut for packages that are both build and exec dependencies -->
  <!--   <depend>roscpp</depend> -->
  <!--   Note that this is equivalent to the following: -->
  <!--   <build_depend>roscpp</build_depend> -->
  <!--   <exec_depend>roscpp</exec_depend> -->
  <!-- Use build_depend for packages you need at compile time: -->
  <!--   <build_depend>message_generation</build_depend> -->
  <!-- Use build_export_depend for packages you need in order to build against this package: -->
  <!--   <build_export_depend>message_generation</build_export_depend> -->
  <!-- Use buildtool_depend for build tool packages: -->
  <!--   <buildtool_depend>catkin</buildtool_depend> -->
  <!-- Use exec_depend for packages you need at runtime: -->
  <!--   <exec_depend>message_runtime</exec_depend> -->
  <!-- Use test_depend for packages you need only for testing: -->
  <!--   <test_depend>gtest</test_depend> -->
  <!-- Use doc_depend for packages you need only for building documentation: -->
  <!--   <doc_depend>doxygen</doc_depend> -->
  <buildtool_depend>catkin</buildtool_depend>
  <build_depend>roscpp</build_depend>
  <build_depend>std_msgs</build_depend>
  <build_export_depend>roscpp</build_export_depend>
  <build_export_depend>std_msgs</build_export_depend>
  <exec_depend>roscpp</exec_depend>
  <exec_depend>std_msgs</exec_depend>

  <!-- The export tag contains other, unspecified, tags -->
  <export>
    <!-- Other tools can request additional information be placed here -->

  </export>
</package>
```

要与新程序包的名字一致，这一点很重要。如果你的目录名和程序包名不一致，ROS会出现混乱

作为代码的作者输入你的名字和电子邮箱地址，这可以公开分享你的成就

明确地声明了roscpp和std_msgs程序包的依赖。在创建程序包时，这两项都明确列为依赖项。在之后的开发中，我们可能会引入大量第三方代码（其他程序包）。为了把这些程序包结合在一起，我们需要把它们添加到package.xml文件中。可以通过以下步骤实现：

编辑程序包中的package.xml文件，模仿已有的roscpp和std_msgs依赖声明，在build_depend，build_export_depend和exec_depend中添加需要利用的新的程序包。

2、ROS创建节点与编译

1）编译ROS节点

· 编写一个最小的ROS程序

-在一个终端中，进入已经创建的my_minimal_nodes程序包的src目录，打开编辑器，创建一个名为minimal.cpp的文件，并输入以下代码。

```
#include <ros/ros.h>
#include <std_msgs/Float64.h>
int main (int argc,char **argv)
{
        ros::init(argc,argv,"minimal_node");
        ros::NodeHandle n;
        std_msgs::Float64 input_float ;
        input_float.data = 0.5;
        while(ros::ok())
        {
                ROS_INFO_STREAM("input_float:"<<input_float);
        }
        return 0;
}
```

2、ROS创建节点与编译

· 编译ROS

- 修改CMakeLists.txt
- catkin_make或者catkin build

```
[build] Found '1' packages in 0.0 seconds.
[build] Package table is up to date.
Starting  >>> my_minimal_nodes
Finished  <<< my_minimal_nodes                [ 0.2 seconds ]
[build] Summary: All 1 packages succeeded!
[build]   Ignored:   None.
[build]   Warnings:  None.
[build]   Abandoned: None.
[build]   Failed:    None.
[build] Runtime: 0.2 seconds total.
```

为生成的可执行文件选定名字

对应包目录而言在哪里找到源代码

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

· 运行ROS节点

- 在一个终端运行 roscore
- 在另一个终端运行节点：source ~/catkin_ws/devel/setup.bash
  rosrun my_minimal_nodes minimal

```
kd@ubuntu:~/catkin_ws_1$ roscore
... logging to /home/kd/.ros/log/9b32f3a6-d72d-11eb-b24b-000c29a33f13/roslaunch-
ubuntu-5817.log
Checking log directory for disk usage. This may take a while.
Press Ctrl-C to interrupt
Done checking log file disk usage. Usage is <1GB.

started roslaunch server http://ubuntu:44449/
ros_comm version 1.14.10


SUMMARY
========

PARAMETERS
 * /rosdistro: melodic
 * /rosversion: 1.14.10

NODES

auto-starting new master
process[master]: started with pid [5886]
ROS_MASTER_URI=http://ubuntu:11311/

setting /run_id to 9b32f3a6-d72d-11eb-b24b-000c29a33f13
process[rosout-1]: started with pid [5897]
started core service [/rosout]
```

```
[ INFO] [1624788341.798137633]: input_float:data: 0.5
[ INFO] [1624788341.798146956]: input_float:data: 0.5
[ INFO] [1624788341.798154335]: input_float:data: 0.5
[ INFO] [1624788341.798160927]: input_float:data: 0.5
[ INFO] [1624788341.798188667]: input_float:data: 0.5
[ INFO] [1624788341.798246140]: input_float:data: 0.5
[ INFO] [1624788341.798297022]: input_float:data: 0.5
[ INFO] [1624788341.798304272]: input_float:data: 0.5
[ INFO] [1624788341.798311244]: input_float:data: 0.5
[ INFO] [1624788341.798363983]: input_float:data: 0.5
[ INFO] [1624788341.798372599]: input_float:data: 0.5
[ INFO] [1624788341.798379256]: input_float:data: 0.5
[ INFO] [1624788341.798385931]: input_float:data: 0.5
[ INFO] [1624788341.798394169]: input_float:data: 0.5
```

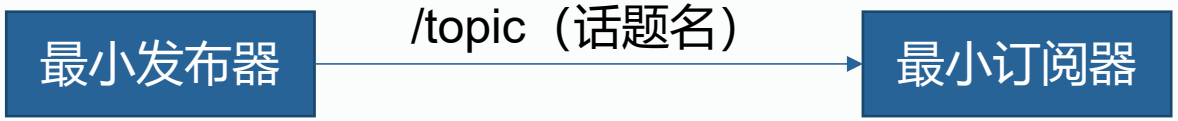
3、ROS编程基础

- 回顾用ROS编写一个订阅器和发布器的代码
- 使用ros完成一个最简单的控制器和最小仿真节点
- 在ros中使用C++类
  - 将ros的代码模块化
  - 针对复杂系统如何去构建ros代码

3、ROS编程基础

◆ 回顾（常见例程，用ros完成一个订阅器，一个发布器）

http://wiki.ros.org/ROS/Tutorials/WritingPublisherSubscriber%28c%2B%2B%29#roscpp_tutorials.2FTutorials.2FWritingPublisherSubscriber.Writing_the_Publisher_Node



发布一下消息到/topic上

订阅/topic，获取发布的消息数据，并打印出来

发布消息的流程：

- 实例化ros Publisher对象，发布消息的类型，话题名，缓存队列大小
- 声明消息类型的变量，对变量进行操作
- 使用ros Publisher对象发布消息变量。

订阅消息的流程：

- 实例化ros::Subscriber对象，话题名，缓存队列大小，回调函数名
- 编写回调函数，回调函数的形参格式为：

const 订阅的消息类型::ConstPtr& msg

注意：

1）**发布消息的类型**与**订阅消息的类型**必须一致，否则会报错

2）**回调函数的形参**需要**严格按照格式定义**，否则会报错

3）注意**缓存队列的大小**，这个值的确定需要从实际需求出发，且与节点运行频率有关，需**要按需设置**

运行结果：

- 运行发布器rosrun minimal_nodes minimal_publisher
- rostopic list //列出所有的活动话题，两个由ROS自己创建，第三个话题“topic1”由发布器自己创建
- rostopic info topic1//显示topic1的基本信息
- rostopic hz topic1//显示topic1话题的频率
- rostopic bw topic1/显示话题消耗的通信带宽，对于识别过渡消耗通信资源的节点会有帮助

- 运行订阅器rosrun minimal_nodes minimal_subscriber

```
kd@ubuntu:~/catkin_ws$ rostopic list
/rosout
/rosout_agg
/topic1
```

```
kd@ubuntu:~/catkin_ws$ rostopic info topic1
Type: std_msgs/Float64

Publishers:
 * /minimal_publisher (http://ubuntu:44225/)

Subscribers: None
```

```
kd@ubuntu:~/catkin_ws$ rostopic hz topic1
subscribed to [/topic1]
no new messages
average rate: 0.999
        min: 1.001s max: 1.001s std dev: 0.00000s window: 2
average rate: 1.000
        min: 1.000s max: 1.001s std dev: 0.00029s window: 3
average rate: 1.000
        min: 1.000s max: 1.001s std dev: 0.00024s window: 4
average rate: 1.000
        min: 1.000s max: 1.001s std dev: 0.00029s window: 5
average rate: 1.000
        min: 0.999s max: 1.001s std dev: 0.00041s window: 6
average rate: 1.000
        min: 0.999s max: 1.001s std dev: 0.00041s window: 7
average rate: 1.000
        min: 0.999s max: 1.001s std dev: 0.00039s window: 8
average rate: 1.000
        min: 0.999s max: 1.001s std dev: 0.00036s window: 9
```

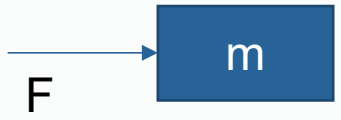
```
average: 8.50B/s
        mean: 8.00B min: 8.00B max: 8.00B window: 11
average: 8.46B/s
        mean: 8.00B min: 8.00B max: 8.00B window: 12
average: 8.42B/s
        mean: 8.00B min: 8.00B max: 8.00B window: 13
average: 8.39B/s
        mean: 8.00B min: 8.00B max: 8.00B window: 14
average: 8.36B/s
        mean: 8.00B min: 8.00B max: 8.00B window: 15
average: 8.33B/s
        mean: 8.00B min: 8.00B max: 8.00B window: 16
average: 8.31B/s
        mean: 8.00B min: 8.00B max: 8.00B window: 17
average: 8.29B/s
```

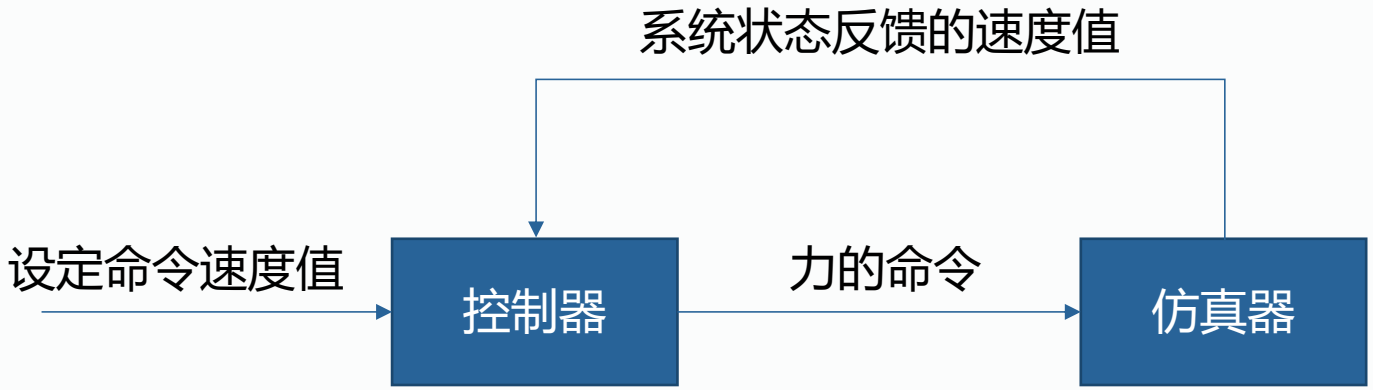
```
[ INFO] [1625501407.179106889]: received value is: 2.650000
[ INFO] [1625501408.178680782]: received value is: 2.651000
[ INFO] [1625501409.178687688]: received value is: 2.652000
[ INFO] [1625501410.179023002]: received value is: 2.653000
[ INFO] [1625501411.178122712]: received value is: 2.654000
[ INFO] [1625501412.179248136]: received value is: 2.655000
[ INFO] [1625501413.178404744]: received value is: 2.656000
[ INFO] [1625501414.179006173]: received value is: 2.657000
```

◆ 用ros编写一个简单的控制器和仿真器

任务：写一个控制器和仿真器

设定速度值为V给控制器，控制器输出F力的大小，使得质量为m的物体达到设定的速度值。





F=ma计算出加速度；加速度积分得到速度值

任务分析：

最小仿真器以仿真公式F=ma，通过加速度的积分来更新速度。

订阅列表：“力的命令”，话题名设定为force_cmd;

发布列表：“系统状态反馈的速度值”，话题名设置为velocity。

最小控制器订阅两个话题，发布一个话题

订阅列表：

“系统状态反馈的速度值”，话题名称：velocity

“设定的命令速度值”，话题名称：vel_cmd

发布列表：“力的命令”，话题名设定为force_cmd;

运行结果：

- rosrun minimal_nodes minimal_simulator//首先运行仿真节点
- rosrun minimal_nodes minimal_controller//再运行控制节点
- rostopic pub -r 10 vel_cmd std_msgs/Float64 1.0//以10hz的频率发布消息数据为1.0到话题“vel_cmd”上

```
[ INFO] [1625502092.758006528]: received velocity value is: 0.000000
[ INFO] [1625502092.758024154]: received velocity command value is: 1.000000
[ INFO] [1625502092.857815049]: force command = 1.000000
[ INFO] [1625502092.857925376]: received velocity value is: 0.000000
[ INFO] [1625502092.857940091]: received velocity command value is: 1.000000
[ INFO] [1625502092.958510747]: force command = 1.000000
[ INFO] [1625502092.958619132]: received velocity value is: 0.090000
[ INFO] [1625502092.958634109]: received velocity command value is: 1.000000
[ INFO] [1625502093.058296183]: force command = 0.910000
[ INFO] [1625502093.058410082]: received velocity value is: 0.190000
[ INFO] [1625502093.058426717]: received velocity command value is: 1.000000
[ INFO] [1625502093.158473047]: force command = 0.810000
[ INFO] [1625502093.158585356]: received velocity value is: 0.281900
[ INFO] [1625502093.158600139]: received velocity command value is: 1.000000
[ INFO] [1625502093.258166712]: force command = 0.718100
[ INFO] [1625502093.258281679]: received velocity value is: 0.363900
[ INFO] [1625502093.258298570]: received velocity command value is: 1.000000
[ INFO] [1625502093.358518762]: force command = 0.636100
[ INFO] [1625502093.358649699]: received velocity value is: 0.436629
[ INFO] [1625502093.358668999]: received velocity command value is: 1.000000
[ INFO] [1625502093.458439374]: force command = 0.563371
[ INFO] [1625502093.458555622]: received velocity value is: 0.501059
[ INFO] [1625502093.458572480]: received velocity command value is: 1.000000
[ INFO] [1625502093.557891331]: force command = 0.498941
```

```
[ INFO] [1625502105.958103567]: force command = 0.000000
[ INFO] [1625502105.958187167]: received velocity value is: 1.000000
[ INFO] [1625502105.958200226]: received velocity command value is: 1.000000
[ INFO] [1625502106.058318163]: force command = 0.000000
[ INFO] [1625502106.058434191]: received velocity value is: 1.000000
[ INFO] [1625502106.058450121]: received velocity command value is: 1.000000
[ INFO] [1625502106.158615549]: force command = 0.000000
[ INFO] [1625502106.158719390]: received velocity value is: 1.000000
[ INFO] [1625502106.158737554]: received velocity command value is: 1.000000
[ INFO] [1625502106.258318298]: force command = 0.000000
[ INFO] [1625502106.258426086]: received velocity value is: 1.000000
[ INFO] [1625502106.258444800]: received velocity command value is: 1.000000
[ INFO] [1625502106.358462375]: force command = 0.000000
[ INFO] [1625502106.358572379]: received velocity value is: 1.000000
[ INFO] [1625502106.358587101]: received velocity command value is: 1.000000
[ INFO] [1625502106.458240604]: force command = 0.000000
[ INFO] [1625502106.458320127]: received velocity value is: 1.000000
[ INFO] [1625502106.458332683]: received velocity command value is: 1.000000
[ INFO] [1625502106.558118199]: force command = 0.000000
[ INFO] [1625502106.558195123]: received velocity value is: 1.000000
[ INFO] [1625502106.558208150]: received velocity command value is: 1.000000
[ INFO] [1625502106.657928660]: force command = 0.000000
```

- 可以看到force_command的数据逐渐变小，直至为0
- Received velocity value的数据逐渐变大，直至为1

◆ 在ros中使用C++类

- ROS代码如果订阅很多个消息，发布多个消息的话，很快会变得过于冗长，若要提高代码效率和代码复用，最好使用类
- 在头文件中定义类：
  - 定义所有成员函数的原型
  - 定义私有和公共数据成员
  - 定义构造函数的原型
- 编写一个单独的实现文件：
  - 包含上面的头文件
  - 包含已经声明成员函数的工作代码
  - 包含在构造函数中封装的必要的初始化的代码

◆ 在ros中使用C++类

使用C++中的类重写刚刚的控制器节点

- 构建文件系统：
  - 在功能包目录下创建一个inlcude/minimal_controller_class的文件夹
  - 并在该文件夹下创建一个minimal_controller_class.h的文件
  - 在src目录下创建minimal_controller_class.cpp文件
- 修改CMakeLists.txt的相关文件，使得ros节点包含进include文件夹中所包含的头文件

```
catkin_package(
 INCLUDE_DIRS include
 LIBRARIES get_master_make_m
 CATKIN_DEPENDS geometry_msgs mavros roscpp std_msgs
#  DEPENDS system_lib
)
```

```
include_directories(
  include
  ${catkin_INCLUDE_DIRS}
  src/include/
)
```

使用C++中的类重写刚刚的控制器节点

- 编写.h文件，在头文件中定义一个类，并在类中定义所有成员函数，成员变量，以及构造函数
- 编写.cpp文件，包含上述的.h文件，包含已经声明的成员函数的工作代码，包含构造函数的相关初始化代码

需要注意的点：

- 在主函数中需要实例化类对象并传入指向nodehandle的指针名称
- 关键字this告诉编译器正在引用此类的当前实例
- 对C++的类相关定义，使用要熟悉

运行程序：

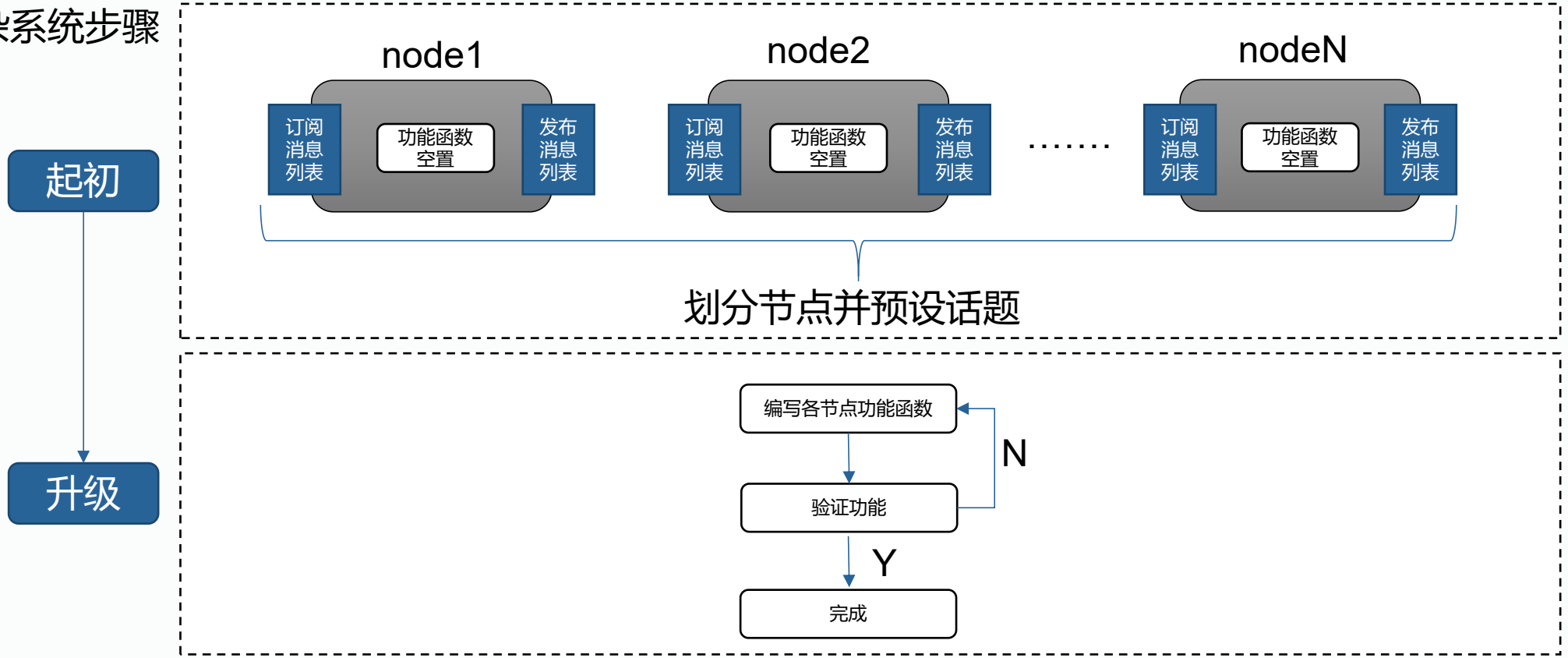
rosrun minimal_nodes minimal_simulator

rosrun minimal_nodes minimal_controller_class

rostopic pub -r 10 vel_cmd std_msgs/Float64 1.0

得到结果与之前的一致。

ROS构建复杂系统步骤

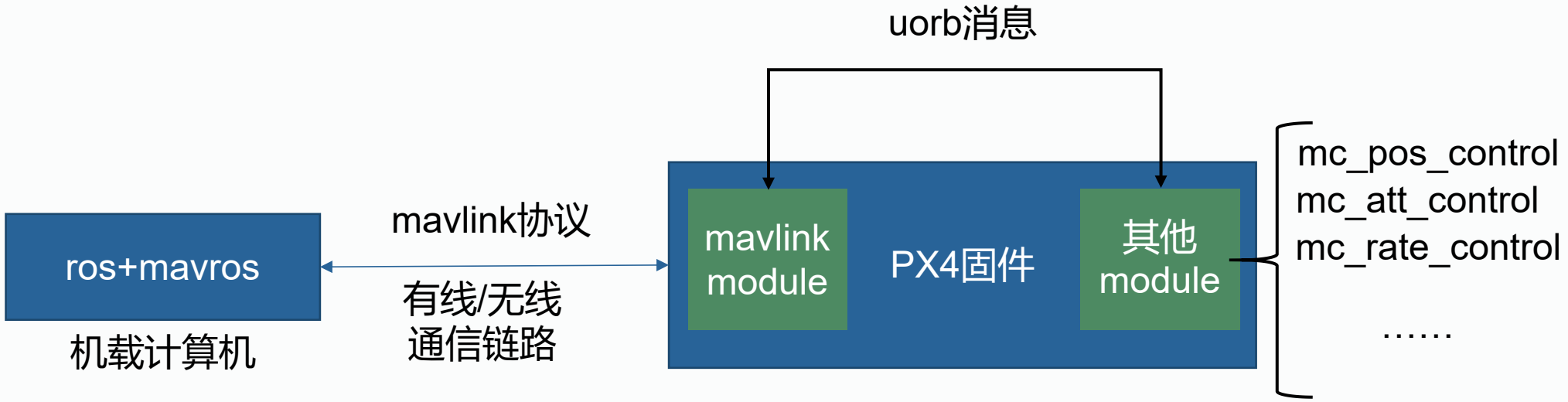


从系统架构的角度来看，ROS有助于实现一个符合预期的软件架构。即从给一个预定的软件架构开始，可以构造一个由若干节点组成的大型系统的骨架。起初每个节点都是简化的节点，可以**通过预设话题（软件接口）发送和接收消息**。架构中的每个模块随后都可以把简化的节点置换为新节点逐步进行**升级**，而且整个系统的其他部分无须改变。ROS支持分布式软件开发和增量测试，这些对于构建大型复杂系统是必要的。

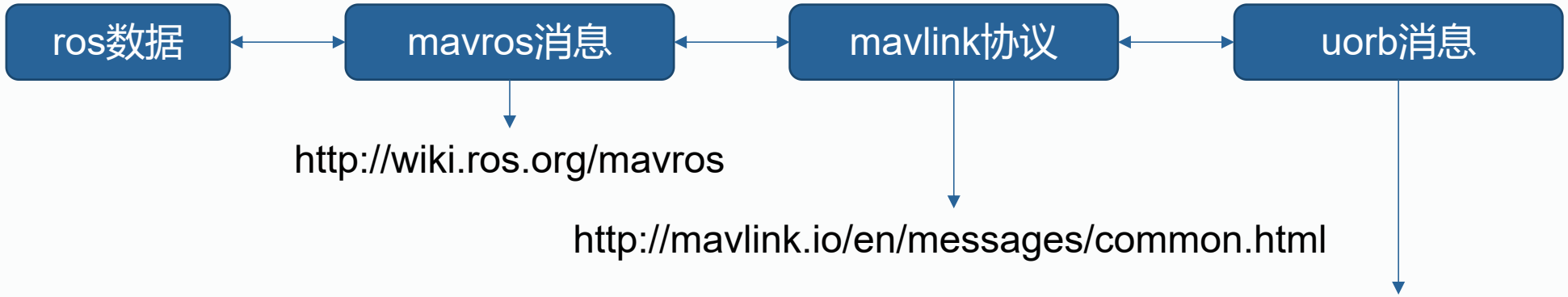
4、mavros消息订阅与发布

- 应用mavros控制无人机的消息流
  - 如何去查阅mavros的消息与mavlink协议之间的关系
  - mavlink协议与uorb消息之间的对应关系
- example offboard例程的仿真与解析
  - 使用gazebo进行软件在环仿真

◆ 应用mavros控制无人机的消息流



消息流间的
对应关系



Example：以mavros中的“setpoint_position/local ” 话题为例

- 1）确定话题的功能
  - 直接阅读mavros中的说明
  - 阅读mavros源码，找到其对应的mavlink协议
  - 查询对应的mavlink协议，阅读该协议对应的说明文档

- 2）确定话题消息的类型
  - 发布消息的时候需要的ROS消息类型(geometry_msgs/PoseStamped)

- 3）确定话题最终转化为的uorb消息
  - 查看PX4固件中的/modules/mavlink/mavlink_receiver.cpp和/modules/mavlink/mavlink_receiver.h

mavlink协议

6.15 setpoint_position

Sent position setpoint using SET_POSITION_TARGET_LOCAL_NED.

6.15.1 Subscribed Topics

话题名称 消息数据类型

~setpoint_position/global (mavros_msgs/GlobalPositionTarget)
Global frame setpoint position. Converts LLA to local ENU.

~setpoint_position/local (geometry_msgs/PoseStamped)
Local frame setpoint position. Only YAW axis extracted from orientation field.

SET_POSITION_TARGET_LOCAL_NED (#84)

[Message] Sets a desired vehicle position in a local north-east-down coordinate frame. Used by an external controller to command the vehicle (manual controller or other system).

Field Name	Type	Units	Values	Description
time_boot_ms	uint32_t	ms		Timestamp (time since system boot).
target_system	uint8_t			System ID
target_component	uint8_t			Component ID
coordinate_frame	uint8_t		MAV_FRAME	Valid options are: MAV_FRAME_LOCAL_NED = 1, MAV_FRAME_LOCAL_OFFSET_NED = 7, MAV_FRAME_BODY_NED = 8, MAV_FRAME_BODY_OFFSET_NED = 9
type_mask	uint16_t		POSITION_TARGET_TYPEMASK	Bitmap to indicate which dimensions should be ignored by the vehicle.
x	float	m		X Position in NED frame
y	float	m		Y Position in NED frame
z	float	m		Z Position in NED frame (note, altitude is negative in NED)
vx	float	m/s		X velocity in NED frame
vy	float	m/s		Y velocity in NED frame
vz	float	m/s		Z velocity in NED frame

```
void
MavlinkReceiver::handle_message(mavlink_message_t *msg)
{
        switch (msg->msgid) {
        case MAVLINK_MSG_ID_COMMAND_LONG:
                handle_message_command_long(msg);
                break;

        case MAVLINK_MSG_ID_COMMAND_INT:
                handle_message_command_int(msg);
                break;

        case MAVLINK_MSG_ID_COMMAND_ACK:
                handle_message_command_ack(msg);
                break;
```

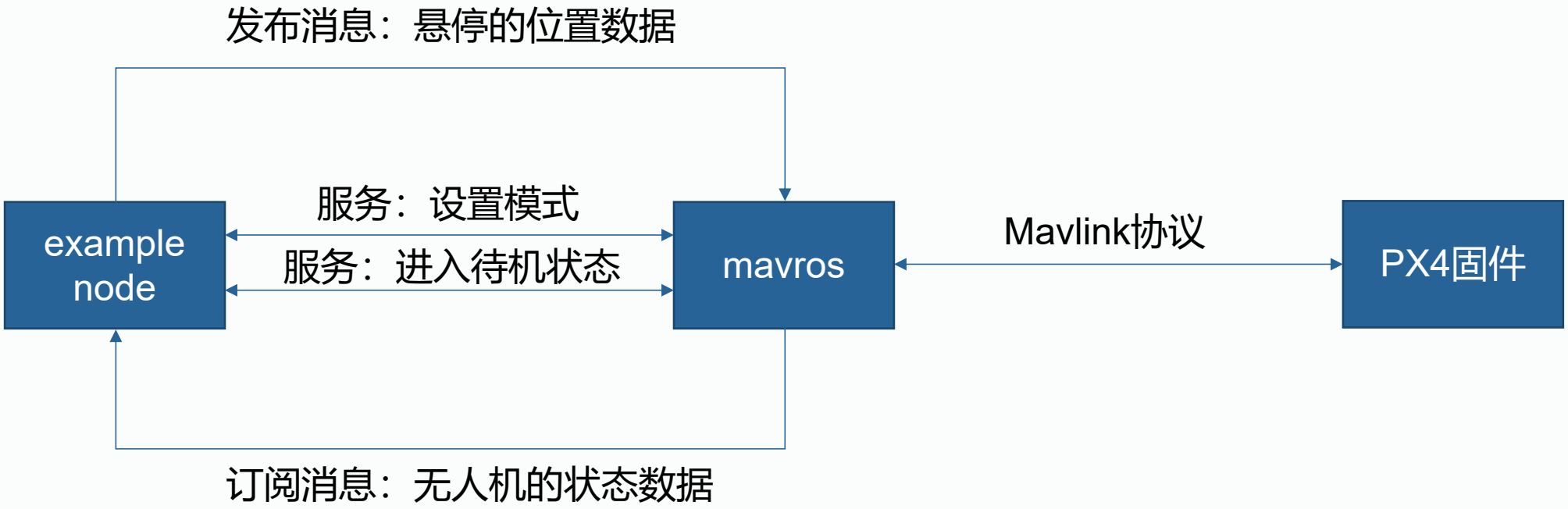
◆ example offboard例程的仿真与解析

1）使用mavros使得PX4进入offboard的模式

2）使用mavros使得PX4进入待机状态

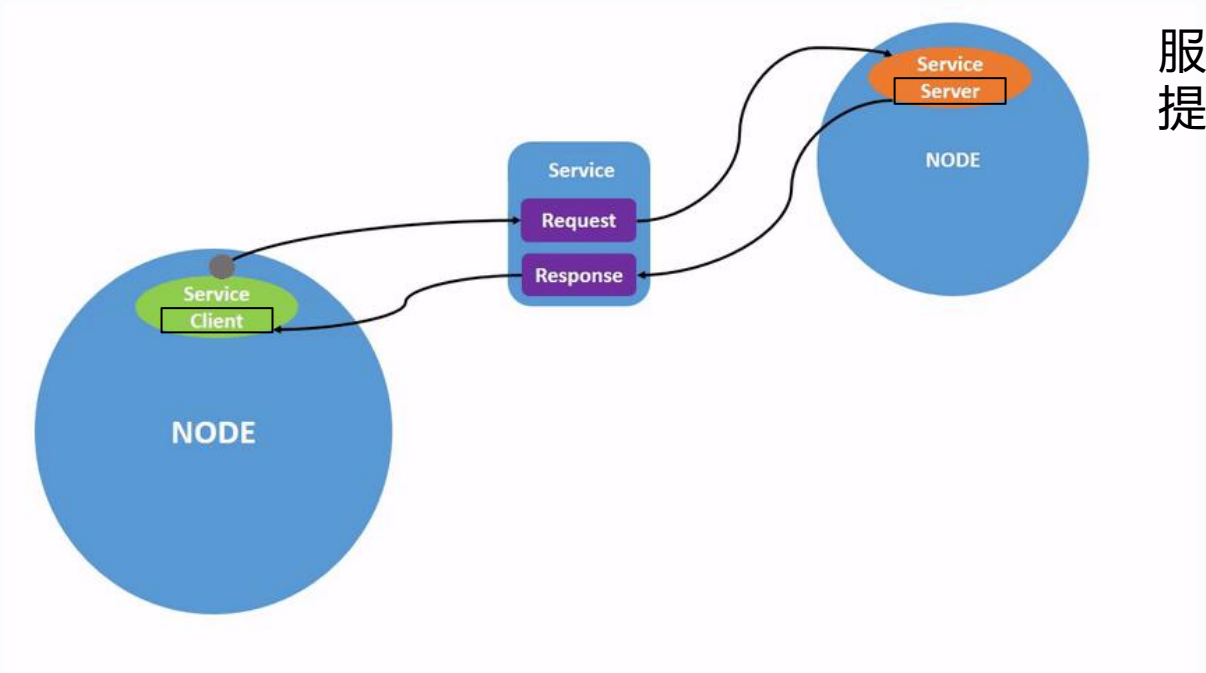
这两步也可以通过**遥控器/地面站**的相关设置来完成；
这两步配置步骤，不需要循环发送数据

3）使用mavros使得无人机悬停在2m的位置 → 需要循环发送数据



add:

1. 之前讲的话题通信是**基于订阅/发布机制**的，无论有没有订阅者，发布者都会周期发布数据，这种模式适合持续数据的收发，比如**传感器数据**。
2. 机器人系统中还有另外一些**配置性质**的数据，并不需要周期处理，此时就要用到另外一种*ROS*通信方式—服务(*Service*)。
3. 服务是基于**客户端/服务器**模型的通信机制，**服务器端只在接收到客户端请求时才会提供反馈数据。**



服务端：
提供反馈

客户端：
发出请求

因此在示例代码中：
客户端为example node；
服务端为mavros。

http://wiki.ros.org/ROS/Tutorials/WritingServiceClient%28c%2B%2B%29

1）确定任务需要用到的mavros话题名称与消息类型
阅读mavros的官方文档

话题/服务的名称	消息类型	功能
/set_mode	mavros_msgs/Setmode	~set_mode (mavros_msgs/SetMode) Set FCU operation mode. See custom mode identifiers at modes page.
/cmd/arming	mavros_msgs::CommandBool	~cmd/arming (mavros_msgs/CommandBool) Change Arming status.
/state	mavros_msgs::State	6.18 sys_status System status plugin. Handles FCU detection. **REQUIRED** never blacklist it! 6.18.1 Published Topics ~state (mavros_msgs/State) FCU state
/setpoint_position/local	geometry_msgs::PoseStamped	~setpoint_position/local (geometry_msgs/PoseStamped) Local frame setpoint position. Only YAW axis extracted from orientation field.

2) 头文件需要包含所需要用到的消息类型的.h文件

```
//在创建功能包的时候需要加入适当的依赖包
catkin_create_pkg offboard_pkg roscpp std_msgs geometry_msgs mavros_msgs
```

```
#include <ros/ros.h>
#include <geometry_msgs/PoseStamped.h>
#include <mavros_msgs/CommandBool.h>
#include <mavros_msgs/SetMode.h>
#include <mavros_msgs/State.h>
```

3）声明发布消息，订阅消息和服务的对象，并编写相应的回调函数

```
//实例化订阅消息的对象，发布消息的对象，服务的对象
ros::Subscriber state_sub = nh.subscribe<mavros_msgs::State>
        ("mavros/state", 10, state_cb);
ros::Publisher local_pos_pub = nh.advertise<geometry_msgs::PoseStamped>
        ("mavros/setpoint_position/local", 10);
ros::ServiceClient arming_client = nh.serviceClient<mavros_msgs::CommandBool>
        ("mavros/cmd/arming");
ros::ServiceClient set_mode_client = nh.serviceClient<mavros_msgs::SetMode>
        ("mavros/set_mode");
```

```
//订阅state话题的回调函数
mavros_msgs::State current_state;
void state_cb(const mavros_msgs::State::ConstPtr& msg){
    current_state = *msg;
}
```

4）确定节点的频率，在进行ros开发的时候，一定要绷住一根弦，频率的影响很大，尤其是在进行多个节点的多个消息的通信的时候。在这里，位置数据的发送频率必须要大于2Hz

```
ros::Rate rate(20.0);
```

5）编写功能代码段

应用mavros控制无人机常用的功能代码段

a. 等待机载计算机与飞控板的通信建立

```
// wait for FCU connection
    while(ros::ok() && !current_state.connected){
        ros::spinOnce();
        rate.sleep();
    }
```

b. 提前开始mavros数据的传输。在进入 Offboard 模式之前，您必须已经开始流式传输设定值。否则模式切换将被拒绝。

```
 geometry_msgs::PoseStamped pose;
    pose.pose.position.x = 0;
    pose.pose.position.y = 0;
    pose.pose.position.z = 2;
for(int i = 100; ros::ok() && i > 0; --i){
        local_pos_pub.publish(pose);
        ros::spinOnce();
        rate.sleep();
    }
```

5）编写功能代码段

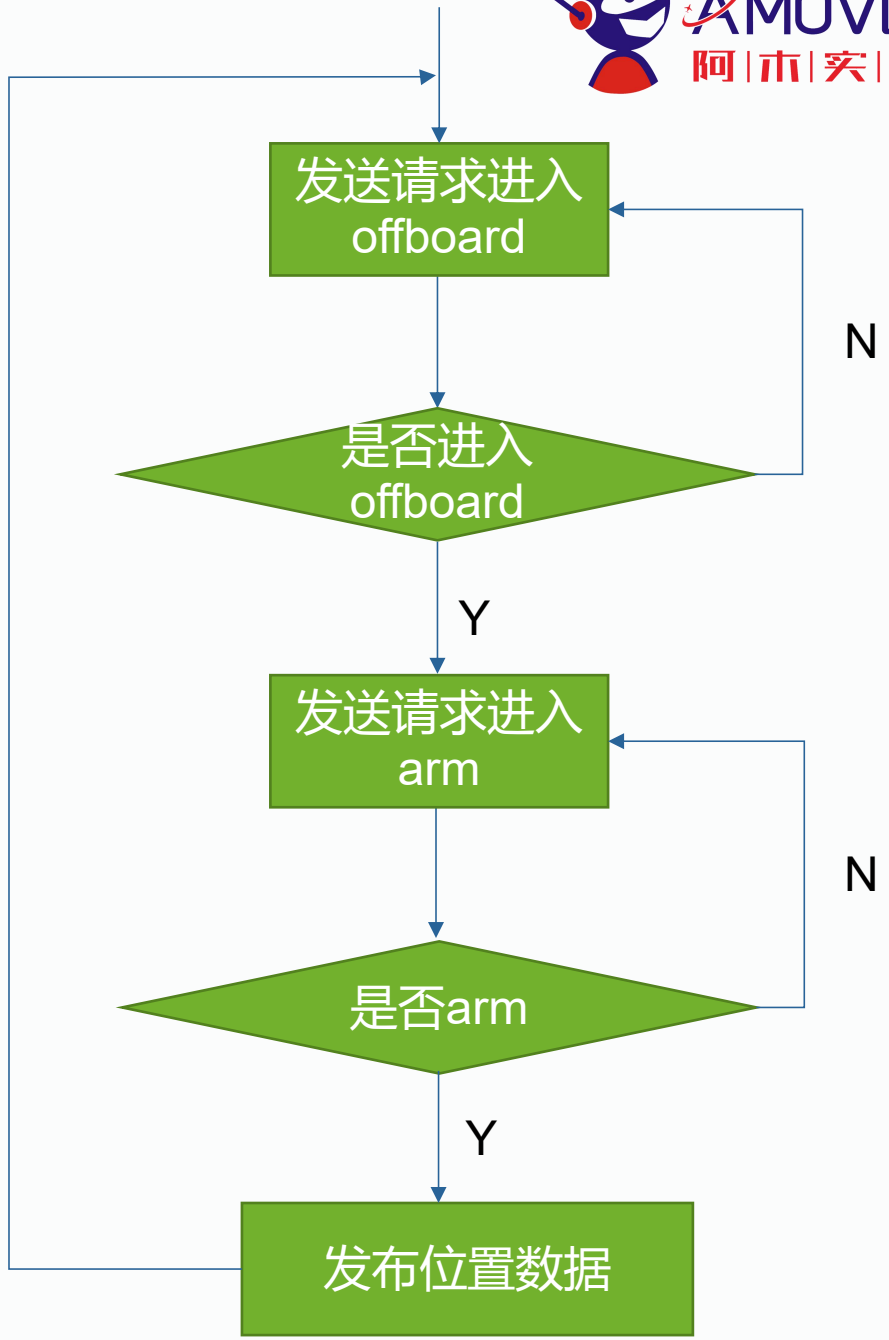
针对具体任务的功能代码段

我们将服务调用间隔 5 秒，以免请求淹没自动驾驶仪

```
while(ros::ok()){
        if( current_state.mode != "OFFBOARD" &&
            (ros::Time::now() - last_request > ros::Duration(5.0))){
            if( set_mode_client.call(offb_set_mode) &&
                offb_set_mode.response.mode_sent){
                ROS_INFO("Offboard enabled");
            }
            last_request = ros::Time::now();
        } else {
            if( !current_state.armed &&
                (ros::Time::now() - last_request > ros::Duration(5.0))){
                if( arming_client.call(arm_cmd) &&
                    arm_cmd.response.success){
                    ROS_INFO("Vehicle armed");
                }
                last_request = ros::Time::now();
            }
        }

        local_pos_pub.publish(pose);

        ros::spinOnce();
        rate.sleep();
    }
```

发送服务请求

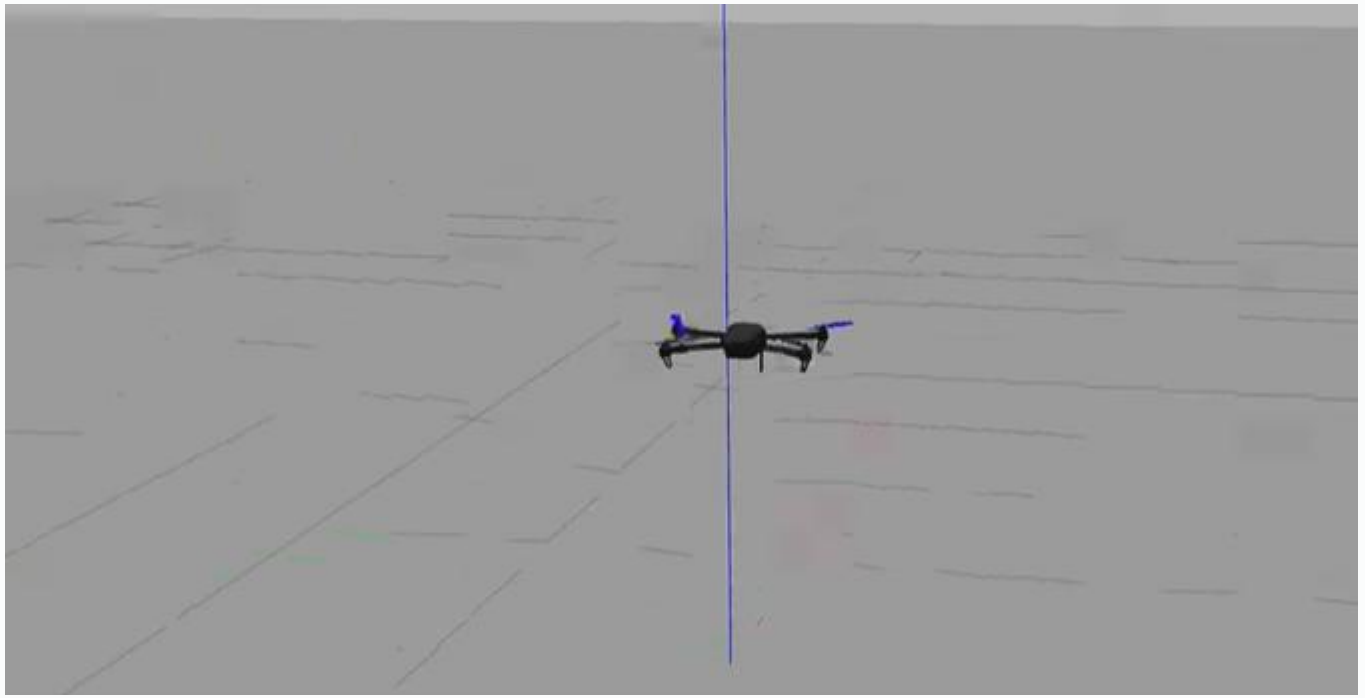
等待服务端的反馈



◆ example offboard例程的仿真与解析

Gazebo的仿真步骤：

make px4_sitl gazebo
roslaunch mavros px4.launch fcu_url:="udp://:14540@127.0.0.1:14557"
rosrun offboard_pkg offboard_node



◆ example offboard例程的仿真与设计飞行测试

硬件说明：

ZD550无人机搭载nvidia nano机载计算机，机载计算机通过usb口与PX4飞控进行通信

飞控参数的修改：

CBRK_USB_CHK→197848//取消飞行前的usb检测，如果不屏蔽的话，飞控无法进入怠速状态

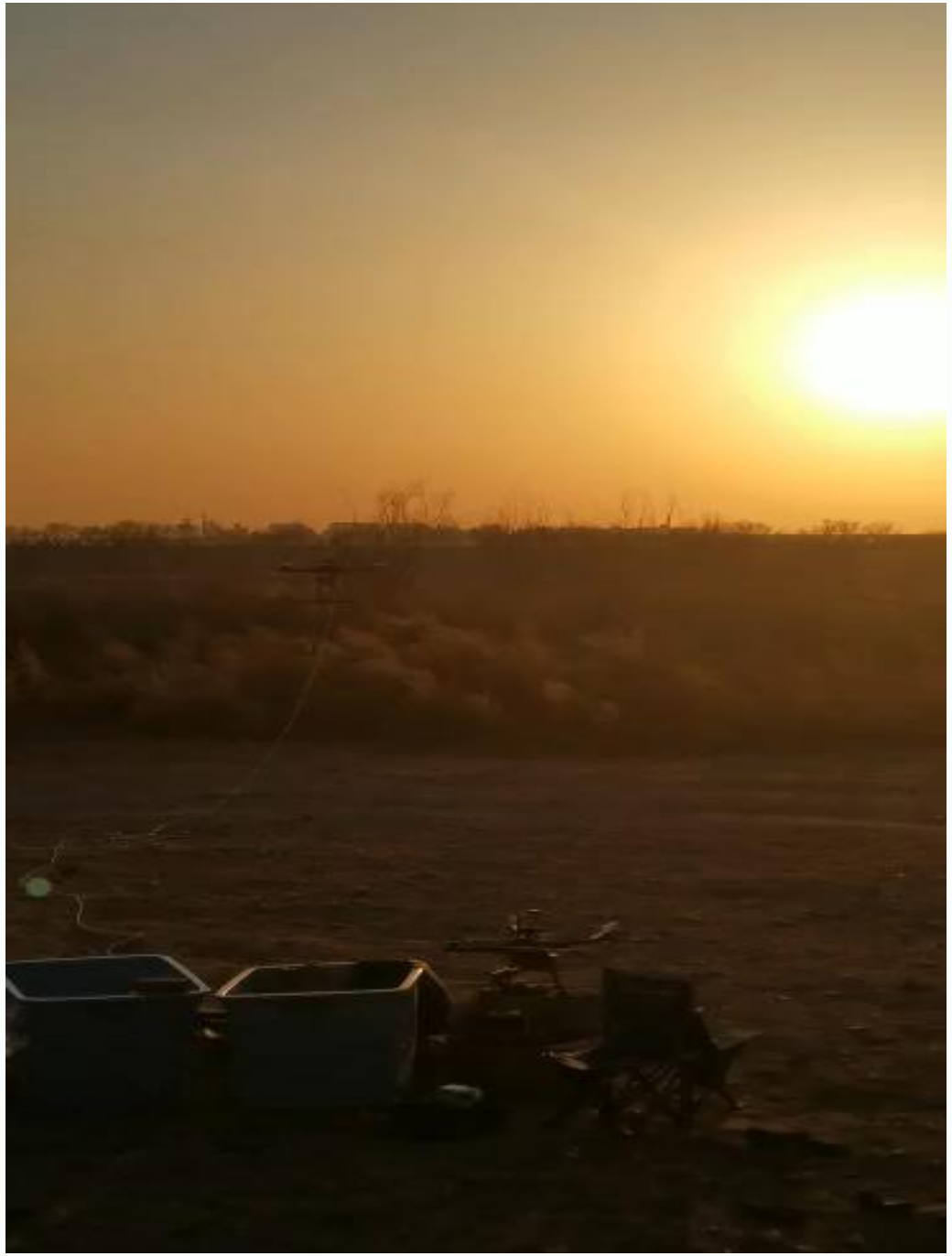
通过局域网ssh的方式，使得地面计算机与机载计算机进行通信，整个系统如下图所示：

启动mavros：

roslaunch mavros px4.launch fcu_url:=“/dev/ttyACM0“

rosrun offboard_pkg offboard_node

AMOVLAB
阿木实验室



AMOVLAB
阿木实验室

02

编程实践

绝知此事要躬行

1、mavros发布位置指令给无人机

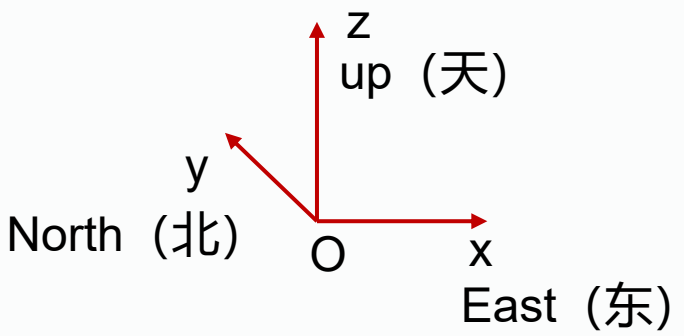
- 应用mavros去发布位置点给无人机，使得无人机完成一个飞行轨迹（正方形轨迹）
  - Mavros的坐标系
  - ROS中定时器的使用
- 应用gazebo进行软件在环仿真
- 实际的飞行测试

◆ 应用mavros去发布位置点给无人机，使得无人机完成一个飞行轨迹（正方形轨迹）

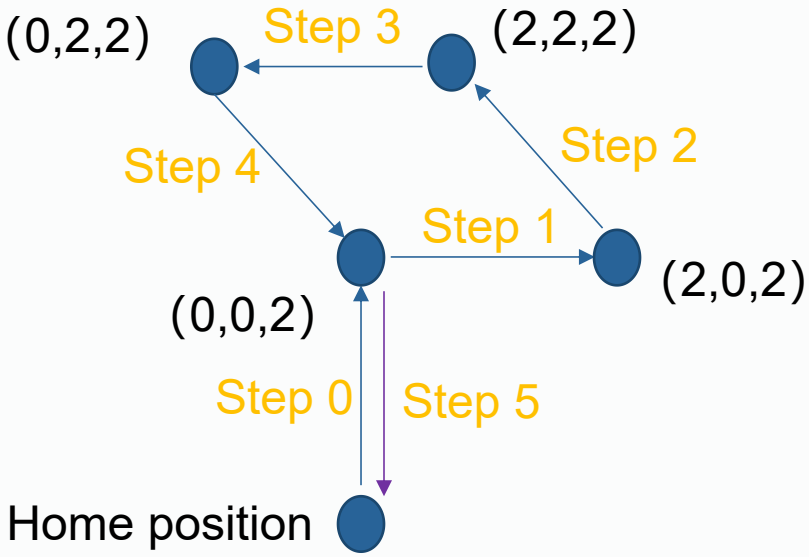
任务描述：
1. 使用机载计算机，应用mavros使得无人机在offboard模式下飞一个正方形的轨迹

起飞点：Home Position（arm时的位置）

以home position 建立ENU
（东北天）坐标系



/mavros/setpoint_position/local
/mavros/local_position/pose消息的坐标系



无人机飞一个正方形的轨迹

- 确认需要用到的mavros话题（服务）的名称及其对应的消息类型

话题/服务的名称	操作	消息类型	
/set_mode	服务的**客户端** ServiceClient	mavros_msgs/Setmode	使用机载计算机进入offboard模式，并arm无人机
/cmd/arming	服务的**客户端** ServiceClient	mavros_msgs::CommandBool	
/state	从mavros**订阅**的消息	mavros_msgs::State	
/setpoint_position/local	向mavros**发布**的消息	geometry_msgs::PoseStamped	使无人机到指定位置
/local_position/pose	从mavros**订阅**的消息	geometry_msgs::PoseStamped	获取无人机的实际位置

- 头文件需要包含所需要用到的消息类型的.h文件
- 声明发布消息，订阅消息和服务的对象，并编写相应的回调函数
- 确定节点的频率，设定为20hz

- 编写功能函数段

Step 0：起飞到（0,0,2），并判断是否到达指定位置，然后设定新的位置点为（2,0,2）
Step 2：并判断是否到达指定位置（2,0,2），然后设定新的位置点为（2,2,2）
Step 3：并判断是否到达指定位置（2,2,2），然后设定新的位置点为（0,2,2）
Step 4：并判断是否到达指定位置（0,2,2），然后设定新的位置点为（0,0,2）
Step 5：并判断是否到达指定位置（0,0,2），然后切换无人机模式为降落模式回到home position

使用变量step记录飞行的阶段，用switch.. case..语句判断飞行阶段，在case段中实现具体的功能代码：

```
//进入降落模式代码段
offb_set_mode.request.custom_mode = "AUTO.LAND";
if (current_state.mode != "AUTO.LAND" && (ros::Time::now() - last_request > ros::Duration(5.0)))
{

        if (set_mode_client.call(offb_set_mode) && offb_set_mode.response.mode_sent)
        {
                ROS_INFO("AUTO.LAND enabled");
        }
        last_request = ros::Time::now();
}
```

```
class offboard_class_timer
{
private:
    ros::Publisher local_pos_pub;
    ros::Subscriber local_pos_sub ;
    ros::Subscriber state_sub ;
    ros::ServiceClient arming_client;
    ros::ServiceClient set_mode_client;

    ros::NodeHandle nh;//we will need this, to pass between "main" and constructor
    void init_publisher();
    void init_subscriber();
    void init_service();
    void local_pos_cb(const geometry_msgs::PoseStamped::ConstPtr& msg);
    void state_cb(const mavros_msgs::State::ConstPtr& msg);

public:
    offboard_class_timer(ros::NodeHandle* nodehandle);
    ~offboard_class_timer();

    ros::Timer calc_timer;
    void calc_cb(const ros::TimerEvent&);
   geometry_msgs::PoseStamped pose;
    mavros_msgs::State current_state;
    geometry_msgs::PoseStamped local_pos;
    mavros_msgs::SetMode offb_set_mode;
    mavros_msgs::CommandBool arm_cmd;
    ros::Time last_request;
    int step = 0;
                int sametimes = 0;
};
```

◆ 在ROS中使用C++的类并使用定时器改写上述代码

1. 将原有程序中声明的全局变量和main函数中的局部变量都在类中进行定义。
2. 依据变量是否需要在类之外的函数进行使用将变量划分为private和public两类。在本示例中，均可以。
3. 声明定时器以及定时器的回调函数

定时器的使用：

本示例中在类的构造函数中定义并开启定时器。

```
//定时器的定义
ros::Timer calc_timer;
//定时器回调函数的定义
void calc_cb(const ros::TimerEvent&);
```

```
//在构造函数中创建并开启定时器
offboard_class_timer::offboard_class_timer(ros::NodeHandle* nodehandle):nh(*nodehandle)
{
    calc_timer = nh.createTimer(ros::Duration(0.05), &offboard_class_timer::calc_cb, this);
}
```

定时器时间间隔，单位是秒，0.05秒即为20hz

定时器的回调函数

在类中使用的话需要加上this

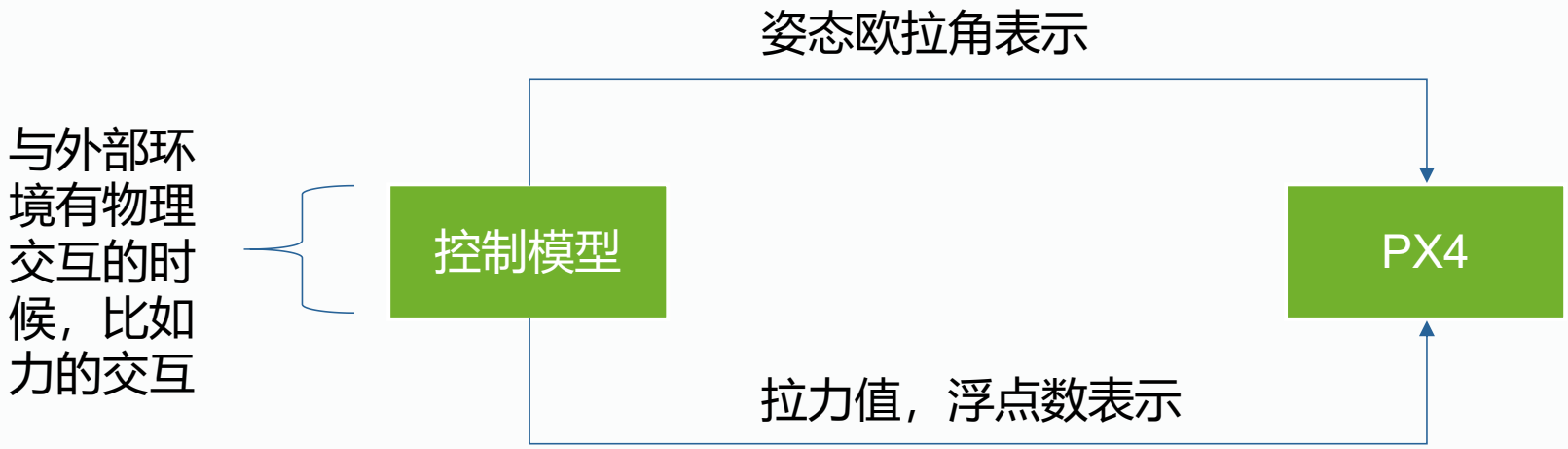
在定时器的回调函数中，编写自己的功能代码段

- Gazebo软件在环仿真
- 实际飞行测试

2、mavros发布姿态指令给无人机

- 应用mavros去发布姿态指令给无人机，使得无人机完成一个调整偏航角的动作
  - Mavros姿态指令的坐标系，以及坐标系变换
  - 欧拉角到四元数的变换
  - 归一化拉力的意义
- 应用gazebo进行软件在环仿真
- 实际的飞行测试

问题描述：



这个时候，我们就需要用mavros发布姿态的指令给无人机，我们不知道如何去发布，百度之后也没有现成的教程，这时候就需要之前讲过的，去阅读mavros的wiki文档，或者源码，去看看有没有现成的消息能够完成这个工作。

阅读mavros的wiki文档

6.14 setpoint_attitude

Send attitude setpoint using SET_ATTITUDE_TARGET.

6.14.1 Subscribed Topics

`~setpoint_attitude/cmd_vel` (geometry_msgs/TwistStamped)
Send angular velocity.

`~setpoint_attitude/attitude` (geometry_msgs/PoseStamped)
Send attitude setpoint.

`~setpoint_attitude/thrust` (mavros_msgs/Thrust)
Send throttle value(0~1).

6.16 setpoint_raw

Send RAW setpoint messages to FCU and provide loopback topics (PX4).

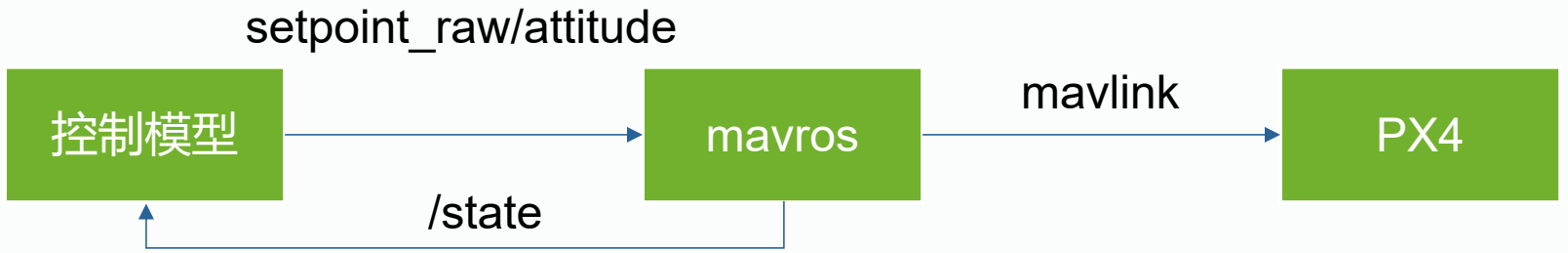
6.16.1 Subscribed Topics

`~setpoint_raw/local` (mavros_msgs/PositionTarget)
Local position, velocity and acceleration setpoint.

`~setpoint_raw/global` (mavros_msgs/GlobalPositionTarget)
Global position, velocity and acceleration setpoint.

`~setpoint_raw/attitude` (mavros_msgs/AttitudeTarget)
Attitude, angular rate and thrust setpoint.

通过试验可以很快的得到验证。这个消息实际上满足我们的要求。



- 确认需要用到的mavros话题（服务）的名称及其对应的消息类型

话题/服务的名称	操作	消息类型
/set_mode	服务的**客户端** ServiceClient	mavros_msgs/Setmode
/cmd/arming	服务的**客户端** ServiceClient	mavros_msgs::CommandBool
/state	从mavros**订阅**的消息	mavros_msgs::State
/setpoint_raw/attitude	向mavros**发布**的消息	mavros_msgs/AttitudeTarget

常规的

着重看一下消息类型

mavros_msgs/AttitudeTarget

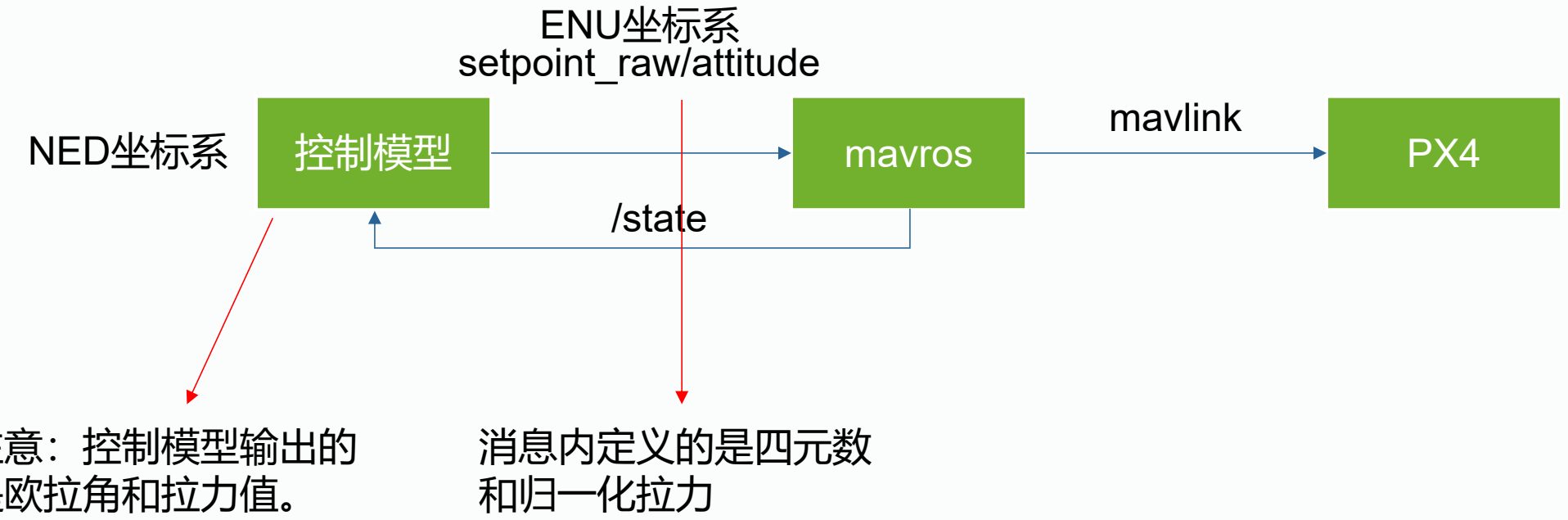
http://docs.ros.org/en/api/mavros_msgs/html/msg/AttitudeTarget.html

```
# Message for SET_ATTITUDE_TARGET
#
# Some complex system requires all feautures that mavlink
# message provide. See issue #402, #418.

std_msgs/Header header

uint8 type_mask
uint8 IGNORE_ROLL_RATE = 1 # body_rate.x
uint8 IGNORE_PITCH_RATE = 2 # body_rate.y
uint8 IGNORE_YAW_RATE = 4 # body_rate.z
uint8 IGNORE_THRUST = 64
uint8 IGNORE_ATTITUDE = 128 # orientation field

geometry_msgs/Quaternion orientation
geometry_msgs/Vector3 body_rate
float32 thrust
```

若干掩码设置，满足不同的需求，比如你需不要设置偏航角速度，则可以将：IGNORE_YAW_RATE = 4

订阅的是四元数数据，因此我们发布的也应该是四元数数据。

归一化的拉力值，范围在0~1之间。0对应的就是PWM_MIN，1对应的就是PWM_MAX。



需要做以下三个转换：NED转ENU、欧拉角转四元数、拉力值转归一化拉力值

- NED转ENU

```
geometry_msgs::Point ned2enu_angle(geometry_msgs::Point ned_angle)
{
    geometry_msgs::Point enu_angle;
    enu_angle.x = ned_angle.x;
    enu_angle.y = -ned_angle.y;
    enu_angle.z = -ned_angle.z + PI/2;
    return enu_angle;
}
```

- 欧拉角转四元数

```
geometry_msgs::Quaternion euler_to_quat(geometry_msgs::Point euler_angle_enu)
{
    tfScalar yaw,pitch,roll;
    tf::Quaternion q;
    geometry_msgs::Quaternion quat;

    roll = euler_angle_enu.x;
    pitch = euler_angle_enu.y;
    yaw = euler_angle_enu.z;

    q.setEulerZYX(yaw,pitch,roll);
    quat.x = q.x();
    quat.y = q.y();
    quat.z = q.z();
    quat.w = q.w();
    return quat;
}
```

- 拉力值转归一化拉力值

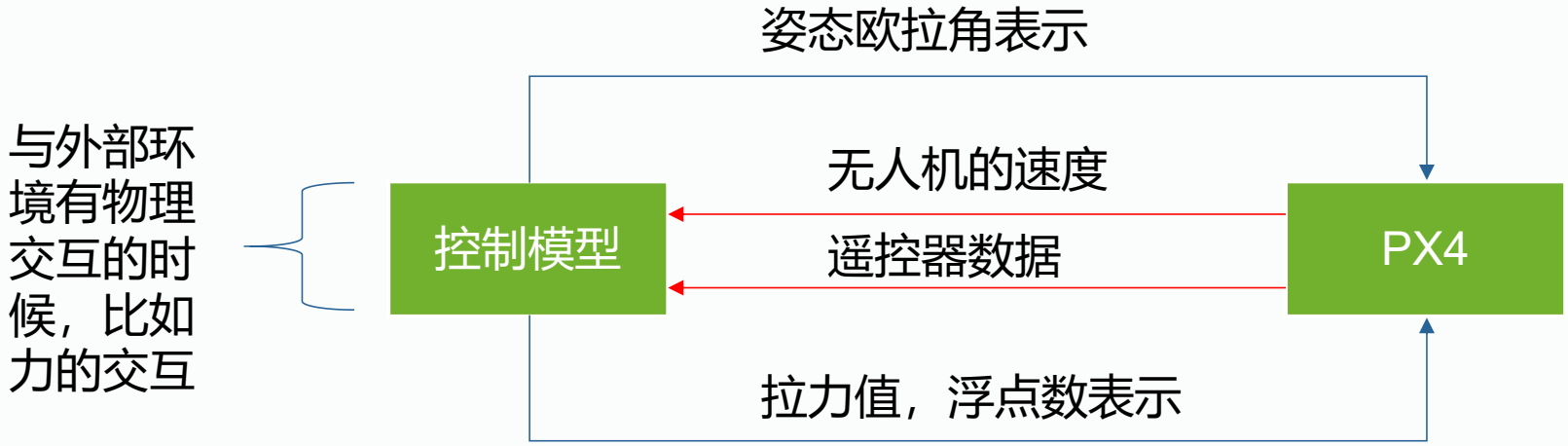
- 测量得到无人机的最大拉力，有专门的仪器可以得到
- 拉力值/最大拉力值=归一化拉力值

```
float THRUST_FULL=67.7;
float thrust_scale(float thrust)
{
        float thrust_scale = 0.0;
        if (thrust>=THRUST_FULL)
        {
                /* code */
                thrust= THRUST_FULL;
        }
         thrust_scale = thrust/ THRUST_FULL;
}
```

3、mavros订阅无人机传感器数据

- 编写程序，应用mavros去订阅无人机传感器数据
  - 获取遥控器的数据
  - 线速度，角速度的坐标系，及其坐标系转换
  - 调整无人机反馈回来的传感器数据的发送频率
- 直接连接px4进行试验验证

问题描述：



1、查阅mavros的文档，找到相关的话题名称
2、阅读源码
在mavros的文档中，并没有包含所有的mavros可订阅/发布的话题名称

所需要用到的消息：

1. 订阅得到遥控器的数据：/mavros/rc/in

~rc/in (mavros_msgs/RCIn)

Publish RC inputs (in raw milliseconds)

数据类型

mavros_msgs/RCIn Message

File: mavros_msgs/RCIn.msg

Raw Message Definition

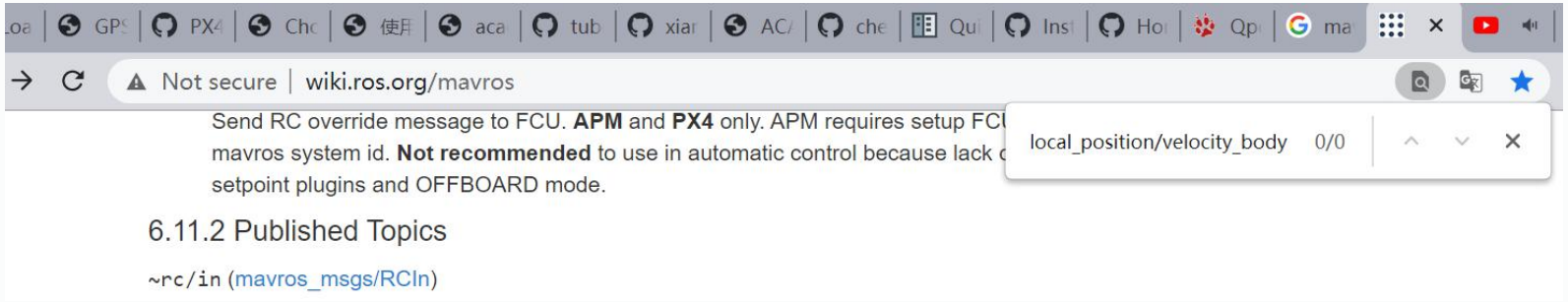
```
# RAW RC input state

std_msgs/Header header
uint8 rssi
uint16[] channels
```

Compact Message Definition

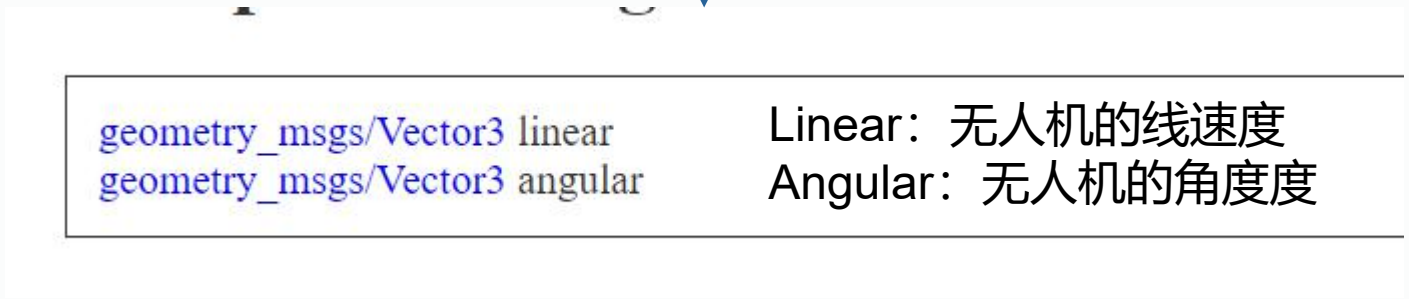
```
std_msgs/Header header
uint8 rssi
uint16[] channels
```

用一个数组来存储获得的遥控器各个通道的数据

2. 订阅得到无人机的速度数据：/mavros/local_position/velocity_body



数据类型：



Notes：

Mavros反馈回来的速度数据的坐标系为北西天坐标系，不是位置数据的ENU坐标系。

修改mavros反馈传感器数据的频率

默认情况下，mavros反馈**位姿和速度的传感器数据**频率为**30hz**

可用mavcmd 进行修改

```
rosrun mavros mavcmd long 511 32 20000 0 0 0 0 0
```

修改消息的间隔时间的命令

ATTITUDE_QUATERNION (#31)

[Message] The attitude in the aeronautical frame (right-handed, Z-down, X-front, Y-right), expressed as q
Quaternion order is w, x, y, z and a zero rotation would be expressed as (1 0 0 0).

Field Name	Type	Units	Description
time_boot_ms	uint32_t	ms	Timestamp (time since system boot).
q1	float		Quaternion component 1, w (1 in null-rotation)
q2	float		Quaternion component 2, x (0 in null-rotation)
q3	float		Quaternion component 3, y (0 in null-rotation)
q4	float		Quaternion component 4, z (0 in null-rotation)
rollspeed	float	rad/s	Roll angular speed
pitchspeed	float	rad/s	Pitch angular speed
yawspeed	float	rad/s	Yaw angular speed

LOCAL_POSITION_NED (#32)

[Message] The filtered local position (e.g. fused computer vision and accelerometers). Coordinate frame is right-handed, Z-axis down (aeronautical frame, NED / north-east-down convention)

Field Name	Type	Units	Description
time_boot_ms	uint32_t	ms	Timestamp (time since system boot).
x	float	m	X Position
y	float	m	Y Position
z	float	m	Z Position
vx	float	m/s	X Speed
vy	float	m/s	Y Speed
vz	float	m/s	Z Speed

MAV_CMD_SET_MESSAGE_INTERVAL (511)

[Command] Set the interval between messages for a particular MAVLink message ID. This interface replaces REQUEST_DATA_STREAM.

Param (:Label)	Description	Values	Units
1: Message ID	The MAVLink message ID	*min:*0 *max:*16777215 *increment:*1	
2: Interval	The interval between two messages. Set to -1 to disable and 0 to request default rate.	*min:* -1 *increment:*1	us
7: Response Target	Target address of message stream (if message has target address fields). 0: Flight-stack default (recommended), 1: address of requestor, 2: broadcast.	*min:*0 *max:*2 *increment:*1	

话题所对应的mavlink ID

修改的时间间隔单位是us

目标地址，在我们的应用中直接置零就可以了

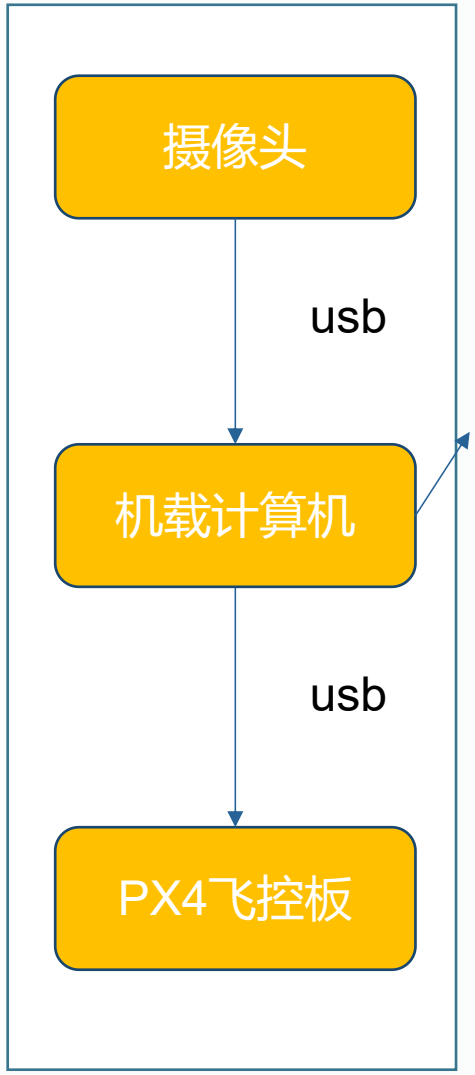
连接PX4飞控进行验证

4、mavros与SLAM

- 应用mavros将由SLAM、室内定位系统获取得到的位置数据发送给无人机
  - 对应的软硬件架构说明
  - 对PX4飞控的配置
  - 应用到的mavros消息
- 实际飞行测试

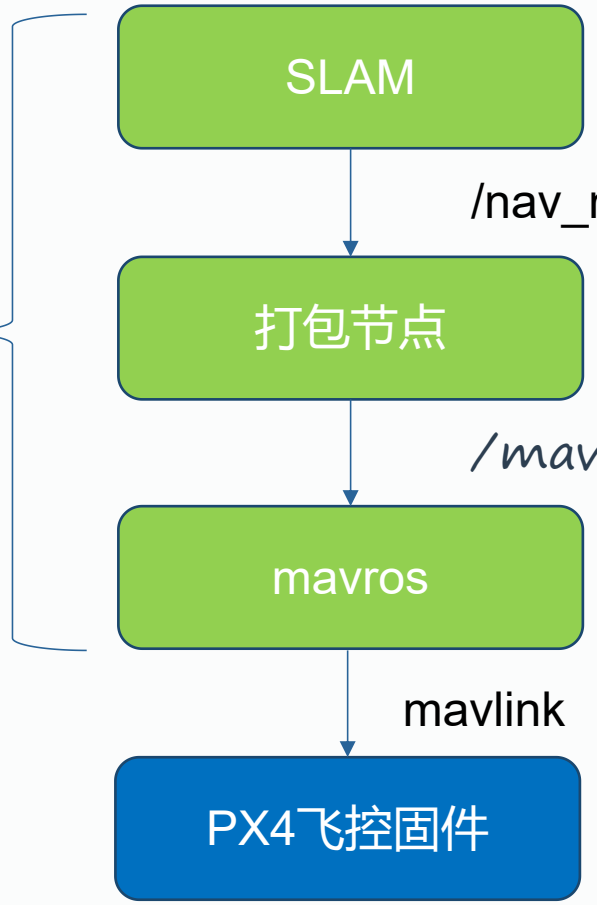
应用SLAM获取无人机的位置数据：

硬件架构

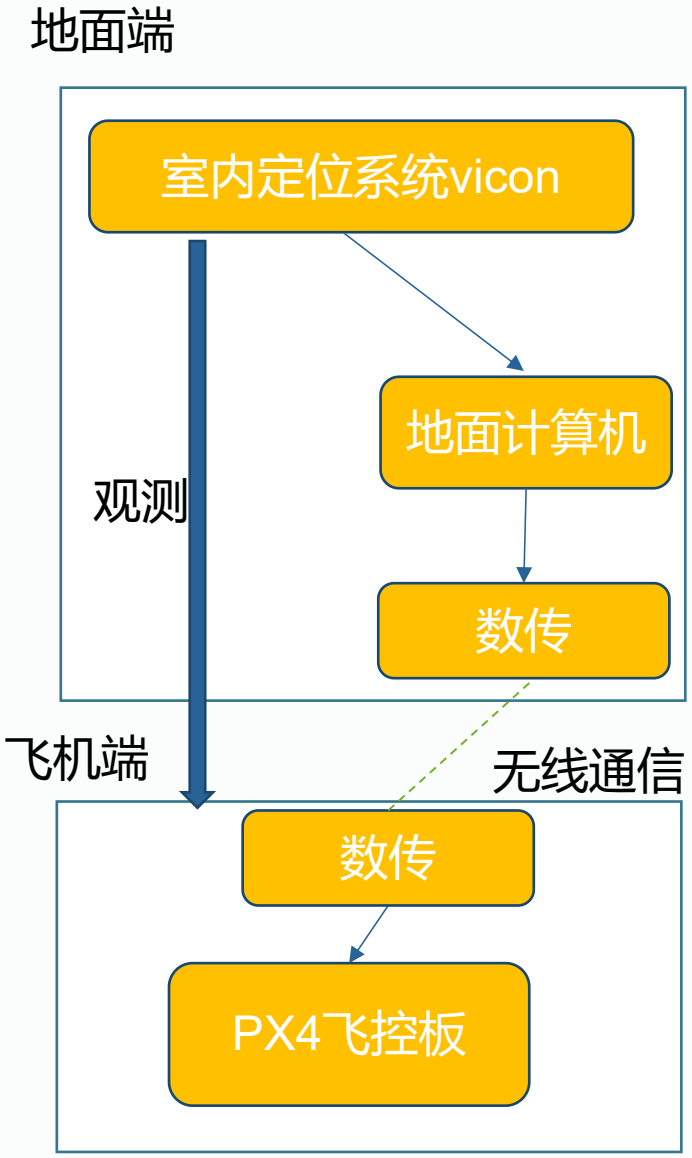


运行在机载电脑端的ros系统上

软件架构

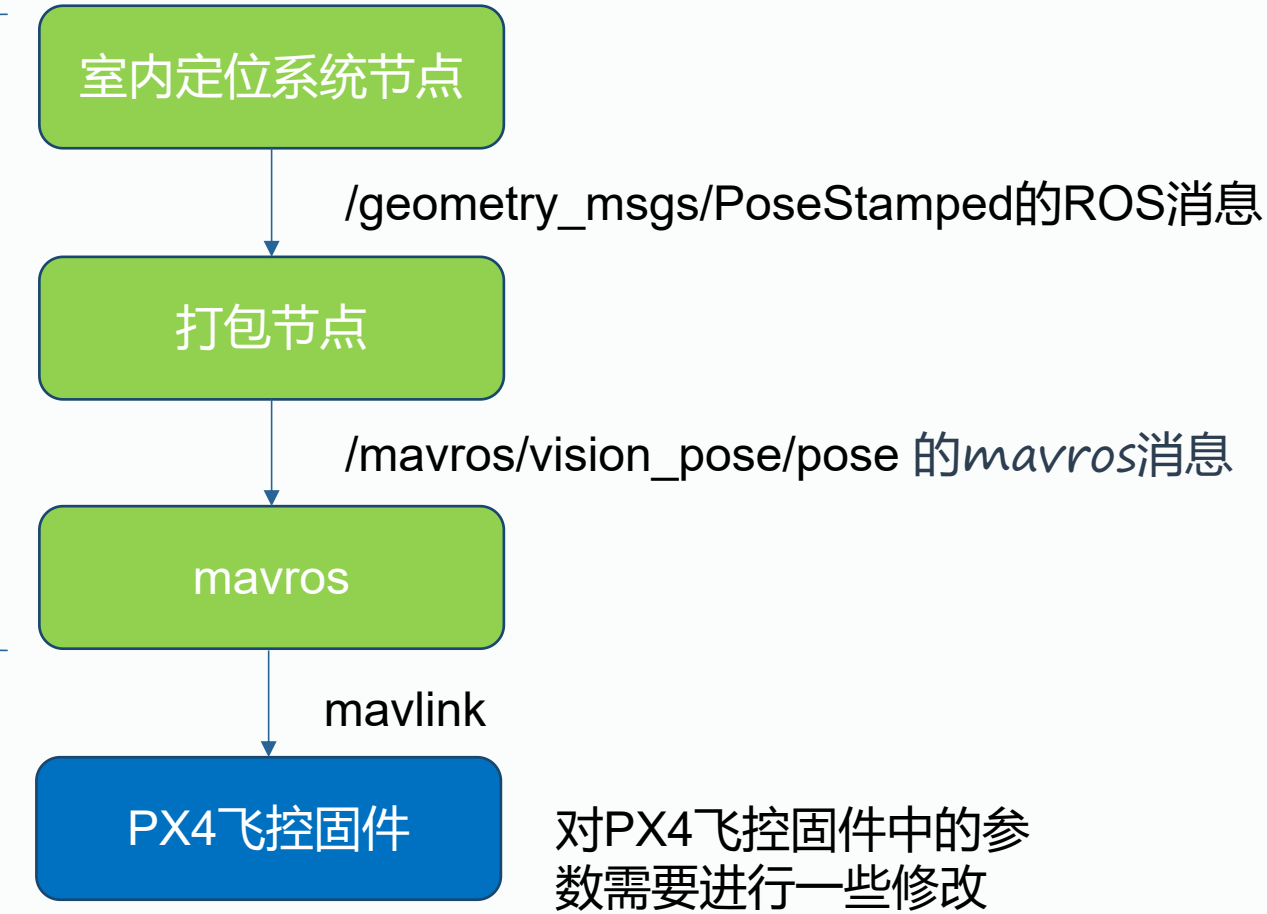


应用室内定位系统（vicon）获取无人机的位置数据：



软件架构

运行在地面计算机端的ros系统上



对PX4飞控固件中的参数需要进行一些修改

https://docs.px4.io/master/en/advanced_config/tuning_the_ecl_ekf.html#external-vision-system

1.使能额外定位系统的垂直方向位置数据的获取：

EKF2_HGT_MODE → 3

2 .配置其他位置数据相关的参数：

EKF2_AID_MASK → 依据不同的需求进行不同的配置

https://docs.px4.io/master/en/advanced_config/parameter_reference.html#EKF2_AID_MASK

EKF2_AID_MASK的详细解释

EKF_AID_MASK value	Set bits	Description
321	GPS + EV_VEL + ROTATE_EV	Heading w.r.t. North (**Recommended**)
73	GPS + EV_POS + ROTATE_EV	Heading w.r.t. North (*Not recommended*, use `EV_VEL` instead)
24	EV_POS + EV_YAW	Heading w.r.t. external vision frame
72	EV_POS + ROTATE_EV	Heading w.r.t. North
272	EV_VEL + EV_YAW	Heading w.r.t. external vision frame
320	EV_VEL + ROTATE_EV	Heading w.r.t. North
280	EV_POS + EV_VEL + EV_YAW	Heading w.r.t. external vision frame
328	EV_POS + EV_VEL + ROTATE_EV	Heading w.r.t. North

坐标系

Reference Frames and ROS

The local/world and world frames used by ROS and PX4 are different.

Frame	PX4	ROS
Body	FRD (X Forward, Y Right, Z Down)	FLU (X Forward, Y Left, Z Up), usually named `base_link`
World	FRD or NED (X North, Y East, Z Down)	FLU or ENU (X East, Y North, Z Up), with the naming being `odom` or `map`

频率

对于PX4飞控来说，在官方示例代码中由说明位置数据要高于2hz，但是我们实际测试的过程中发现，当SLAM的位置数据发布频率小于10Hz在的时候，PX4无法进入位置模式。

应用到的mavros消息

https://docs.px4.io/master/en/ros/external_position_estimation.html

ROS	MAVLink	uORB
/mavros/vision_pose/pose	VISION_POSITION_ESTIMATE	vehicle_visual_odometry
/mavros/odometry/out	ODOMETRY	vehicle_visual_odometry
/mavros/mocap/pose	ATT_POS_MOCAP	vehicle_mocap_odometry
/mavros/odometry/out	ODOMETRY	vehicle_mocap_odometry

如果用的是EKF，只用前两个消息是可以被使用的，究竟使用哪个，由视觉定位系统获得得到的消息类型而定：

- geometry_msgs/PoseStamped 或者 geometry_msgs/PoseWithCovarianceStamped必须是/mavros/vision_pose/pose
- nav_msgs/Odometry 则为/mavros/odometry/out

实际的飞行测试

基于ORB_SLAM的实际飞行测试
机载电脑为：nvidia NX

AMOVLAB
阿木实验室

03

总结引申

君子博学而日参省乎己

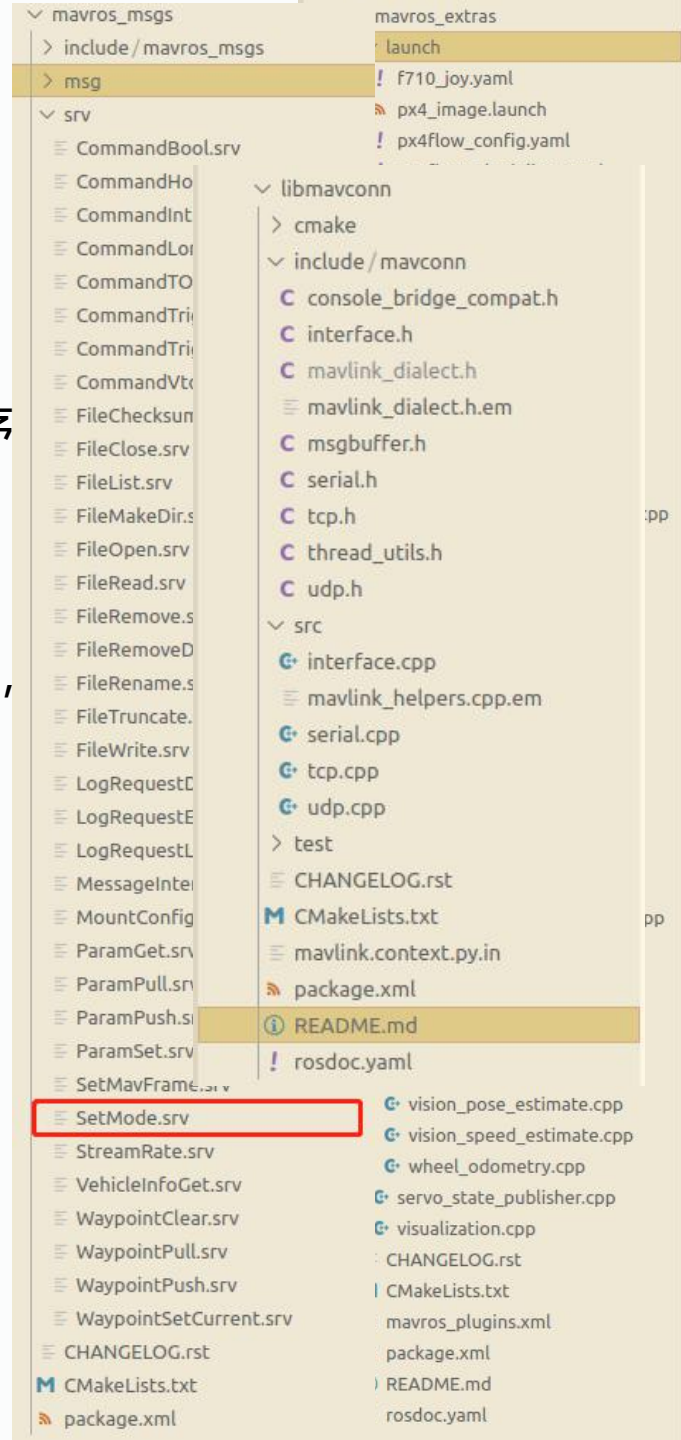
1、mavros代码分析

- mavros整体代码结构分析
- mavros主功能包代码分析
  - 以setpoint_raw.cpp为例分析

◆ mavros整体代码结构分析

- mavros
  - mavros：主功能包
    - Include
      - px4.launch
      - px4_pluginlists.yaml
      - px4_config.yaml
    - launch
    - script
      - mavcmd等脚本文件
    - src
      - mavros_node.cpp 主节点程序
      - gcs_bridge.cpp 添加信道
      - plugins
  - libmavconn：通信功能包（mavconn）
  - mavros_extras：mavros额外的节点，eg:slam视觉相关的plugin
  - mavros_msgs: mavros中定义的消息和服务
  - test_mavros：使用SITL进行手工测试

实现mavros各个功能的程序，每一个程序可以看做是一个独立的线程



绿色的框均可以看做是一个ros的节点编程，其中的主要文件架构与我们之前自己编写的程序是一样的。所以理解ros编程，对于mavros源码的阅读是有帮助的。

◆ mavros主功能包代码分析

从主节点开始分析，mavros_node.cpp

```
#include <mavros/mavros.h>

int main(int argc, char *argv[])
{
        ros::init(argc, argv, "mavros");

        mavros::MavRos mavros;  //实例化一个MavRos的类，在类的构造函数中，启动所有的plugin的代码

        mavros.spin();

        return 0;
}
```

◆ 以setpoint_raw.cpp为例分析

1. initialize（）函数

```
void initialize(UAS &uas_) override
{
        PluginBase::initialize(uas_);

        bool tf_listen;

        local_sub = sp_nh.subscribe("local", 10, &SetpointRawPlugin::local_cb, this);
        global_sub = sp_nh.subscribe("global", 10, &SetpointRawPlugin::global_cb, this);
        attitude_sub = sp_nh.subscribe("attitude", 10, &SetpointRawPlugin::attitude_cb, this);
        target_local_pub = sp_nh.advertise<mavros_msgs::PositionTarget>("target_local", 10);
        target_global_pub = sp_nh.advertise<mavros_msgs::GlobalPositionTarget>("target_global", 10);
        target_attitude_pub = sp_nh.advertise<mavros_msgs::AttitudeTarget>("target_attitude", 10);
}
```

2. 跳转到attitude_cb（）回调函数

```
desired_orientation = ftf::to_eigen(req->orientation);

// Transform desired orientation to represent aircraft->NED,
// MAVROS operates on orientation of base_link->ENU
auto ned_desired_orientation = ftf::transform_orientation_enu_ned(
        ftf::transform_orientation_baselink_aircraft(desired_orientation));

body_rate = ftf::transform_frame_baselink_aircraft(
        ftf::to_eigen(req->body_rate));

set_attitude_target(
        req->header.stamp.toNSec() / 1000000,
        req->type_mask,
        ned_desired_orientation,
        body_rate,
        thrust);
```

进行姿态的坐标系转换，由ENU到NED

赋角速度值

mavlink协议打包

3. 跳转到set_attitude_target()函数

```
mavros::UAS *m_uas_ = static_cast<D *>(this)->m_uas;
mavlink::common::msg::SET_ATTITUDE_TARGET sp = {};

m_uas_->msg_set_target(sp);
mavros::ftf::quaternion_to_mavlink(orientation, sp.q);

sp.time_boot_ms = time_boot_ms;
sp.type_mask = type_mask;
sp.thrust = thrust;
sp.body_roll_rate = body_rate.x();
sp.body_pitch_rate = body_rate.y();
sp.body_yaw_rate = body_rate.z();

UAS_FCU(m_uas_)->send_message_ignore_drop(sp);
```

mavlink协议的消息名称

发送mavlink协议数据

2、总结

mavros是mavlink协议在ros系统上的封装

- ros
  1. 理解ros的文件架构
  2. 要多使用C++的类进行编程
  3. 分析问题，划分ros子节点
- mavros
  1. 分析问题，寻找能够相应的mavros消息，并单独验证
  2. 列出所有的消息，根据需求确定频率
  3. 按步骤完成程序的编写
  - wiki
    - 试验
      1. 运行mavros
      2. rostopic list可以查询所有的mavros消息
      3. 依据消息名称，试验各个消息的功能
    - 阅读源码
- mavlink
  1. mavros消息最终还是要归结到mavlink协议上
  2. 在拿不准消息究竟是实现什么功能的时候，要通过其对应的mavlink协议去明确。

AMOVLAB
阿|木|实|验|室

THANKS