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

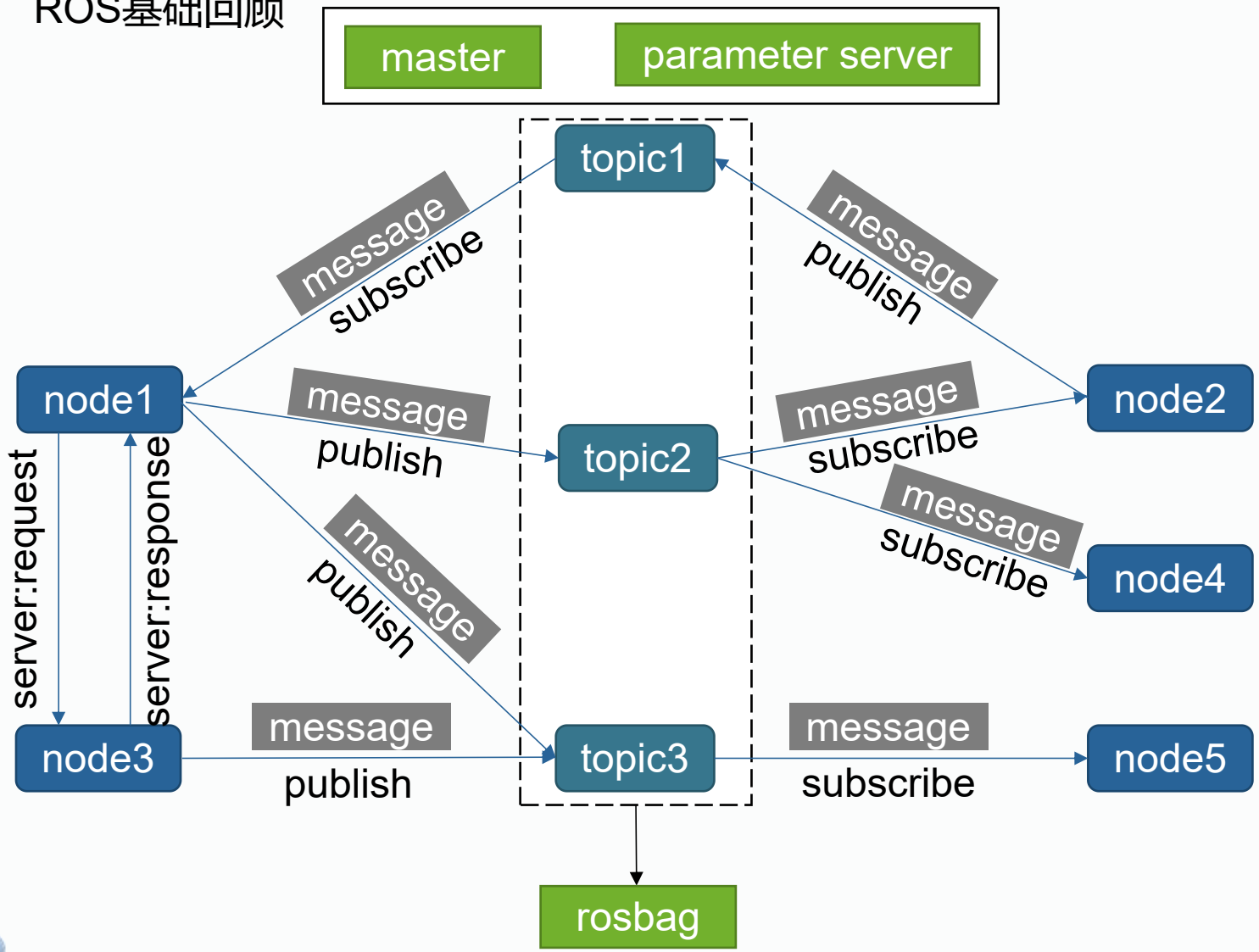
- 本课程偏重于工程实践，通过一个个实践项目，帮助大家熟悉mavros的使用。所涉及的面比较广，基本上涵盖了mavros常用的应用场景。

- 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。

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
```

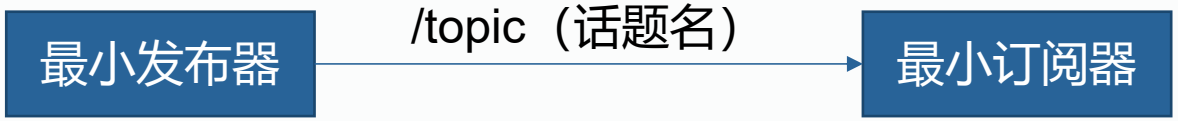
3、ROS编程基础

- 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
```

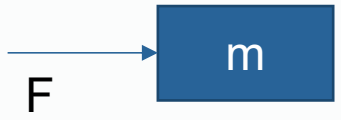
```
average: 8.50B/s
        mean: 8.00B min: 8.00B max: 8.00B window: 11
average: 8.46B/s
        mean: 8.00B min: 8.00B max: 8.00B window: 12
average: 8.42B/s
        mean: 8.00B min: 8.00B max: 8.00B window: 13
average: 8.39B/s
        mean: 8.00B min: 8.00B max: 8.00B window: 14
average: 8.36B/s
        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
```

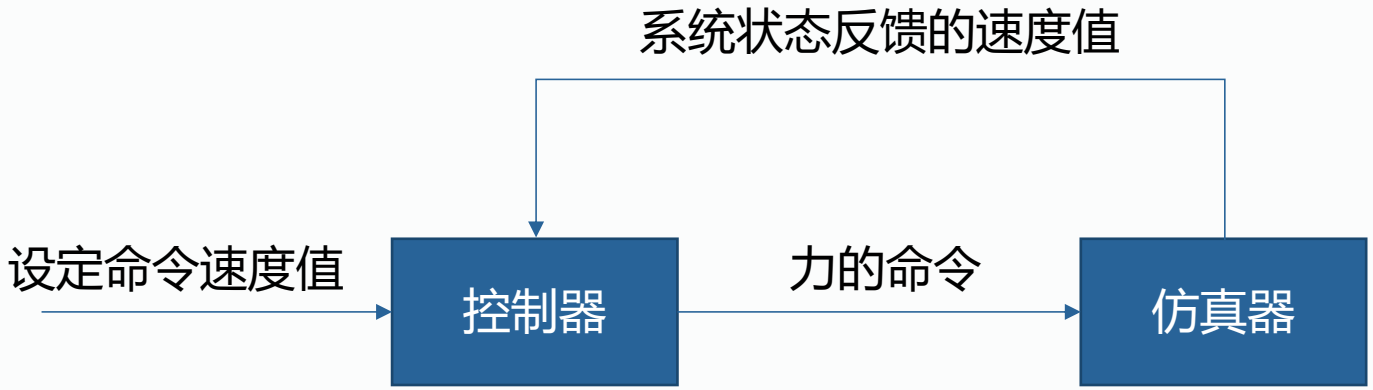
```
[ INFO] [1625501407.179106889]: received value is: 2.650000
[ INFO] [1625501408.178680782]: received value is: 2.651000
[ INFO] [1625501409.178687688]: received value is: 2.652000
[ INFO] [1625501410.179023002]: received value is: 2.653000
[ INFO] [1625501411.178122712]: received value is: 2.654000
[ 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：

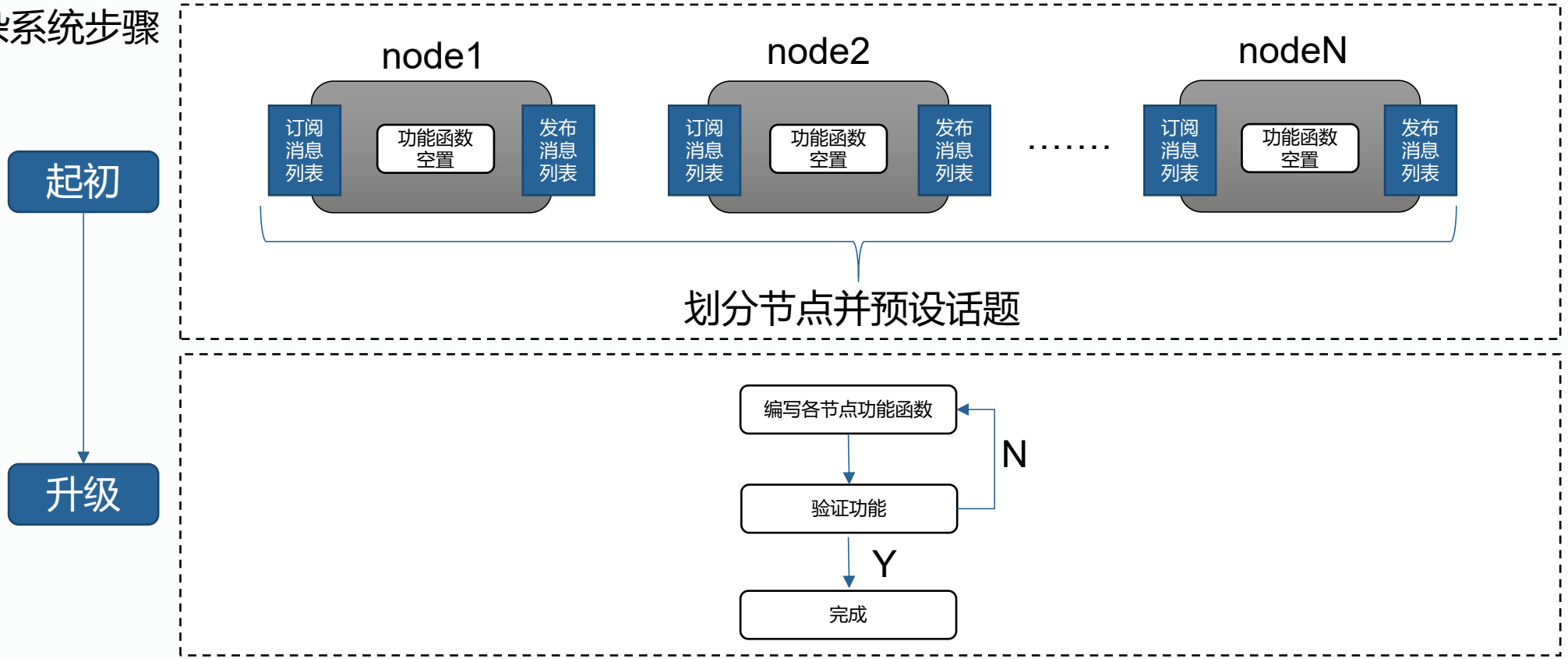
rosrun minimal_nodes minimal_simulator

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。

AMOVLAB
阿木实验室

02

编程实践

绝知此事要躬行

AMOVLAB
阿木实验室

03

总结引申

君子博学而日参省乎己

AMOVLAB
阿|木|实|验|室

THANKS