ROS机器人操作系统

·WHAT：ROS是什么？

ROS (Robot Operating System)是2007年诞生于斯坦福大学的一个机器人操作系统，原生测试系统为Linux Ubuntu。作为机器人软件的元操作系统，ROS主要提供了四个方面的支持，通信机制、开发工具、应用软件以及生态系统（社区支持）。

·WHY：为什么要用ROS？

ROS控制系统的建构基于逻辑简明的计算图和模块化的节点，有丰富的社区资源和软件、工具支持，对于开发大型的机器人项目是很友好的。

撇开ROS的各种好处，最直接的原因就是无人机的Offboard离线控制算法运行在ROS系统上，通过MAVROS软件包给飞控模块发送位置、速度等控制命令。



·HOW：怎么使用ROS？

这是这一篇文章需要解决的问题。

ROS的程序开发需要扎实的C++、Python语言基础和一定的机器人学知识；但只要熟悉一些基本的概念和命令，就能开始尝试对其他ROS项目做一些简单的修改了。

·ROS & ROS2：如何选择？

ROS2是新一代的ROS系统，2022年发布第一个长期支持版，其底层架构全部重新设计，因此与第一代ROS是不兼容的。在可见的未来，ROS将会被ROS2取代。

目前推荐先学习ROS，再学习ROS2，这是因为ROS的社区支持、教程和软件支持都多于ROS2，而且两者大部分基本概念都是相似的。

目录

[1. 系统安装 Installation 4](#_Toc159688484)

[1.1. 版本选择 4](#_Toc159688485)

[1.2. 下载安装 4](#_Toc159688486)

[1.3. 主文件访问 5](#_Toc159688487)

[1.4. 卸载 5](#_Toc159688488)

[2. 基本概念 Basic Concepts 5](#_Toc159688489)

[2.1. 节点管理器Master 6](#_Toc159688493)

[2.2. 参数服务器Parameter Server 6](#_Toc159688494)

[2.3. 节点Node 6](#_Toc159688495)

[2.4. 话题与消息Topic & Message(msg) 7](#_Toc159688496)

[2.5. 服务 Service 7](#_Toc159688497)

[3. 编程与开发：工作空间 Workspace 7](#_Toc159688498)

[3.1. 工作空间：创建与初始化 8](#_Toc159688500)

[3.2. 创建软件包 8](#_Toc159688501)

[3.3. 编译C++文件 9](#_Toc159688502)

[3.4. 编写消息和服务类型定义文件.msg及.srv 9](#_Toc159688503)

[3.4.1. 基本语法 9](#_Toc159688511)

[3.4.2. packages.xml新增依赖项 10](#_Toc159688512)

[3.4.3. 修改CMakeLists.txt 11](#_Toc159688513)

[3.5. 编写启动脚本文件.launch (.xml) 11](#_Toc159688514)

[3.6. Python文件 12](#_Toc159688515)

[3.7. 插件 Plugin 12](#_Toc159688516)

[4. 编程与开发: C++ Coding and Developing 12](#_Toc159688517)

[4.1. 命名规范 12](#_Toc159688519)

[4.2. 节点的初始化、启动与关闭 12](#_Toc159688520)

[4.3. 导入消息和服务类型 13](#_Toc159688521)

[4.4. 话题：发布者与订阅者的创建 13](#_Toc159688522)

[4.5. 服务：服务器与客户端的创建 14](#_Toc159688523)

[4.6. 参数服务器：参数的创建与获取 14](#_Toc159688524)

[4.7. 回调与旋回：ros::spin()、ros::spinOnce() 14](#_Toc159688525)

[4.8. 定频循环：ros::Rate 15](#_Toc159688526)

[4.9. 计时：ros::Time、ros::Duration 15](#_Toc159688527)

[5. 其他工具 Tools 15](#_Toc159688528)

[5.1. Rqt：多功能数据可视化 15](#_Toc159688530)

[5.1.1. Rqt：主界面 15](#_Toc159688537)

[5.1.2. Rqt Graph：节点话题可视化 16](#_Toc159688538)

[5.1.3. Rqt Topic：话题监视器 17](#_Toc159688539)

[5.1.4. Rqt Plot：话题数据绘图 17](#_Toc159688540)

[5.1.5. Rqt Image View：查看图像通道 17](#_Toc159688541)

[5.1.6. Rqt Reconfigure： 17](#_Toc159688542)

[5.2. PlotJuggler：日志、话题绘图分析 17](#_Toc159688546)

[5.2.1. 实时显示话题数据 18](#_Toc159688553)

[5.2.2. 日志分析 19](#_Toc159688554)

[5.3. Rviz：3D场景可视化 20](#_Toc159688555)

[5.4. Gazebo：仿真 20](#_Toc159688556)

[6. 常用命令 Common Commands 20](#_Toc159688557)

[6.1. ROS相关文件访问 20](#_Toc159688559)

[1) roscd：访问指定包的路径 20](#_Toc159688560)

[2) rosls：列出指定包文件夹内的对象 21](#_Toc159688561)

[3) rosed：编译指定软件包中的指定文件 21](#_Toc159688562)

[4) rospack：列出软件包的信息 21](#_Toc159688563)

[5) rosversion：查看软件包的版本 21](#_Toc159688564)

[6.2. 启动与运行 21](#_Toc159688565)

[1) roscore：启动ROS系统 21](#_Toc159688566)

[2) rosrun：启动指定软件包中的指定节点 21](#_Toc159688567)

[3) roslaunch：运行指定软件包中的指定.launch文件 21](#_Toc159688568)

[6.3. 参数服务器 22](#_Toc159688569)

[1) rosparam：参数服务器的操作 22](#_Toc159688570)

[6.4. 节点 22](#_Toc159688571)

[1) rosnode：运行中节点检索与操作 22](#_Toc159688572)

[6.5. 话题与消息 23](#_Toc159688573)

[1) rostopic：活跃话题检索与操作 23](#_Toc159688574)

[2) rosmsg：消息类型查询 24](#_Toc159688575)

[6.6. 服务 24](#_Toc159688576)

[1) rosservice：活跃服务检索与操作 24](#_Toc159688577)

[2) rossrv：服务类型查询 25](#_Toc159688578)

[6.7. 调试命令 25](#_Toc159688579)

[1) roswtf：系统纠错 25](#_Toc159688580)

[7. 无人机接口：MAVROS 26](#_Toc159688581)

[7.1. 26](#_Toc159688589)

[8. RealSense相机接口：Realsense2 26](#_Toc159688590)

[9. 实践：Turtlesim小乌龟移动控制 26](#_Toc159688591)

[8.1. 相关节点 26](#_Toc159688593)

[10. 实践：MAVROS无人机SITL 26](#_Toc159688594)

[8.1. 安装MAVROS 27](#_Toc159688603)

[8.2. 下载PX4源码 27](#_Toc159688604)

[8.3. 编译PX4 SITL仿真模型 28](#_Toc159688605)

[8.4. 安装QGroundControl 29](#_Toc159688606)

[8.5. 编写控制程序（C++） 29](#_Toc159688607)

[8.6. 运行仿真 29](#_Toc159688608)

[8.7. 仿真效果 30](#_Toc159688609)

[参考资料 32](#_Toc159688610)

# 系统安装 Installation

## 版本选择

ROS的发行版本与Ubuntu的版本一一对应，与Ubuntu系统一样，每两年发布一个长期支持版（Long-term Support），安装时，ROS的版本一定要与Ubuntu系统的版本对应。

由于目前（2024年）正处于ROS与ROS2的过渡时期，运行于Ubuntu 20.04的Noetic已是第一代ROS系统的最终发行版，Ubuntu 22.04以及未来的系统将搭配ROS2，不再支持ROS。不过目前ROS2的社区和软件支持还未成熟，大多数的开发者还在使用第一代ROS。

目前开发常用的ROS版本如下：

1. Ubuntu 18.04 – ROS Melodic - <https://wiki.ros.org/melodic/Installation/Ubuntu>
2. Ubuntu 20.04 – ROS Noetic - <https://wiki.ros.org/noetic/Installation/Ubuntu>
3. Ubuntu 22.04 – ROS2 Humble - <https://docs.ros.org/en/humble/Installation.html>

## 下载安装

ROS与ROS2的下载步骤有一定区别，具体参考对应版本下载页面上的教程。下面是基本通用的安装步骤说明：

·ROS：添加下载源 >> curl下载密钥 >> 更新apt源 >> apt下载 >> 配置环境变量

以一代ROS的最终发行版ROS Noetic安装为例，依次在命令行输入：

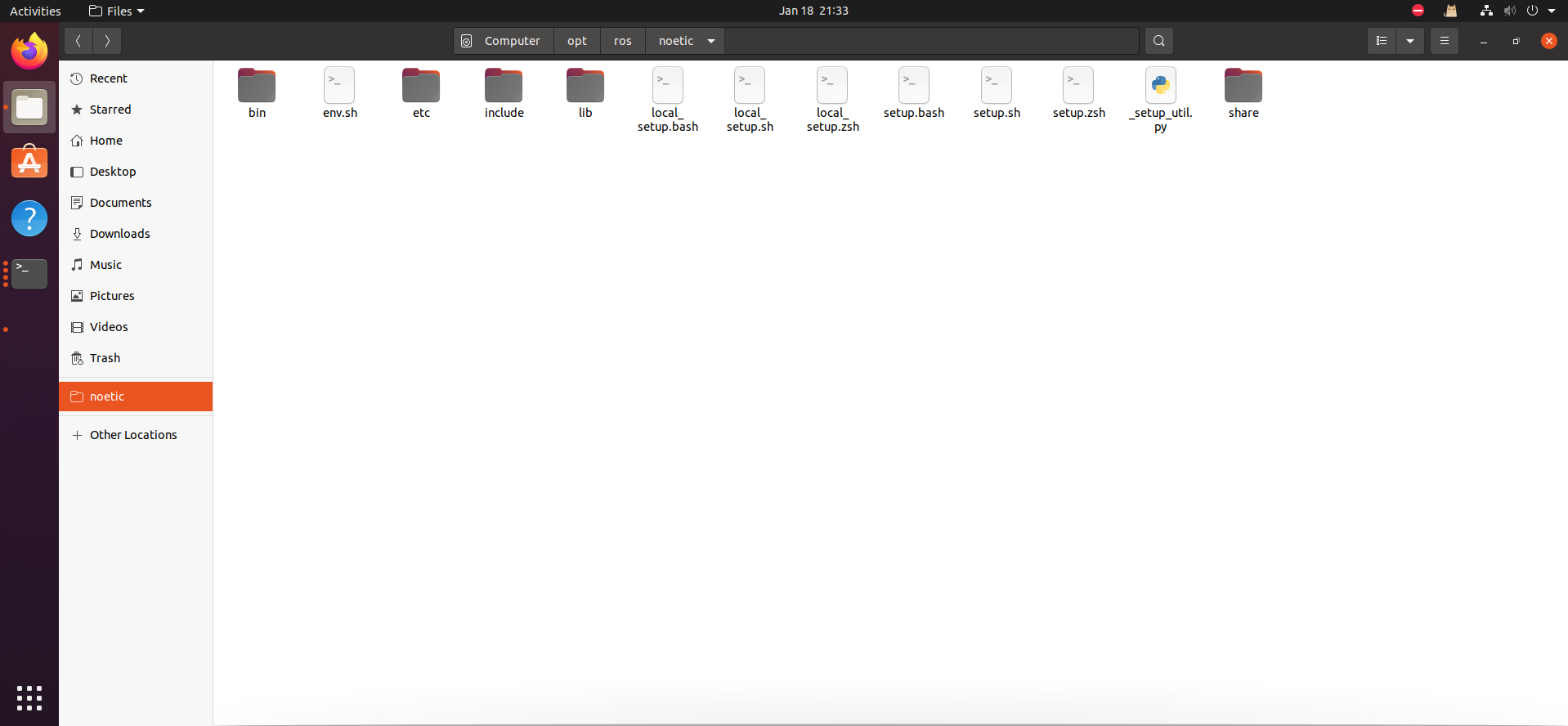
|  |
| --- |
| $ sudo sh -c 'echo "deb http://packages.ros.org/ros/ubuntu $(lsb\_release -sc) main" > /etc/apt/sources.list.d/ros-latest.list'  $ sudo apt install curl  $ curl -s https://raw.githubusercontent.com/ros/rosdistro/master/ros.asc | sudo apt-key add  $ sudo apt update  $ sudo apt install ros-noetic-desktop-full  $ echo "source /opt/ros/noetic/setup.bash" >> ~/.bashrc |

如果用curl命令获取密钥的速度很慢，可以直接在当前路径创建一个名为“ros.asc”的文件，浏览器打开上面curl提到的网页，把文本复制到新创建的文件里，运行“sudo apt-key add ros.asc”

## 主文件访问

安装成功后，ROS的主文件夹位于以下路径：

|  |
| --- |
| $ cd /opt/ros/<distros>  #<distros>代表当前下载的发行版 |



图：ROS主文件夹

主文件夹下的share文件夹中可以找到下载的软件包，ROS系统也提供了一系列命令用于直接访问软件包所在路径，参见[文件访问](#_文件访问)。

## 卸载

有时候ROS出现一些难以排查的问题，我们希望卸载重装整个ROS系统，卸载的命令如下：

|  |
| --- |
| $ sudo apt remove ros-\*  $ sudo apt autoremove  #第一行自动卸载所有ROS相关的软件包；第二行卸载所有额外的依赖项 |

# 基本概念 Basic Concepts

·官方文档：<https://wiki.ros.org/ROS/Concepts>

ROS中最重要的三个概念是节点Node、话题Topic、服务Service，节点是机器人系统运行的单元，消息与服务则是节点之间建立通讯的基础。在一个庞大的机器人系统中，节点、话题、服务组成类似计算图Graph的结构，可以用ROS自带的rqt\_graph工具查看当前系统的计算图。

|  |
| --- |
| $ rosrun rqt\_graph rqt\_graph  $ rqt\_graph  #以上两种都行 |

在第一代ROS中，系统的节点注册Registration、通讯连接依靠的是一个统筹全局的节点管理器Master，在ROS2则不再有Master的概念。



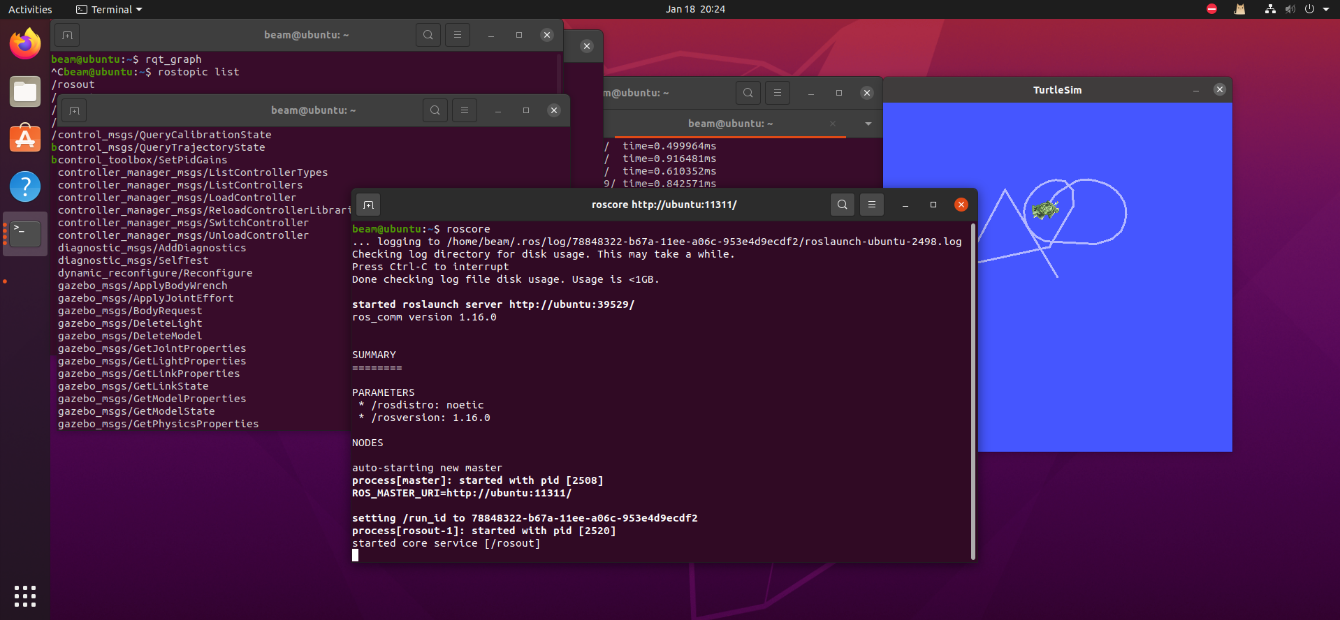
## 节点管理器Master

·官方文档：<https://wiki.ros.org/Master>

·相关命令：[roscore](#_roscore：启动ROS系统)

·关键词：注册Registration

Master是ROS中最核心的进程，它对ROS系统中启用的节点进行注册Registration，为它们提供一个系统中唯一的名字，并在各节点之间建立起话题和服务的通信。在开启ROS系统时运行的roscore命令其中一个功能就是启动Master。

****

图：Master运行的命令行

## 参数服务器Parameter Server

·官方文档：<http://wiki.ros.org/Parameter%20Server>

·相关命令：[rosparam](#_rosparam：参数服务器的操作)、[roscore](#_roscore：启动ROS系统)

·关键词：静态参数Static Parameters

参数服务器Parameter Server随roscore命令执行而启动，是一个供各节点保存静态参数（不需要经常改变的参数），参数服务器上的参数对全局开放，任何节点都可以读写（命令为获取get、设置set）上面保存的参数。

需要注意的是，参数的读写并不支持实时动态更新，这意味着，如果参数<param1>在被节点<node1>读取后更新了值，<node1>并不知道参数<param1>的更新，除非<node1>重新读取<param1>。

## 节点Node

·官方文档：<http://wiki.ros.org/Nodes>

·相关命令：[rosnode](#_rosnode：)

·关键词：进程Process、可执行文件Executable、

节点是ROS系统中运行的单个进程，在ROS的软件包中表现为一个可执行文件Executable。作为ROS系统的运行单元，节点可以发布或者订阅话题、提供或者使用服务。

无论多复杂的ROS系统，其结构都是一个由很多节点组成的图。

ROS系统可以用rosrun命令直接运行节点，但在大型项目中，一般使用roslaunch运行.launch文件，在.launch文件内部用<node>元素运行节点。

## 话题与消息Topic & Message(msg)

·官方文档：<http://wiki.ros.org/Topics>（话题）& <http://wiki.ros.org/Messages>（消息）

·相关命令：[rostopic](#_rostopic：活跃话题检索与操作)、[rosmsg](#_rosmsg：消息查询)

·关键词：发布者Publisher、订阅者Subscriber

话题是ROS系统提供的一种单向传输的通信方式，节点可以发布话题，也可以订阅话题，信息单向地从发布者流向订阅者，不存在反馈，通俗来讲，话题相当于现实中的广播。发布者、订阅者是相对具体话题而言的概念，同一个节点可以同时发布多个话题、接收多个话题。

多个节点同时向一个话题中发布消息是可能的，但并不推荐，因为这种做法容易引起话题信息的混乱，类似无线电串频。

话题的类型Type称为消息，消息是由软件包中消息文件.msg所定义的数据格式，描述了话题传输的数据类型Type。话题的发布需要指定所使用的消息类型。

## 服务 Service

·官方文档：<https://wiki.ros.org/Services>

·相关命令：[rosservice](#_rosservice：)、[rossrv](#_rossrv：)

·关键词：服务器Server、客户端Client、请求Request、回答Reply、服务类型Service Type(srv)

服务是一种双向传播的通讯方式，客户端Client节点向服务器Server节点发送请求Request信息，并接受服务端发送的回答Reply信息。与话题类似，同一个节点可以同时充当服务器和客户端。

服务类型文件.srv定义了一种服务的请求和回答信息数据格式。服务类型与消息的定义很类似，因此两者经常被统称为消息。

# 编程与开发：工作空间 Workspace

为了让用户可以编写自己的机器人控制程序，ROS提供了两种主流语言C++和Python的编程接口roscpp和rospy，这种接口被称为客户端库Client Library（注意！和服务通讯的客户端ServiceClient没有任何关系）。用户的自主开发一般在自创的“工作空间”文件夹中进行。

工作空间Workspace是由你自己定义的，用于存放包Packages、编译源代码的文件夹，运行catkin\_make之后，工作空间就成为一个标准的catkin空间。在一个定义规范的工作空间中进行开发是良好的习惯。



## 工作空间：创建与初始化

工作空间一般存放在Home文件夹中，创建名为catkin\_ws的工作空间命令如下，catkin\_ws可以改成别的名字，下级的src不能改名字。

|  |
| --- |
| $ mkdir -p ~/catkin\_ws/src  $ cd ~/catkin\_ws/  $ catkin\_make  $ catkin\_make install  #命令mkdir创建新文件夹，cd进入新创建的catkin\_ws工作空间，catkin\_make将工作空间初始化为可用于编译的标准catkin工作空间，catkin\_make install（非必须）用于创建install文件夹。 |

完成后，catkin\_ws中有以下文件和文件夹。

1. 编译文件夹“build”：编译过程中产生的文件，一般不怎么用到；
2. 开发文件夹“devel”： 可执行文件以及一些库文件；
3. 安装文件夹“install”：编译的成果——可执行文件和bash脚本；
4. 文件夹“src”：我们定义的包和源代码文件、编译参数配置文件；
5. “.catkin\_workspace”：标记当前文件夹是catkin文件夹。



图：工作空间“catkin\_ws”文件夹

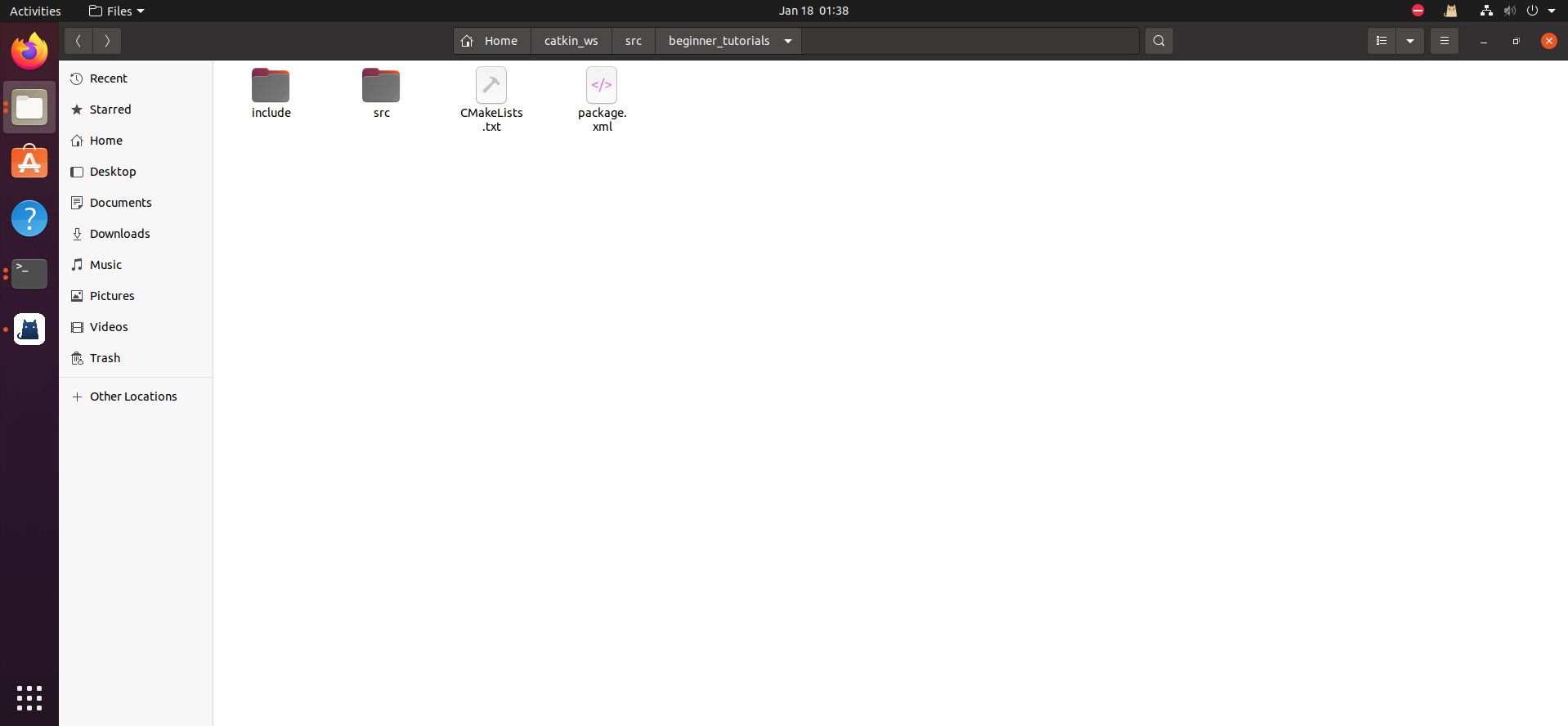
## 创建软件包

进入src文件夹中，输入命令catkin\_create\_pkg，后面的第一个参数beginner\_tutorials是新创建包的名字，其他参数是包的多个依赖项。

|  |
| --- |
| $ catkin\_create\_pkg beginner\_tutorials std\_msgs rospy roscpp |

在新创建的beginner\_tutorials包的文件夹中，有以下文件和文件夹：

1. 文件夹“include”：存放源代码的头文件
2. 文件夹“src”：存放源代码的主体部分
3. “CmakeLists.txt”：包的编译配置文件
4. “packages.xml”：记录包的依赖项



图：工作空间“catkin\_ws”中的“beginner\_tutorials”软件包文件夹

## 编译C++文件

假设现在已经在beginner\_tutorials包的src文件夹下创建好名为test\_code.cpp的C++源码文件，并且已经在里面写好程序，下一步操作就是编译这个C++文件。

点击进入beginner\_tutorials包的CmakeLists.txt文件，写入以下内容并保存，test\_code.cpp就添加到工作空间的编译列表上了。

|  |
| --- |
| add\_executable(test\_node src/test\_code.cpp)  target\_link\_libraries(test\_node ${catkin\_LIBRARIES})  # 第一行表示将test\_code.cpp文件编译成名叫test\_node的节点 |

退回到工作空间主目录~/catkin\_ws，直接运行catkin\_make，系统就会自动编译列表上列出的所有C++源码文件了，已经编译过且未作修改的文件会被跳过。

## 编写消息和服务类型定义文件.msg及.srv

·官方教程：<https://wiki.ros.org/ROS/Tutorials/CreatingMsgAndSrv>

编写前需要先在功能包文件夹中创建msg和srv文件夹（名字不能改）。



### 基本语法

消息文件.msg和服务类型文件.srv的语法结构非常相似，两者本质上都是声明一些变量的文本文件，只是.srv需要定义调用服务时的输入参数和调用完成后的命令行输出。为了确保ROS系统跨语言的兼容性，涉及数值变量的关键词用字长来表示。

|  |  |
| --- | --- |
| .msg文件 | .srv文件 |
| Header header  string name  geometry\_msgs/Twist twist  int32 value | int32 addA  int32 addB  ---  int32 sum  string feedback |

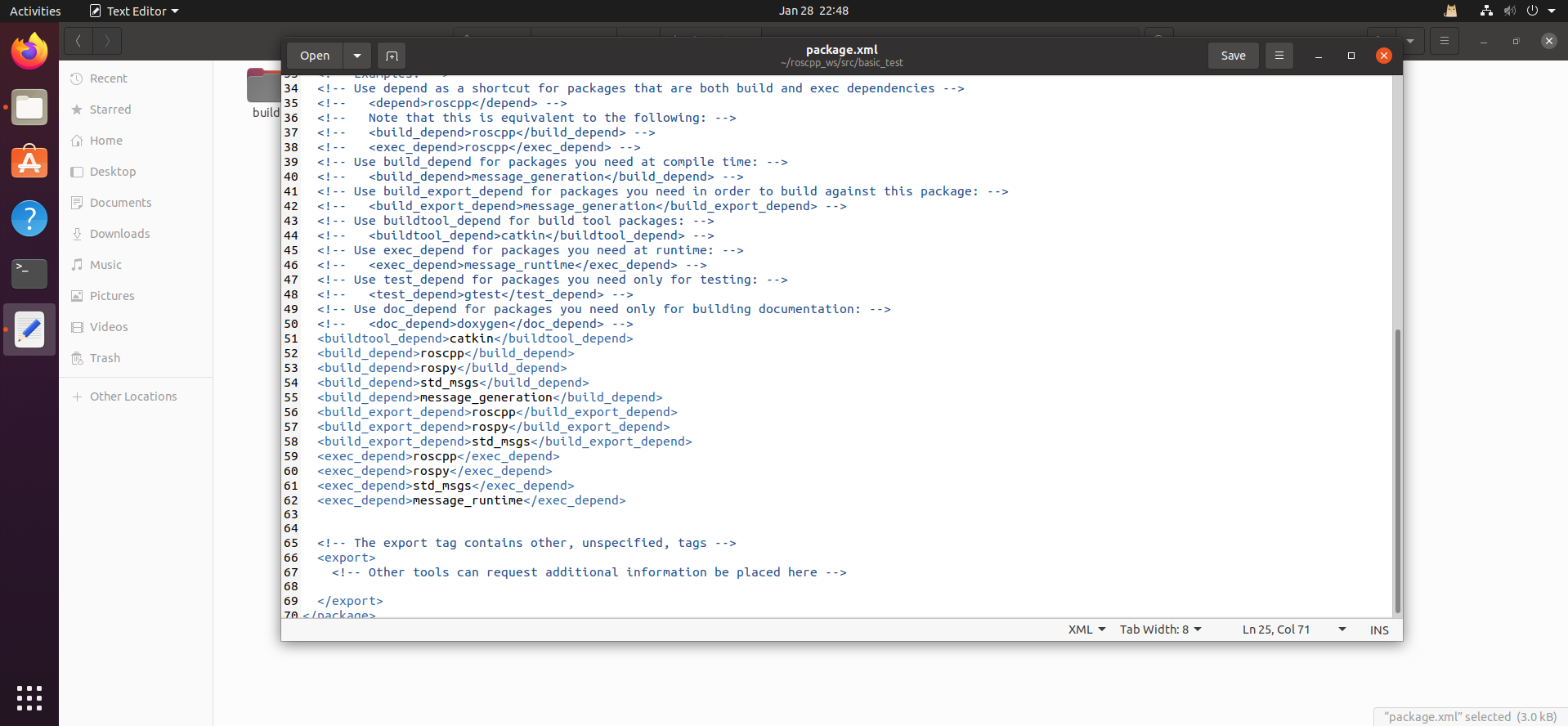
常用的声明关键词：

1. 数值：有符号整型（int8、int16、int32、int64）、无符号整型（uint8、uint16、uint32、uint64）、浮点型（float32、float64）
2. 字符串：string
3. 时间相关：time、duration
4. 嵌套定义：允许嵌套使用其他类型的消息/服务类型定义一个子变量，比如在“geometry\_msgs/TwistStamped”消息中就定义了一个“geometry\_msgs/Twist”自变量“twist”
5. 数组：上述所有类型可以在类型关键词后面加中括号“[]”声明成数组的形式，如“geometry\_msgs/PoseStamped[]”

### packages.xml新增依赖项

编写完成后，打开功能包的packages.xml依赖项配置文件，补充下面两行内容，加入消息的生成与运行依赖项。

|  |
| --- |
| <build\_depend>message\_generation</build\_depend>  <exec\_depend>message\_runtime</exec\_depend> |



### 修改CMakeLists.txt

|  |  |
| --- | --- |
| find\_package中加入message\_generation |  |
| catkin\_package去除CATKIN\_DEPENDS的注释，后面补充message\_runtime |  |
| 生成消息：add\_message\_files取消注释，在FILE下面按行列出需要生成的消息文件名 |  |
| 生成服务类型：add\_service\_files取消注释，补上文件名 |  |
| generate\_messages取消注释，补上自定义的消息、服务类型所调用的消息、服务所属功能包 |  |

如果add\_message\_files和add\_service\_files没有取消注释，默认会编译msg和srv文件夹中的所有识别到的文件。

修改完成以后，回到工作空间一级目录运行catkin\_make即可完成消息和服务的编译。编译成功的标志是rosmsg和rossrv命令可以检测到新建立的消息和服务类型，这时就可以在编写节点时调用这些消息、服务来声明发布者和服务器了。

## 编写启动脚本文件.launch (.xml)

·格式文档：<http://wiki.ros.org/roslaunch/XML>

.launch文件是一种功能强大的启动机制，通过roslaunch命令启动，它提供了一种同时启动多个节点，并配置参数服务器中参数的方法，.launch文件甚至可以在不修改源码的情况下，更改节点的启动名称、修改发布者、订阅者的话题名称。大型项目一般都使用roslaunch启动

.launch文件使用XML文件格式编写，后缀为.launch的文件可以由roslaunch直接运行，后缀为.xml的文件不能由roslaunch运行，但可以在.launch文件中使用<include>调用。

## Python文件

## 插件 Plugin

·官方文档：

·关键词：

# 编程与开发: C++ Coding and Developing

·官方roscpp文档：<https://wiki.ros.org/roscpp/Overview>

C++是ROS官方支持的两种开发语言之一，相比Python而言，C++实现的ROS程序的运行效率更高，在社区也更为常用。使用C++开发时，参数服务器、发布者、订阅者、消息、服务端、客户端、服务类型等通讯方式通过面向对象实现。

ROS节点的C++源码必须包含以下头文件，以确保roscpp的相关函数和类可以被成功调用。

|  |
| --- |
| #include <ros/ros.h> |



## 命名规范

·官方文档：<https://wiki.ros.org/Names>

## 节点的初始化、启动与关闭

·官方文档：<https://wiki.ros.org/roscpp/Overview/Initialization%20and%20Shutdown>

·相关函数及类：ros::init()、ros::start()、ros::NodeHandle、ros::shutdown()、ros::ok()、ros::isShuttingdown()

ROS节点在启动前必须要用函数ros::init()初始化；argc和argv是为了让节点可以接受命令行传入的参数；<node\_name>是节点在ROS系统中的默认名称（不能含有斜杠“/”），在启动时可以用Remapping重命名。

|  |
| --- |
| ros::init (argc, argv, “node\_name”) |

虽然ROS提供了开启节点的函数ros::start()，不过一般不使用这个函数，而是以节点句柄Node Handle的创建作为节点开启的标志。节点句柄的销毁则意味着节点的终止运行。

|  |
| --- |
| ros::NodeHandle nh |

函数ros::ok()返回一个布尔值用于判断节点是否已经被关闭，常作为是否跳出while循环体的判断条件。

|  |
| --- |
| while(ros::ok())  {} |

默认情况下，在启动节点的命令行中按CTRL+C就会终止节点的运行。

## 导入消息和服务类型

在C++中，消息与服务类型通过include头文件的方式引入，以类的形式使用，如果代码中需要用到消息“geometry\_msgs/PoseStamped”，则需要

|  |
| --- |
| #include <geometry\_msgs/PoseStamped.h> |

在代码中引用到消息与服务类型时，格式是

|  |
| --- |
| geometry\_msgs::PoseStamped |

## 话题：发布者与订阅者的创建

·官方教程：<https://wiki.ros.org/ROS/Tutorials/WritingPublisherSubscriber%28c%2B%2B%29>

·官方文档：<https://wiki.ros.org/roscpp/Overview/Publishers%20and%20Subscribers>

·相关函数及类：ros::Publisher、ros::Subscriber

·发布者：

创建一个发布者对象的代码如下。其中，“pub”是代码中发布者对象的名称，仅在源代码内部使用；“nh”是节点句柄的名称；“std\_msgs::String”是发布话题使用的消息类型；“topic\_name”是话题的名称；“100”是话题寄存器的大小。

|  |
| --- |
| ros::Publisher pub = nh.advertise<std\_msgs::String>("topic\_name", 100); |

调用发布者对象的成员函数publish()将代码中事先定义好的信息“my\_text”发布到话题中，就完成了一次发布。

|  |
| --- |
| std\_msgs::String my\_text = “I love ROS”  pub.publish(my\_text) |

特别的，如果需要在订阅者的回调函数中进行发布，创建发布者时需要将声明和定义分开，在全局空间中声明，在main()函数中定义。

|  |
| --- |
| ros::Publisher pub  int main(int argc, char \*\*argv)  {  …  pub = nh.advertise<std\_msgs::String>("topic\_name", 100);  } |

·订阅者：

创建订阅者对象的代码与之类似，但需要指定回调函数“receive\_callback”才能接收到消息，声明时可省略消息类型；“topic\_name”是订阅话题的名称（不包含斜杠“/”）

|  |
| --- |
| ros::Subscriber sub = nh.subscribe<std\_msgs::String>("topic\_name", 100, receive\_callback); |

下面是订阅者的回调函数的基本结构：订阅者的回调函数接受一个消息的常类指针“const <消息名称>::ConstPtr& msg”，在函数内部通过“msg->data”访问订阅者接收到的消息类内部的成员值。

|  |
| --- |
| void receive\_callback(const std\_msgs::String::ConstPtr& msg)  {  ROS\_INFO("I heard: [%s]", msg->data.c\_str());  } |

如果订阅者订阅的话题上没有新的消息发布，则回调函数不会被调用。

## 服务：服务器与客户端的创建

·官方教程：<https://wiki.ros.org/ROS/Tutorials/WritingServiceClient%28c%2B%2B%29>

·官方文档：<https://wiki.ros.org/roscpp/Overview/Services>

·相关函数及类：ros::ServiceServer、ros::ServiceClient

·服务端：

·客户端：

## 参数服务器：参数的创建与获取

·官方教程：

·官方文档：

参数服务器

## 回调与旋回：ros::spin()、ros::spinOnce()

·官方文档：<https://wiki.ros.org/roscpp/Overview/Callbacks%20and%20Spinning>

·相关函数及类：ros::spin()、ros::spinOnce()

订阅者Subscriber创建后，依靠定义在main()外的回调函数Callback Function对接收的数据进行处理，但回调函数并不会自己执行，因此需要一种方法来启动回调函数。

ros::spin()函数一般写在main()主函数的结尾处，一但开始执行，节点进入回调事件循环，循环调用各订阅者的回调函数，直到节点被关闭。因此，main()中在ros::spin()函数之后的代码是不会执行的。

ros::spinOnce()函数在执行时只调用一次回调函数，完成后会继续执行main()中的代码，一般会配合while循环使用以确保对回调函数的持续调用。

## 定频循环：ros::Rate

·官方文档：<https://docs.ros.org/en/kinetic/api/rostime/html/classros_1_1Rate.html>

·相关函数及类：ros::Rate

## 计时器：ros::Time、ros::Duration

·官方文档：

# 其他工具 Tools



ROS系统中有自带的辅助程序，

## Rqt：多功能数据可视化

·官方文档：<https://wiki.ros.org/rqt>

Rqt是ROS自带的基于Qt图形交互界面开发的多功能数据可视化工具，它包含了许多常用的调试插件Plugin，这些插件可以在Rqt主界面中打开，也可以使用命令单独打开。

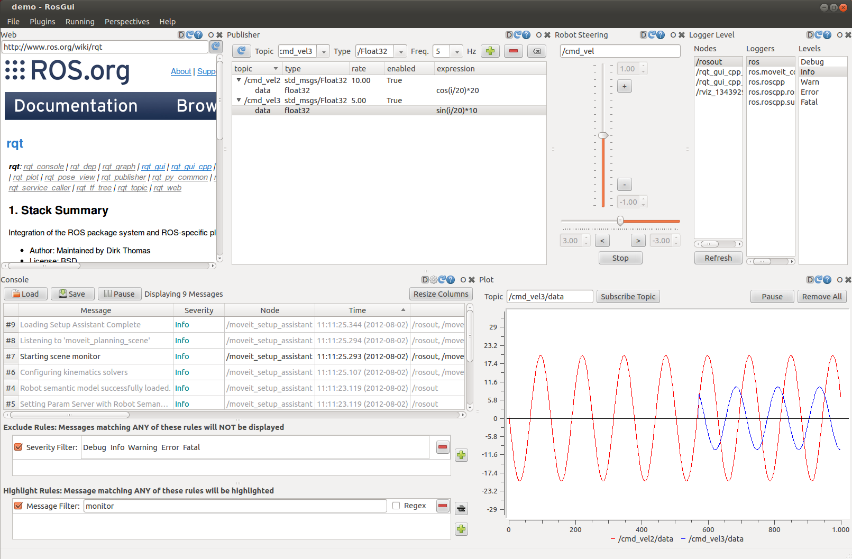


### Rqt：主界面

·打开命令：

|  |
| --- |
| rqt |

在Rqt的主页面中，不同的插件可以放置在同一个窗口中，用户可以自定义一个综合的开发调试工具。



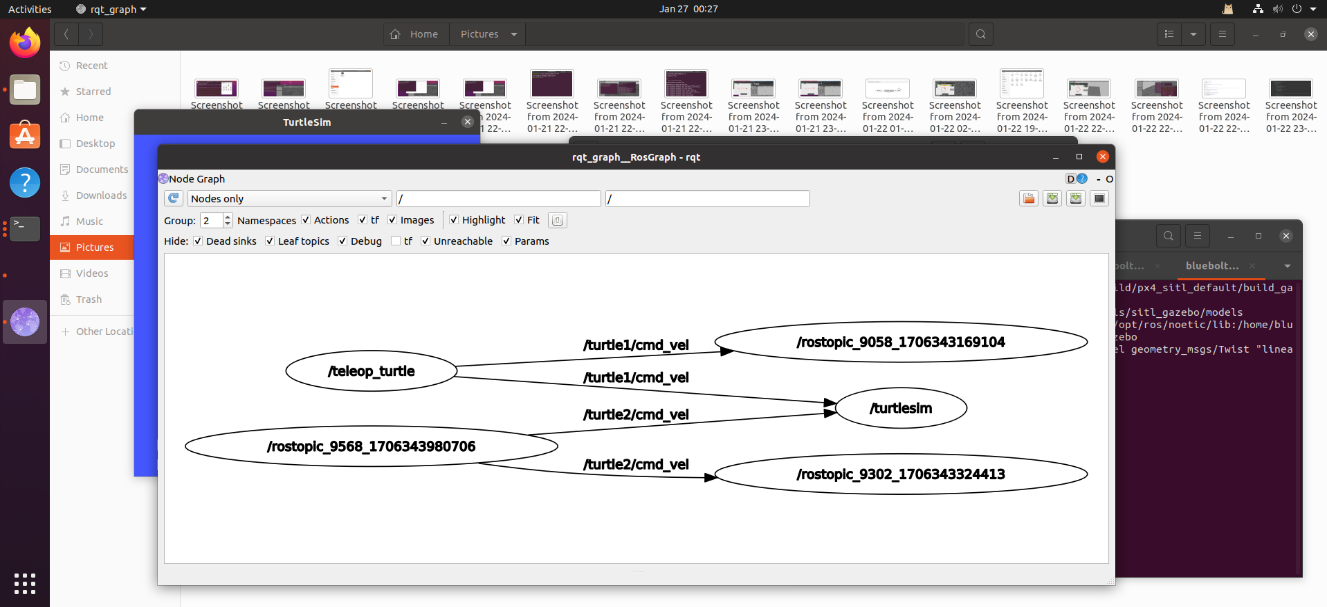
### Rqt Graph：节点话题可视化

·打开命令：

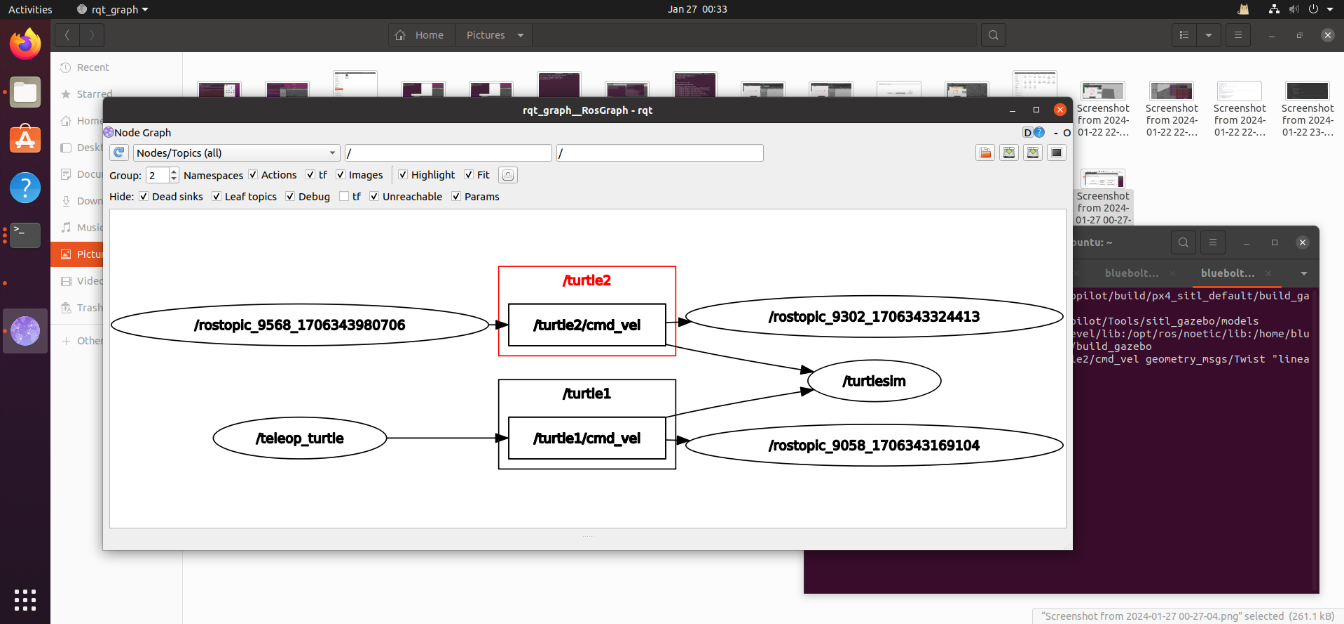
|  |
| --- |
| rqt\_graph |

Rqt Graph是一个节点图可视化工具，它会自动识别ROS系统中启用的节点以及节点间的话题传播关系并绘制可视化的有向图。

·“仅显示节点”模式：图中的椭圆表示一个节点，箭头表示一个话题。（ROS系统的日志记录节点/rosout和Rqt自身的节点会被省略）；



·“节点与消息”模式：节点会单独以方框显示



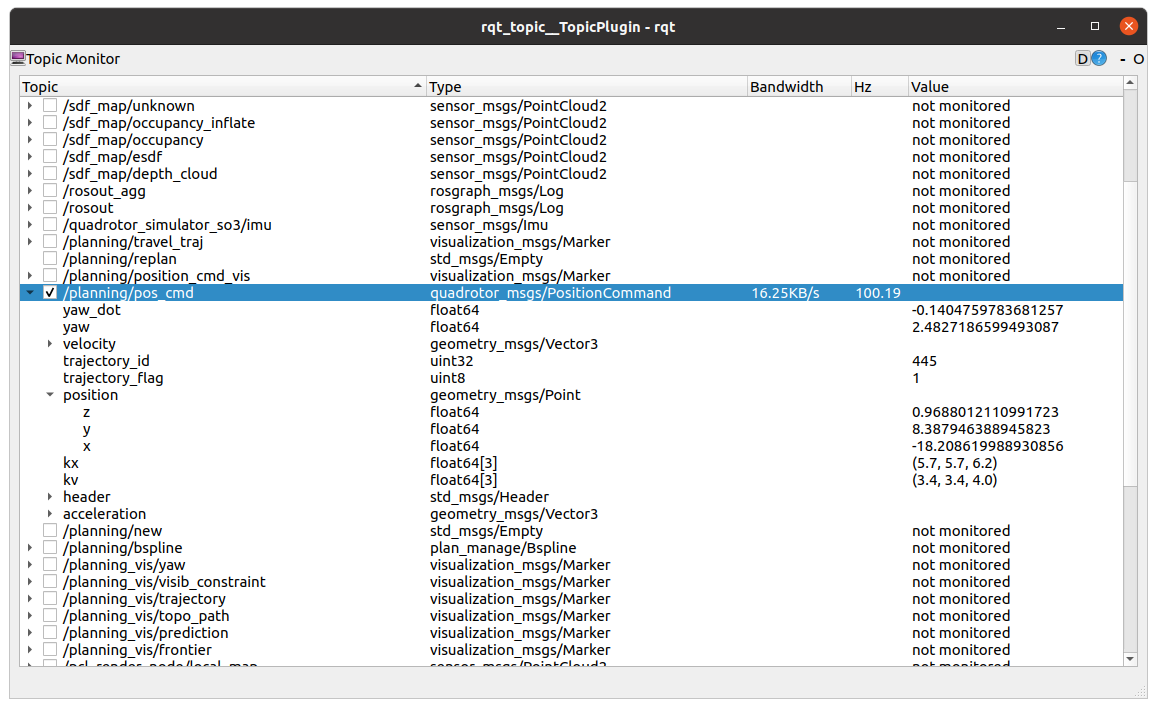
### Rqt Topic：话题监视器

·打开命令：

|  |
| --- |
| rqt\_topic |

Rqt Topic是一个话题监视器，其功能覆盖了rostopic命令的list、info、hz、echo。它以分级表格的形式列出了所有活跃话题，并可以以1Hz的频率监视选中的话题具体取值。

唯一的缺点是，同时监视的话题太多可能会卡住。



### Rqt Plot：话题数据绘图

### Rqt Image View：查看图像通道

### Rqt Reconfigure：



## PlotJuggler：日志、话题绘图分析

PlotJuggler支持话题数据实时分析、日志分析，被认为是ROS环境下最好用的数据绘图工具，由于ROS安装包默认不包含PlotJuggler使用前需要另外安装。

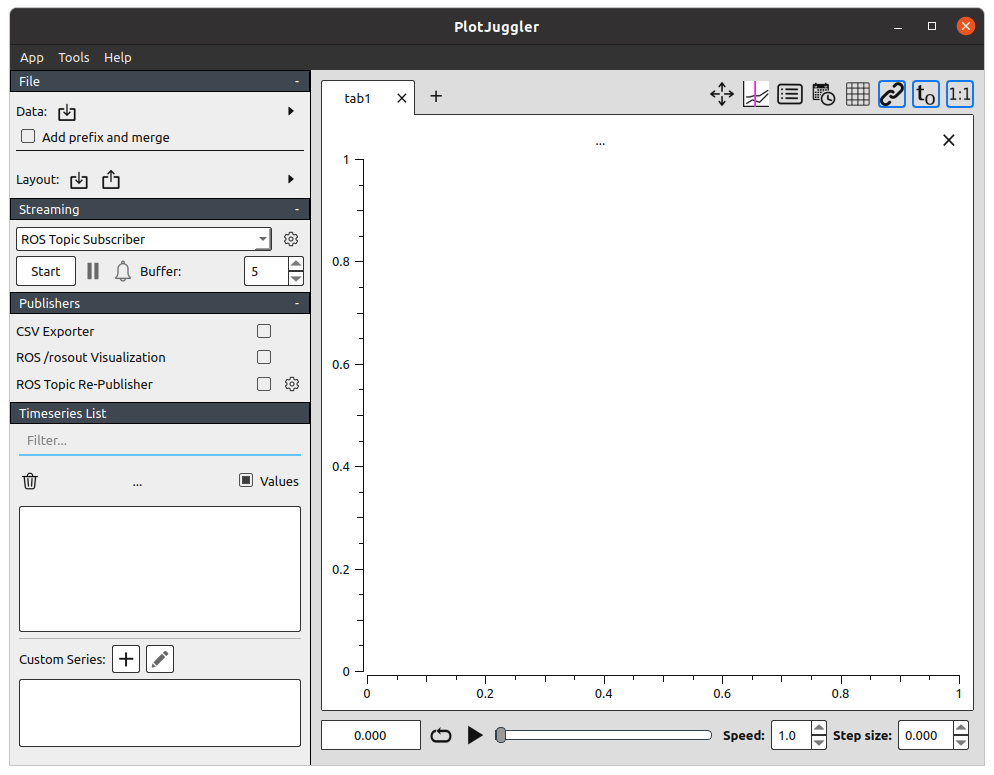
·安装命令：（Noetic是ROS发行版本）

|  |
| --- |
| sudo apt install ros-noetic-plotjuggler-ros  #一定要加最后面的“\*-ros”，否则无法实时分析话题数据 |

·打开命令：

|  |
| --- |
| rosrun plotjaggler plotjaggler |

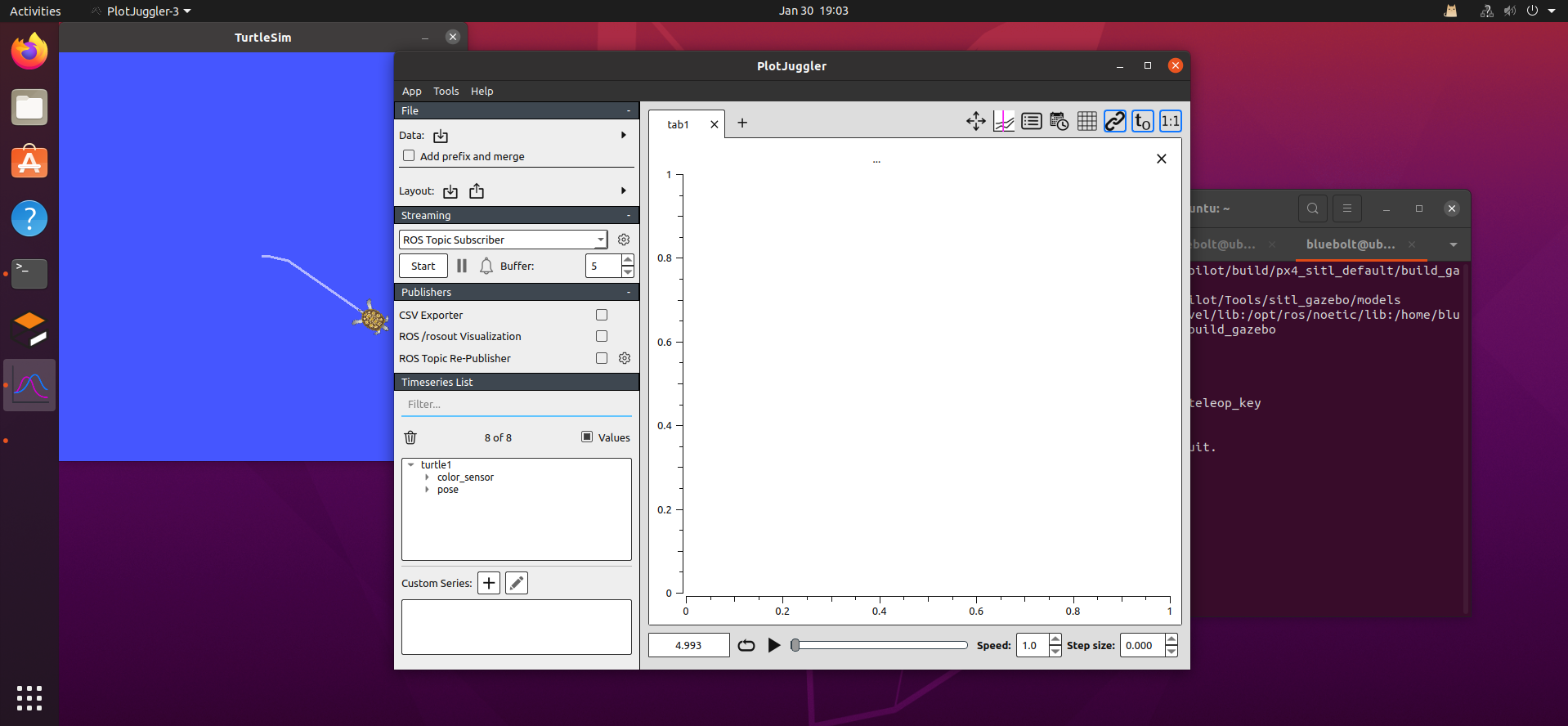
成功打开后，~~会弹出来一张模因图~~，界面如下图所示。一定要确认左上角的信息流“Streaming”有“ROS话题订阅者”的选项，否则无法实时分析话题。





### 实时显示话题数据

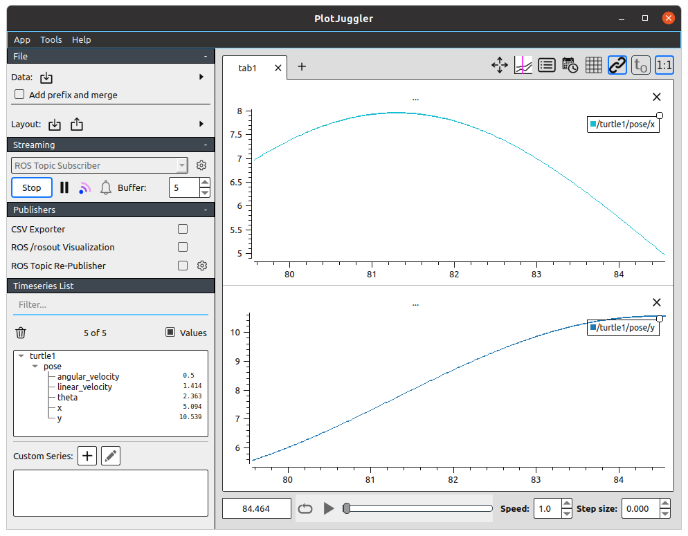
左上角的“Streaming”选择“ROS Topic Subscriber”，点击开始，在弹出的窗口里选择，需要监听的话题通道，然后在左下角的时间序列“Timeseries List”把需要看的通道拖到右侧图窗中即可实现数据的在线绘图。



·暂停/继续监听：左上角“Streaming”的暂停按钮；

·关闭/显示曲线：点击图窗上的对应图例；

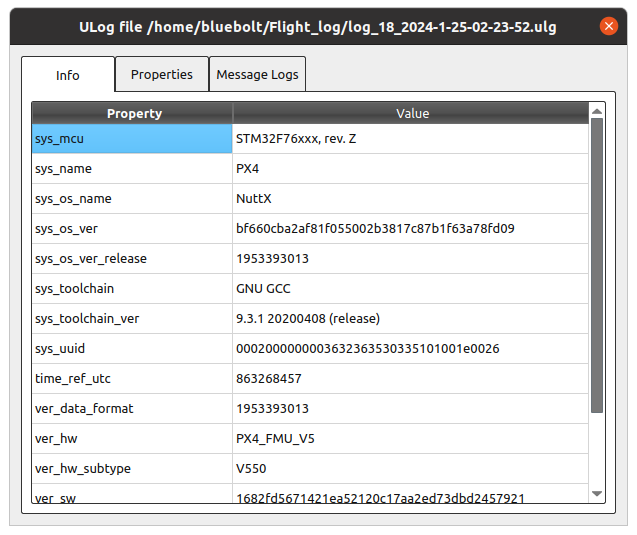
·多图窗分屏：图窗区域右键选择水平分屏“Split Horizontally”、垂直分屏“Split Vertically”



### 日志分析

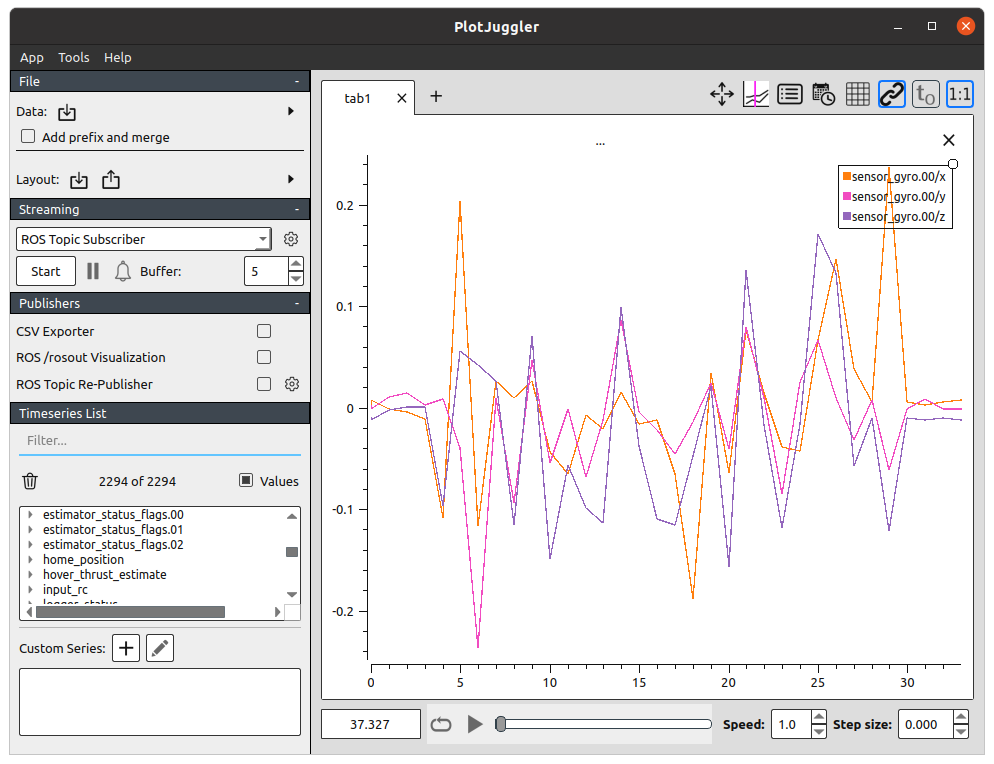
·PX4飞控日志分析教程：<https://docs.px4.io/main/en/log/plotjuggler_log_analysis.html>

这里以飞控日志为例。在QGC中导出的飞控日志文件可以直接拖进PlotJuggler里，弹出的窗口会显示日志的总览、飞控参数和显示消息；其余数据在“Timeseries List”列表中，可以拖到图窗中分析。



飞控日志数据的具体含义参考PX4内部传输协议uORB。

·PX4中间层uORB文档：<https://docs.px4.io/main/en/middleware/>



## Rviz：3D场景可视化

## Gazebo：仿真

# 常用命令 Common Commands

·常用命令行总览：<http://wiki.ros.org/ROS/CommandLineTools>

根植于Linux系统，ROS的启动、调试、排错和文件处理都是靠命令行实现的。



## ROS相关文件访问

ROS的原生软件包rosbash引入了一系统接受软件包名称作为路径参数的命令，可用于高效地访问、查找、修改ROS软件包中的文件。下面所列举的文件操作命令并不是全部位于rosbash包中。

·官方文档：<https://wiki.ros.org/rosbash>

### roscd：访问指定包的路径

·官方文档：<https://wiki.ros.org/rosbash#roscd>

|  |  |
| --- | --- |
| roscd | cd到ROS主文件夹/opt/ros/<distros> |
| roscd roscpp | cd到roscpp包所在的文件夹 |

### rosls：列出指定包文件夹内的对象

·官方文档：<https://wiki.ros.org/rosbash#rosls>

|  |  |
| --- | --- |
| rosls roscpp -a | 列出roscpp包所在的文件夹中的所有对象 |

### rosed：编译指定软件包中的指定文件

·官方文档：<https://wiki.ros.org/rosbash#rosed>

|  |  |
| --- | --- |
| rosed test\_pkg test\_code.cpp | 使用vim编辑test\_pkg软件包中的test\_code.cpp |

### rospack：列出软件包的信息

·官方文档：<https://docs.ros.org/en/independent/api/rospkg/html/rospack.html>

### rosversion：查看软件包的版本

## 启动与运行

### roscore：启动ROS系统

·官方文档：<https://wiki.ros.org/roscore>

每次使用ROS前要运行这个命令，它实际上执行三个任务：开启节点管理器Master、开启参数服务器Parameter Server、开启/rosout日志节点。

### rosrun：启动指定软件包中的指定节点

·官方文档：<https://wiki.ros.org/rosbash#rosrun>

|  |  |
| --- | --- |
| rosrun test\_pkg test\_node | 运行test\_pkg软件包中的test\_node节点（可执行文件） |

### roslaunch：运行指定软件包中的指定.launch文件

·官方文档：<http://wiki.ros.org/roslaunch>

特别的，若roslaunch检测到roscore指令未运行，会自动运行roscore。

|  |  |
| --- | --- |
| rosrun test\_pkg test.launch | 运行test\_pkg软件包中的test.launch文件 |

## 参数服务器

### rosparam：参数服务器的操作

·官方文档：<http://wiki.ros.org/rosparam>

|  |  |
| --- | --- |
| rosparam list | 列出参数服务器中的所有参数 |
| rosparam get /turtlesim/background\_g | 查看/turtlesim/background\_g参数的值 |
| rosparam set /turtlesim/background\_g 9 | 设置/turtlesim/background\_g参数的值为9 |
| rosparam dump | 显示所有参数以及具体的值 |
| rosparam dump > save.txt | 将当前所有参数以YAML格式写入save.txt文件中 |
| rosparam load save.txt | 从save.txt中读取参数设置值 |
| rosparam delete /turtlesim/background\_g | 从参数服务器中删除参数/turtlesim/background\_g |

## 节点

### rosnode：运行中节点检索与操作

·官方文档：<http://wiki.ros.org/rosnode>

|  |  |
| --- | --- |
| rosnode list | 列出所有运行中的节点 |
| rosnode info /turtlesim | 查看/turtlesim节点的详细信息，包括节点名称、发布的消息、订阅的消息、提供的服务、进程编号PID、输入输出端口及协议 |
| rosnode ping /turtlesim | 测试能否连接到/turtlesim节点 |
| rosnode cleanup | 清除当前所有无法连接节点Unreachable Nodes的注册信息（无法连接的节点常为在命令行已经结束运行，但注册信息残存的节点） |
| rosnode kill /turtlesim | 关闭节点/turtlesim |
| rosnode machine my\_laptop | 查看计算机my\_laptop上运行的节点，用于多机器ROS系统 |

## 话题与消息

### rostopic：活跃话题检索与操作

·官方文档：<http://wiki.ros.org/rostopic>

|  |  |
| --- | --- |
| rostopic list | 列出系统中所有活跃话题 |
| rostopic info /turtle1/cmd\_vel | 查看/turtle1/cmd\_vel话题的详细信息，包括话题消息类型、发布者、订阅者 |
| rostopic type /turtle1/cmd\_vel | 查看/turtle1/cmd\_vel话题的消息类型 |
| rostopic find geometry\_msgs/Twist | 搜索使用了geometry\_msgs/Twist消息类型的活跃话题 |
| rostopic echo /turtle1/pose | 在命令行输出/turtle1/pose话题的消息（该命令会新创建一个订阅/turtle1/pose话题的节点） |
| rostopic hz /turtle1/pose | 输出/turtle1/pose话题的消息发布频率（该命令会新创建一个订阅/turtle1/pose话题的节点） |
| rostopic pub -r 2 /turtle1/cmd\_vel geometry\_msgs/Twist -- '[2.0, 0.0, 0.0]' '[0.0, 0.0, 1.8]' | 向/turtle1/pose话题发布指定参数的geometry\_msgs/Twist消息，且发布频率为2Hz（消息的类型必须与话题匹配） |

### rosmsg：消息类型查询

·官方文档： <http://wiki.ros.org/rosmsg>

|  |  |
| --- | --- |
| rosmsg list | 列出所有软件包中所有定义的消息 |
| rosmsg packages | 列出所有定义了消息（含有.msg）的软件包 |
| rosmsg info/show geometry\_msgs/Twist | 输出geometry\_msgs/Twist消息的具体定义，用info和show的效果是一样的，若包含嵌套的定义，则会用逐级缩进表示 |
| rosmsg package mavros\_msgs | 列出mavros\_msgs软件包中定义的所有消息 |
| rosmsg md5 mavros\_msgs/State | （高级功能）显示mavros\_msgs/State的md5sum码 |

## 服务

### rosservice：活跃服务检索与操作

·官方文档： <http://wiki.ros.org/rosservice>

|  |  |
| --- | --- |
| rosservice list | 列出当前系统中所有活跃的服务 |
| rosservice find tf/FrameGraph | 列出所有正在使用服务类型tf/FrameGraph的活跃服务，服务类型的查找要用到下一节的rossrv |
| rosservice info /turtle1/set\_pen | 显示/turtle1/set\_pen服务的详细信息（提供服务的节点、URI码、服务类型、调用所需参数），这个命令囊括了下面三个命令的功能 |
| rosservice type /turtle1/set\_pen | 显示/turtle1/set\_pen服务的服务类型 |
| rosservice args /turtle1/set\_pen | 显示/turtle1/set\_pen服务调用所需的输入参数，返回值显示/turtle1/set\_pen接受五个参数r、g、b、width、off |
| rosservice uri /turtle1/set\_pen | 显示/turtle1/set\_pen服务的URI码 |
| rosservice call /turtle1/set\_pen 34 240 190 20 False | 请求调用/turtle1/set\_pen服务，该服务用于设置turtle1的移动痕迹外观。设置rgb颜色值分别为34、240、190，线宽width = 20，不关闭痕迹off = False |

### rossrv：服务类型查询

·官方文档： <https://wiki.ros.org/action/show/rosmsg#rossrv>

rossrv和消息类型查询命令rosmsg使用方法基本相同。

|  |  |
| --- | --- |
| rossrv list | 列出所有软件包中所有定义的服务类型 |
| rossrv packages | 列出所有定义了服务类型（含有.srv）的软件包 |
| rossrv info/show turtlesim/Spawn | 输出turtlesim/Spawn服务的具体定义，包含输入参数和打印到命令行的输出。用info和show的效果是一样的 |
| rossrv package mavros\_msgs | 列出mavros\_msgs软件包中定义的所有服务类型 |
| rossrv md5 mavros\_msgs/State | （高级功能）显示mavros\_msgs/State的md5sum码 |

## 调试命令

### roswtf：系统纠错

·官方文档：<https://wiki.ros.org/roswtf>

# 无人机接口：MAVROS

·ROS官网上的MAVROS文档：<https://wiki.ros.org/mavros>

无人机的Offboard离线模式需要机载电脑上ROS控制程序发送命令才能成功运行，ROS环境与飞控模块之间的桥梁就是MAVROS，这是由MAVLink官方开发的ROS系列软件包。MAVROS的下载方法参见：[这一节](#_安装MAVROS)

严格意义上，MAVROS由三个软件包mavros、mavros\_msgs、mavros\_extras构成：

1. mavros：MAVROS的主节点和插件
2. mavros\_msgs：MAVROS的用到的消息和服务类型定义
3. mavros\_extras：一些和机器视觉有关的节点和插件

## 

# RealSense相机接口：Realsense2

# 实践：Turtlesim小乌龟移动控制



## 相关节点

# 实践：MAVROS无人机SITL



·本教程基于Ubuntu 20.04虚拟机 + ROS Noetic环境 + PX4 1.12.1

无人机的软件在环仿真SITL指的是不涉及实际运行硬件（飞控模块）的仿真，主要用于测试控制算法的可行性；与之相对的概念是硬件在环仿真HITL，用于测试算法在飞控模块上的运行情况。

在ROS环境下，通过PX4官方源码编译出的，基于Gazebo的仿真器和MAVROS配合可以进行SITL和HITL仿真，这里以SITL为例。

## 安装MAVROS

·MAVROS官方文档：<http://wiki.ros.org/mavros>

·PX4官方的配置教程：<https://docs.px4.io/main/en/ros/mavros_installation.html>

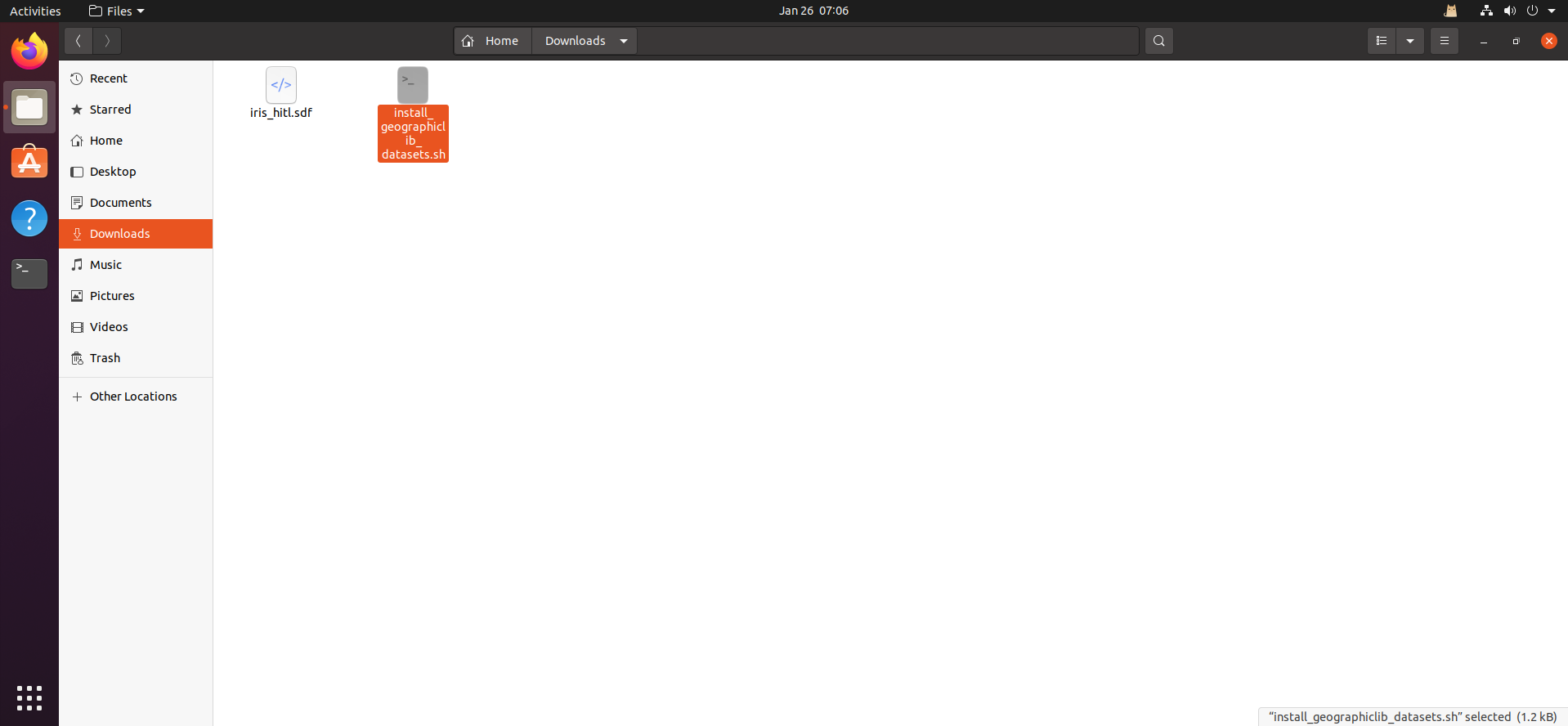
MAVROS是MAVLink官方开发的ROS功能包，用于建立ROS系统与飞控模块之间的连接，它定义了无人机控制、调试相关的一系列节点、消息和服务。

ROS环境的安装并不包含MAVROS功能包，因此需要输入以下命令手动下载，其中的“neotic”要替换成使用的ROS发行版的名字

|  |
| --- |
| sudo apt install ros-neotic-mavros ros-neotic-mavros-extras ros- neotic-mavros-msgs |

另外，需要执行下面的两行代码安装一个MAVROS依赖的地理计算相关的数据库GeographicLib Dataset

|  |
| --- |
| wget https://raw.githubusercontent.com/mavlink/mavros/master/mavros/scripts/install\_geographiclib\_datasets.sh  sudo bash ./install\_geographiclib\_datasets.sh |



## 下载PX4源码

·PX4源码Github页面：<https://github.com/PX4/PX4-Autopilot>

PX4的源码一定要通过Git克隆的方式下载，如果直接下载压缩包，PX4的文件夹里面会缺少Git相关的文件，导致后续的编译失败。

另外，因为众所周知的原因，从Github上直接拉取代码经常会出现连接失败的情况，因此在Ubuntu系统里要提前装好梯子，安装参考下面的教程：

·Clash的配置：<http://confluence.mmcuav.com:8002/pages/viewpage.action?pageId=44695559>

首先，先运行下面的指令安装Git：

|  |
| --- |
| sudo apt install git |

运行下面的指令拉取Github上的PX4仓库。加上“--recursive”是为了同时下载所有的依赖项和子模块。包含所有子模块的PX4-Autopilot文件夹约为1.9GB。

|  |
| --- |
| git clone --recursive https://github.com/PX4/PX4-Autopilot.git |

这里的话，考虑到兼容性问题，个人建议不要下载最新版的PX4源码，下载版本和飞控烧录的固件版本一致就行，我用的是1.12.1。下载指定版本的PX4仓库命令如下：

|  |
| --- |
| git clone --recursive <https://github.com/PX4/PX4-Autopilot.git> --branch --branch v1.12.1 |

下载完成后，在PX4-Autopilot文件夹中打开命令行，输入以下代码用bash运行ubuntu.sh配置文件。（注意，文件ubuntu.sh在PX4-Autopilot文件夹中的具体路径因具体的PX4版本而异，可使用“find -name ubuntu.sh”文件查找它的具体位置）

|  |
| --- |
| bash ./Tools/setup/ubuntu.sh |

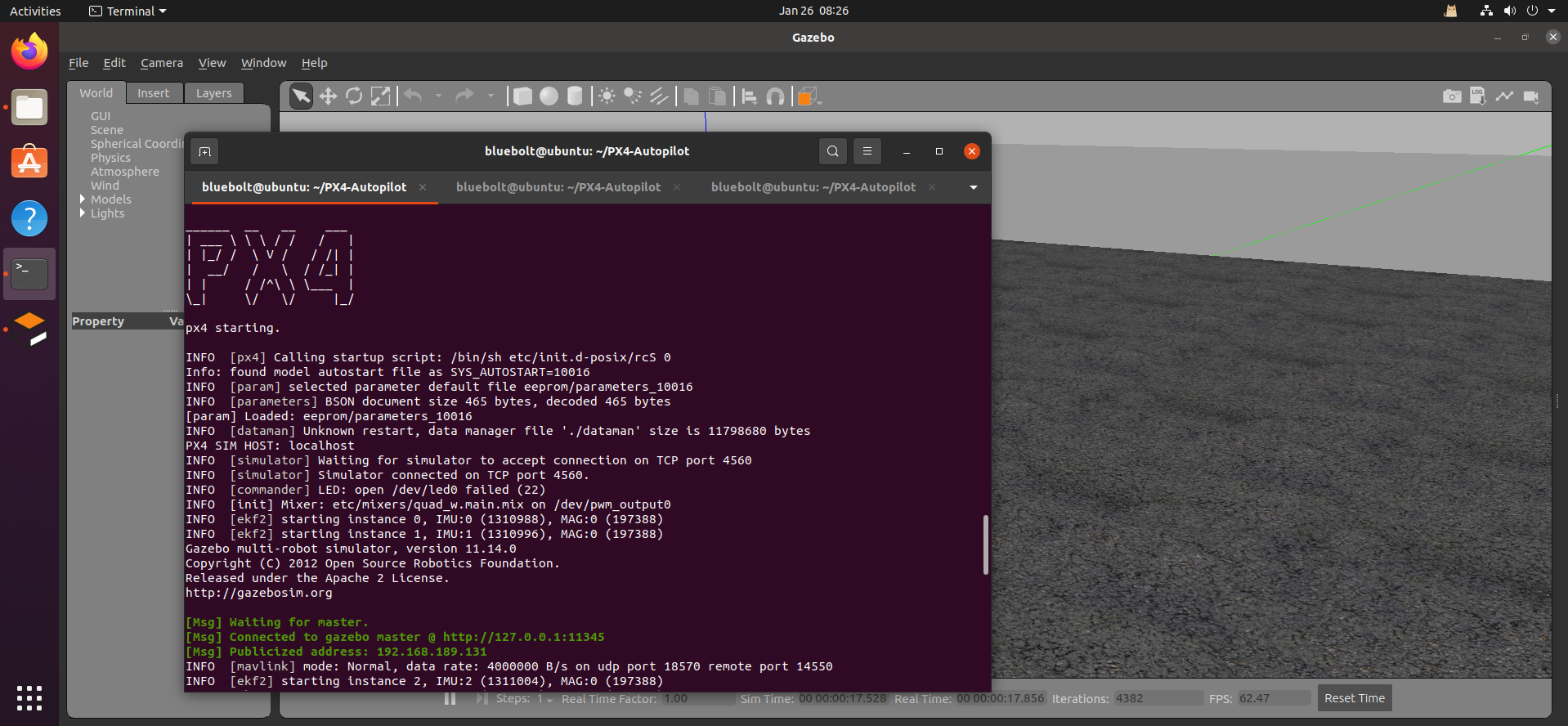
## 编译PX4 SITL仿真模型

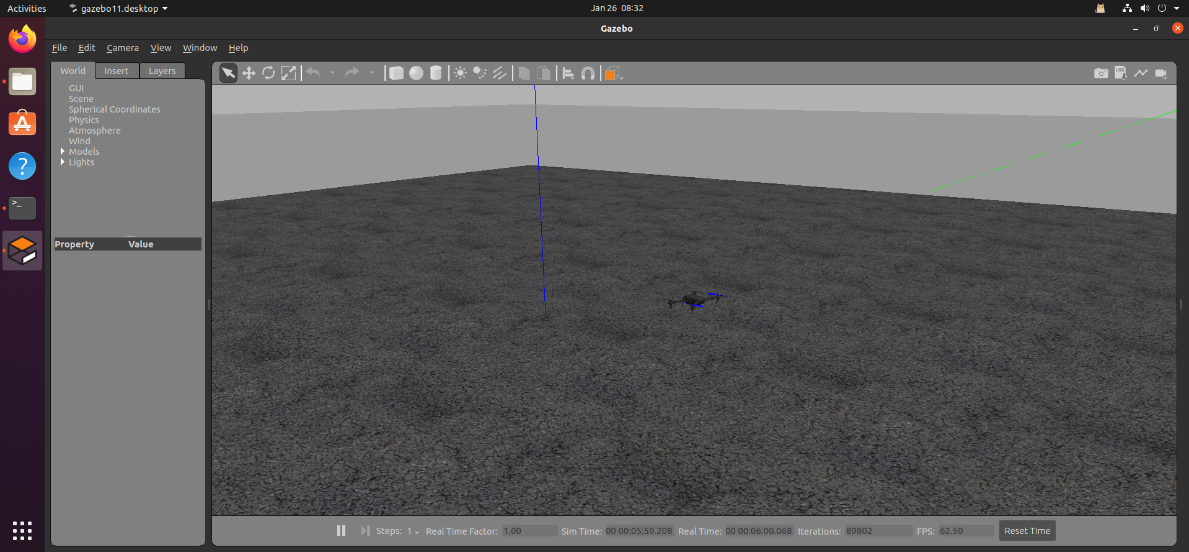
在PX4-Autopilot文件夹中打开命令行，输入以下代码：

|  |
| --- |
| make px4\_sitl gazebo |

等待编译结束后，会自动打开ROS自带的Gazebo仿真器，里面可以看到一架无人机，同时命令行出现PX4的Logo和仿真模型的URL地址。

下图中绿色字体显示的URL地址为“@http://127.0.0.1:11345”，记下这个地址，后面启动MAVROS需要用到。





## 安装QGroundControl

·QGC安装教程：

<https://docs.qgroundcontrol.com/master/en/qgc-user-guide/getting_started/download_and_install.html>

使用QGroundControl是为了方便看到无人机飞行的轨迹。

## 编写控制程序（C++）

·PX4官方C++离线控制例程：<https://docs.px4.io/main/en/ros/mavros_offboard_cpp.html>

创建名为mavros\_ws的工作空间，在工作空间内创建功能包ofb\_test，然后创建ofb\_node\_circle.cpp文件，编译成名为ofb\_node\_circle的节点。

## 运行仿真

·启动PX4模式和Gazebo仿真器：

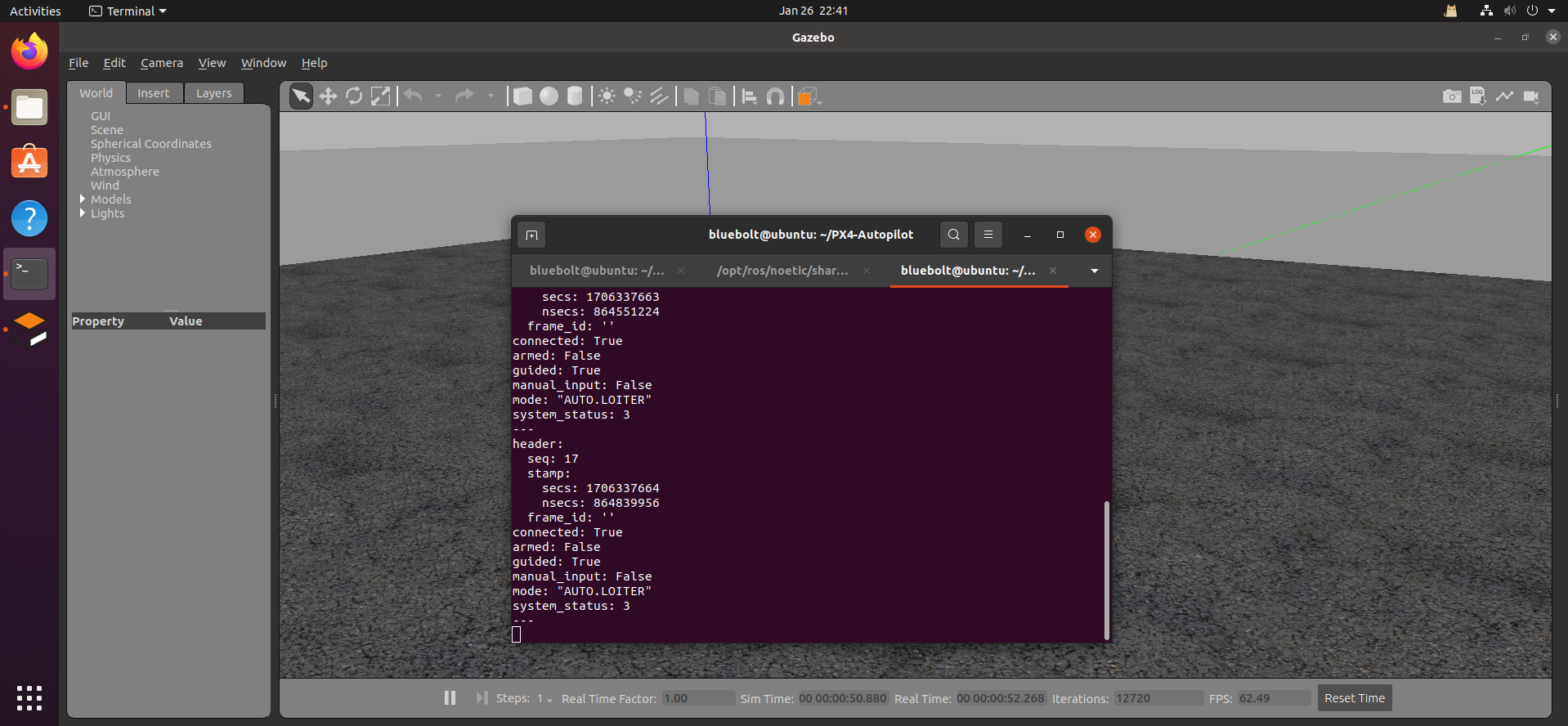
|  |
| --- |
| cd ./PX4-Autopilot  make px4\_sitl gazebo |

·启动MAVROS节点：

注意URL地址要换成PX4仿真器启动时显示的地址。

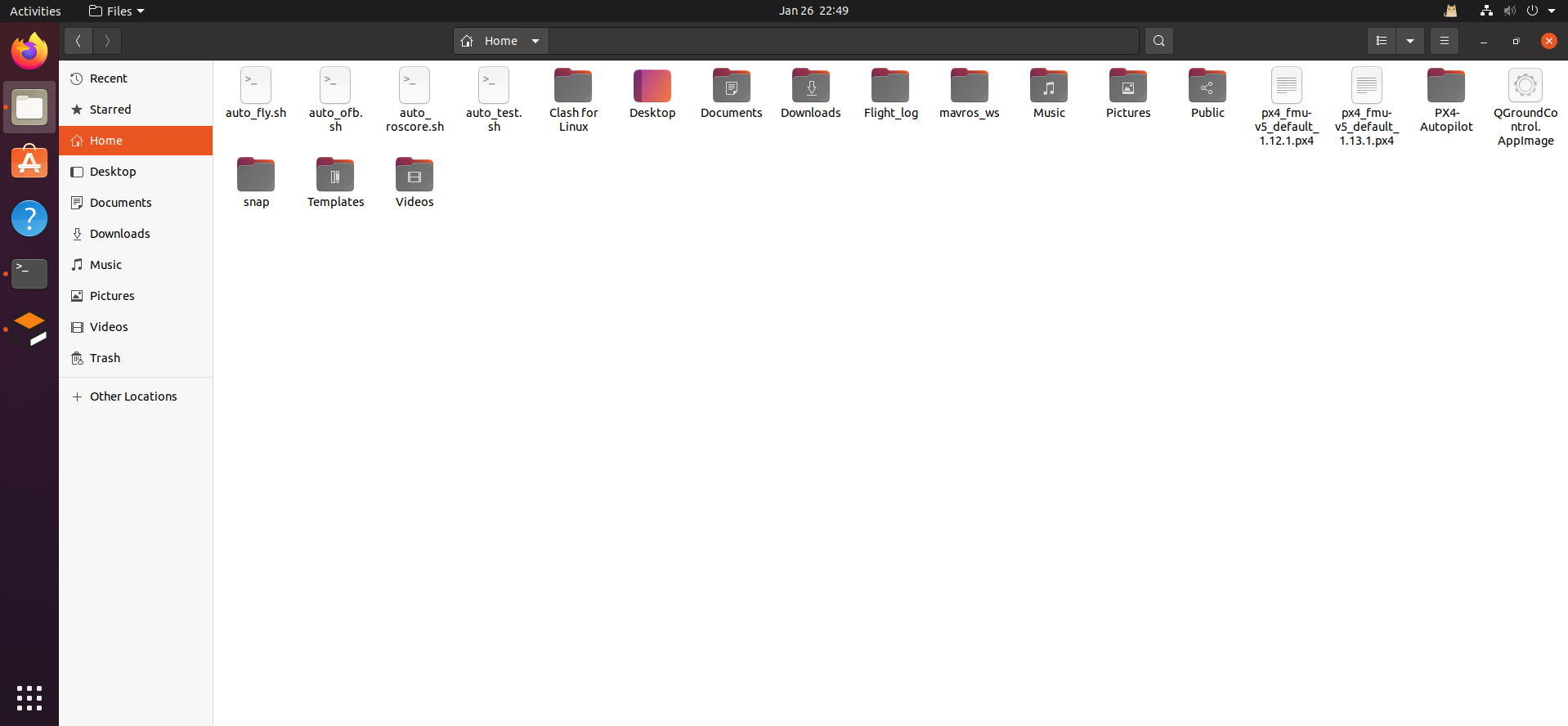
|  |
| --- |
| roslaunch mavros px4.launch fcu\_url:="udp://:14540@127.0.0.1:11345" |

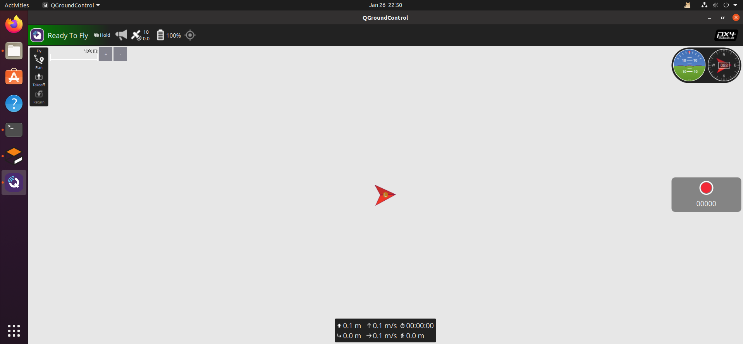
订阅/mavros/state话题，测试MAVROS是否成功连接到PX4仿真器，若命令行返回信息中显示connected: True，则表示连接成功。



·启动地面站QGroundControl：

直接点击图标即可启动QGC，启动后QGC主页面显示“Really to fly”，地图上显示飞机图标，表示QGC已经连接上模拟飞机。



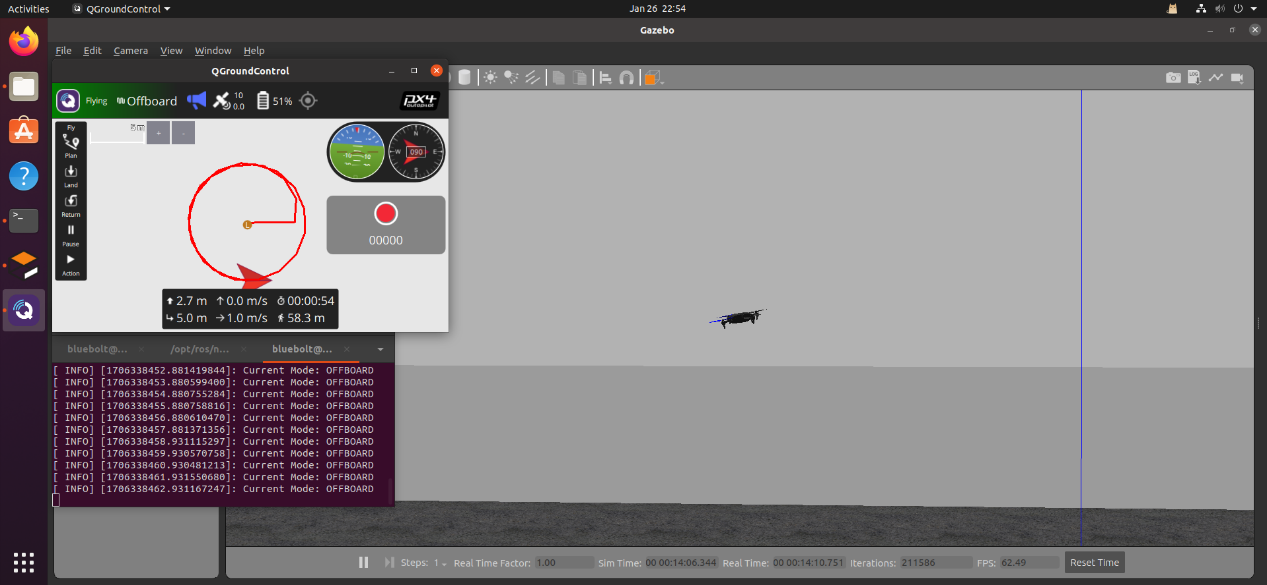


·启动自己编写的控制节点ofb\_node\_circle：

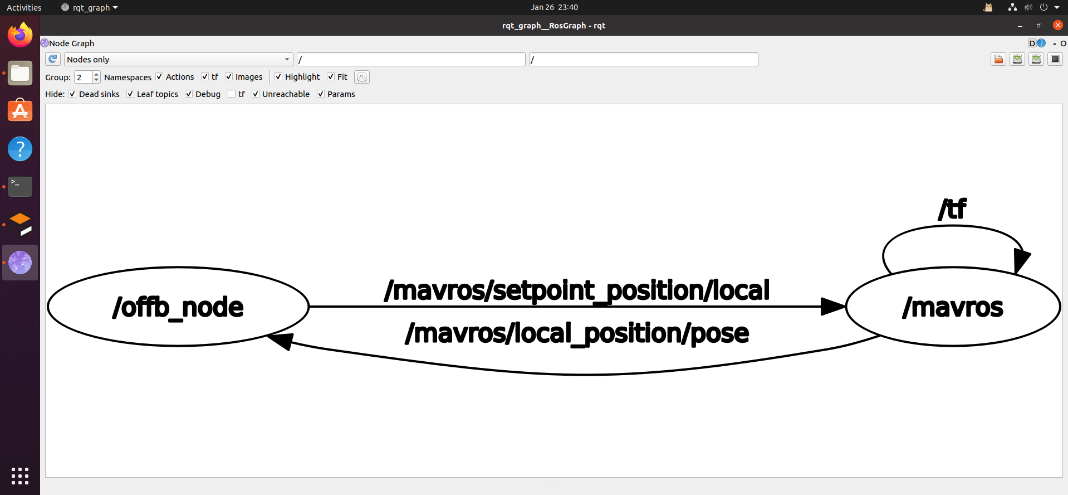
|  |
| --- |
| rosrun ofb\_test ofb\_node\_circle |

## 仿真效果

启动控制节点后，输出显示现在是Offboard模式，可以观察到Gazebo中的无人机解锁并起飞，QGC中显示无人机的飞行轨迹是正圆形。



在Rqt Graph中可以看到控制节点“/offb\_node”和MAVROS节点“/mavros”之前通过两个话题构成一个闭环的控制系统：



# 参考资料

1. ROS官方教程：

<http://wiki.ros.org/ROS/Tutorials>

1. 古月居ROS入门21讲：

<https://www.bilibili.com/video/BV1zt411G7Vn/?vd_source=d208713af4296ed6230d640d411a0ebf>

1. PX4官方文档：

<https://docs.px4.io/main/en/>