

ELEG 4701: Intelligent Interactive Robot Practice

Lab 3: Introduction to ROS

KONG Detian

1155156921@link.cuhk.edu.hk

20 Sep, 2021

Outline

1. What is ROS
2. Install ROS on Ubuntu
3. Core concepts in ROS
4. Use of command line tools
5. Workspace and package
6. Programming practice

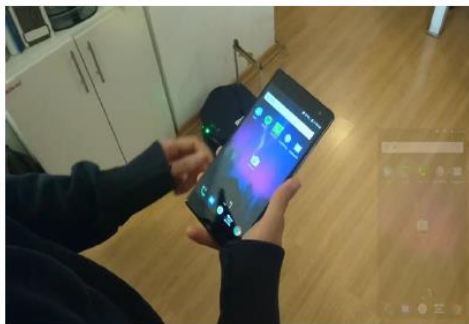
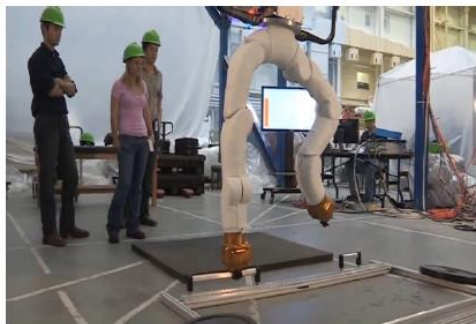
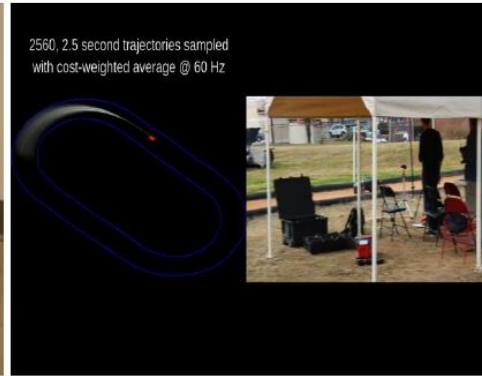
What is ROS

The history of ROS

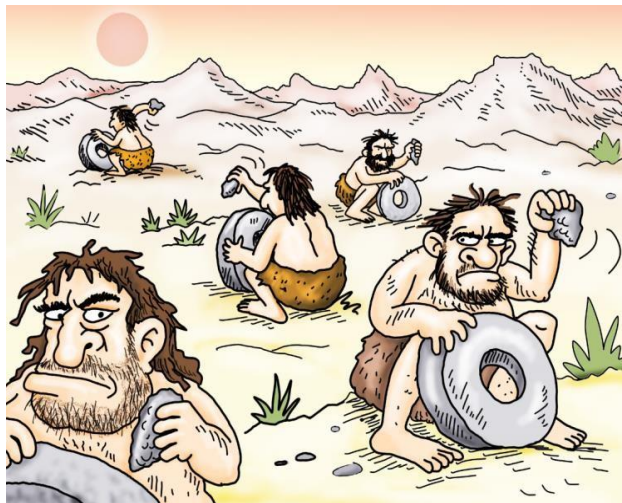
Distro	Release date	Poster	Tuturtle, turtle in tutorial	EOL date
ROS Noetic Ninjemys (Recommended)	May 23rd, 2020			May, 2025 (Focal EOL)
ROS Melodic Morenia	May 23rd, 2018			May, 2023 (Bionic EOL)
ROS Lunar Loggerhead	May 23rd, 2017			May, 2019
ROS Kinetic Kame	May 23rd, 2016			April, 2021 (Xenial EOL)
ROS Jade Turtle	May 23rd, 2015			May, 2017
ROS Indigo Igloo	July 22nd, 2014			April, 2019 (Trusty EOL)

ROS Hydro Medusa	September 4th, 2013			May, 2015
ROS Groovy Galapagos	December 31, 2012			July, 2014
ROS Fuerte Turtle	April 23, 2012			--
ROS Electric Emys	August 30, 2011			--
ROS Diamondback	March 2, 2011			--
ROS C Turtle	August 2, 2010			--
ROS Box Turtle	March 2, 2010			--

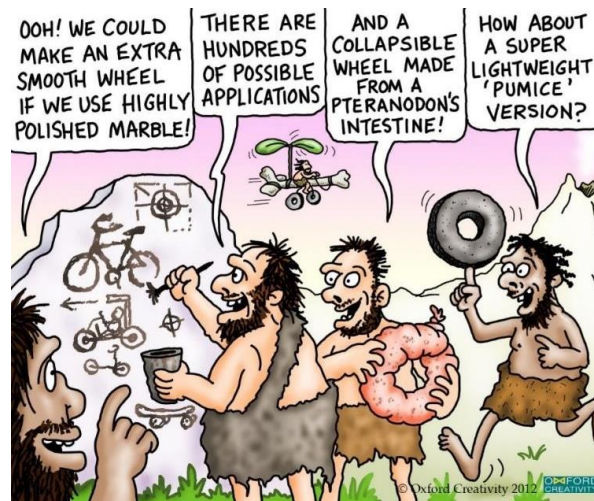
Application of ROS



What is ROS



Legacy Mode

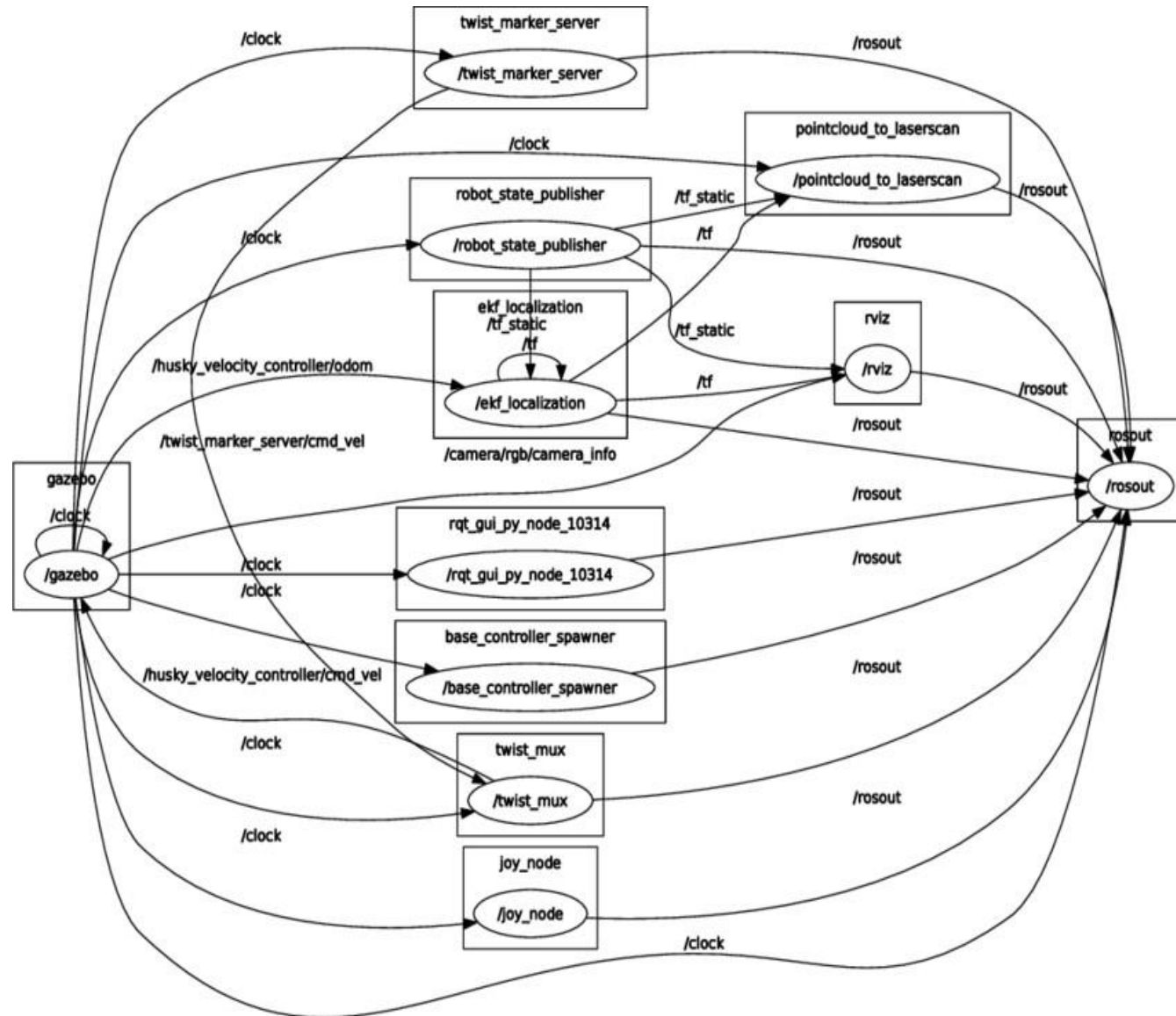


Modern Mode

Improve the software reuse rate in robotics research and development

Communication

Loosely coupled
distributed
communication



Development tools in ROS

WORKSPACES

Create Workspace

```
mkdir catkin_ws && cd catkin_ws
wstool init src
catkin_make
source devel/setup.bash
```

Add Repo to Workspace

```
roscd; cd ../src
wstool set repo_name \
--git http://github.com/org/repo_name.git \
--version=kinetic-devel
wstool up
```

Resolve Dependencies in Workspace

```
sudo rosdep init # only once
rosdep update
rosdep install --from-paths src --ignore-src \
--rosdistro=${ROS_DISTRO} -y
```

PACKAGES

Create a Package

```
catkin_create_pkg package_name [dependencies ...]
```

Package Folders

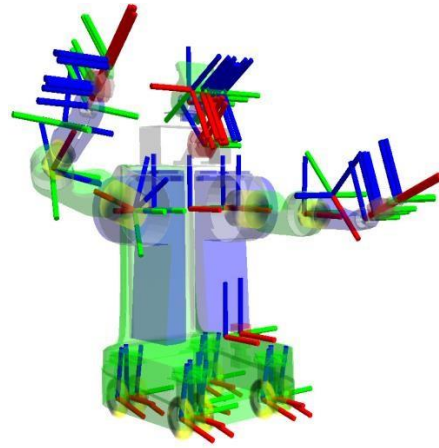
include/package_name	C++ header files
src	Source files. Python libraries in subdirectories
scripts	Python nodes and scripts
msg, srv, action	Message, Service, and Action definitions

Release Repo Packages

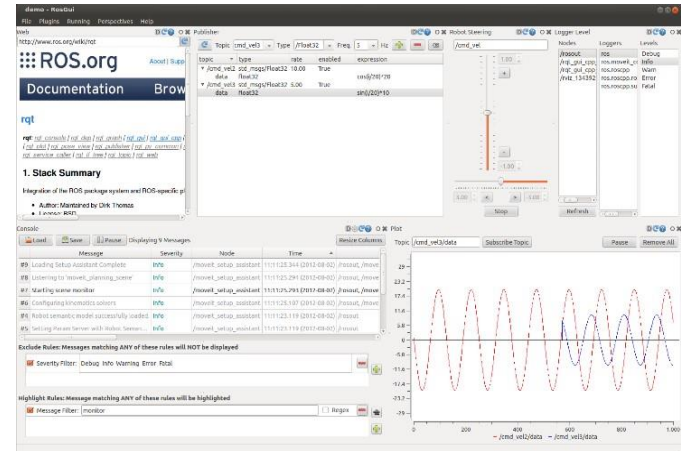
```
catkin_generate_changelog
# review & commit changelogs
catkin_prepare_release
bloom-release --track kinetic --ros-distro kinetic repo_name
```

Reminders

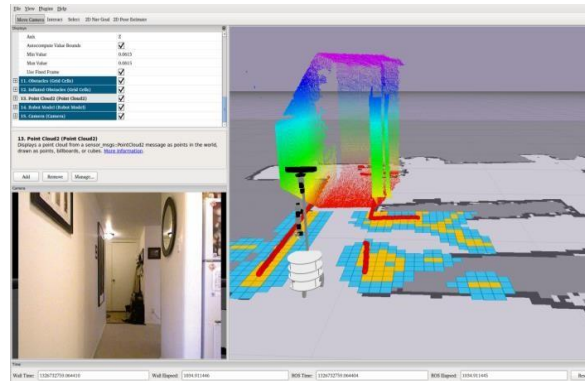
- Testable logic
- Publish diagnostics
- Desktop dependencies in a separate package



TF coordinate transformation



QT Toolbox



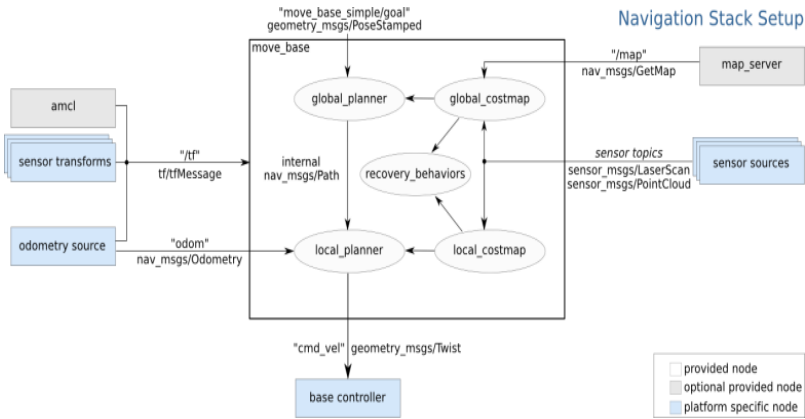
Rviz



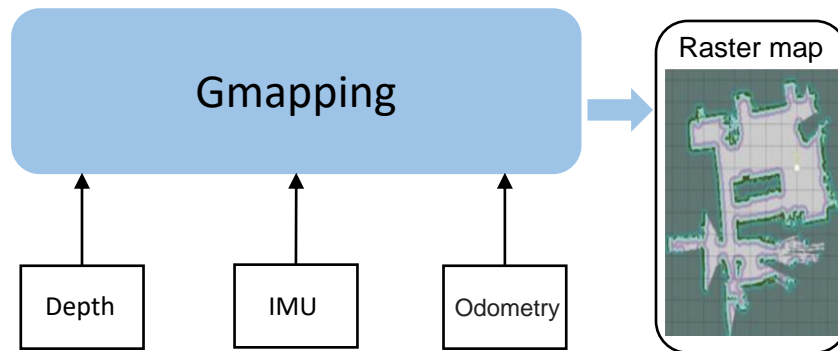
Gazebo

Command line and compiler

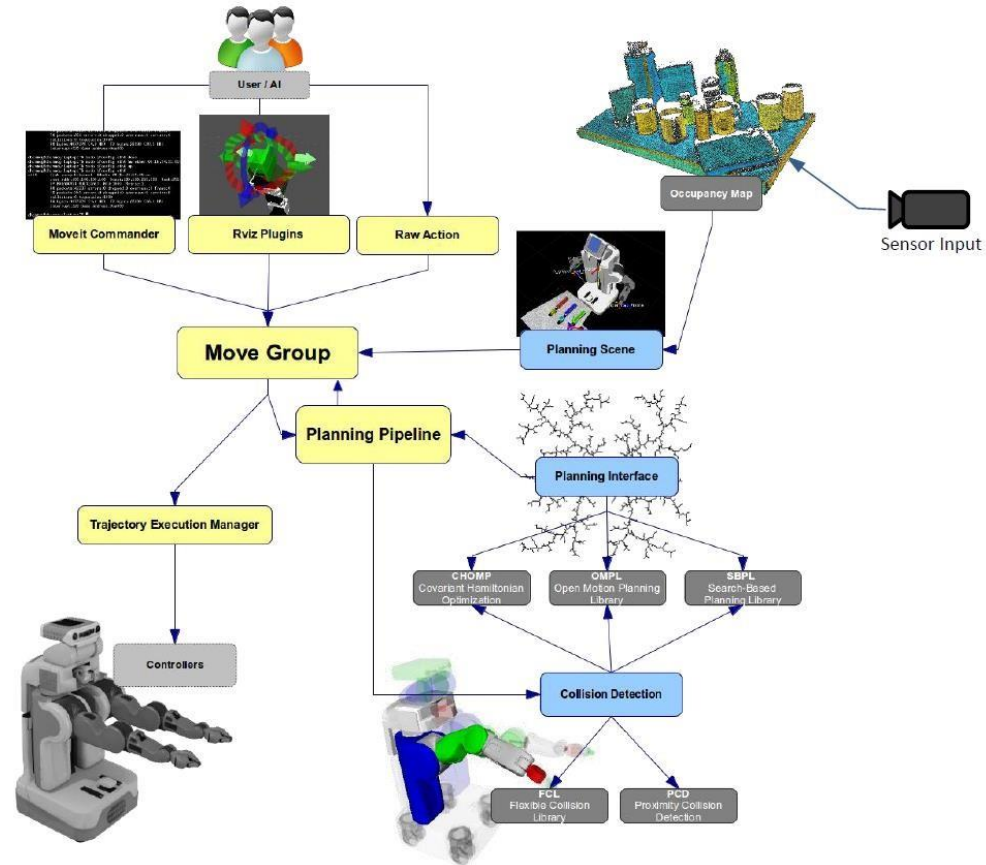
Applications



Navigation



SLAM



MoveIt

Ecosystem of ROS

1. Distribution: ROS distribution includes a series of feature packages with version numbers that can be installed directly

2. Repository: ROS relies on opensource code on a shared network, and different organizations can develop or share their own robotics software

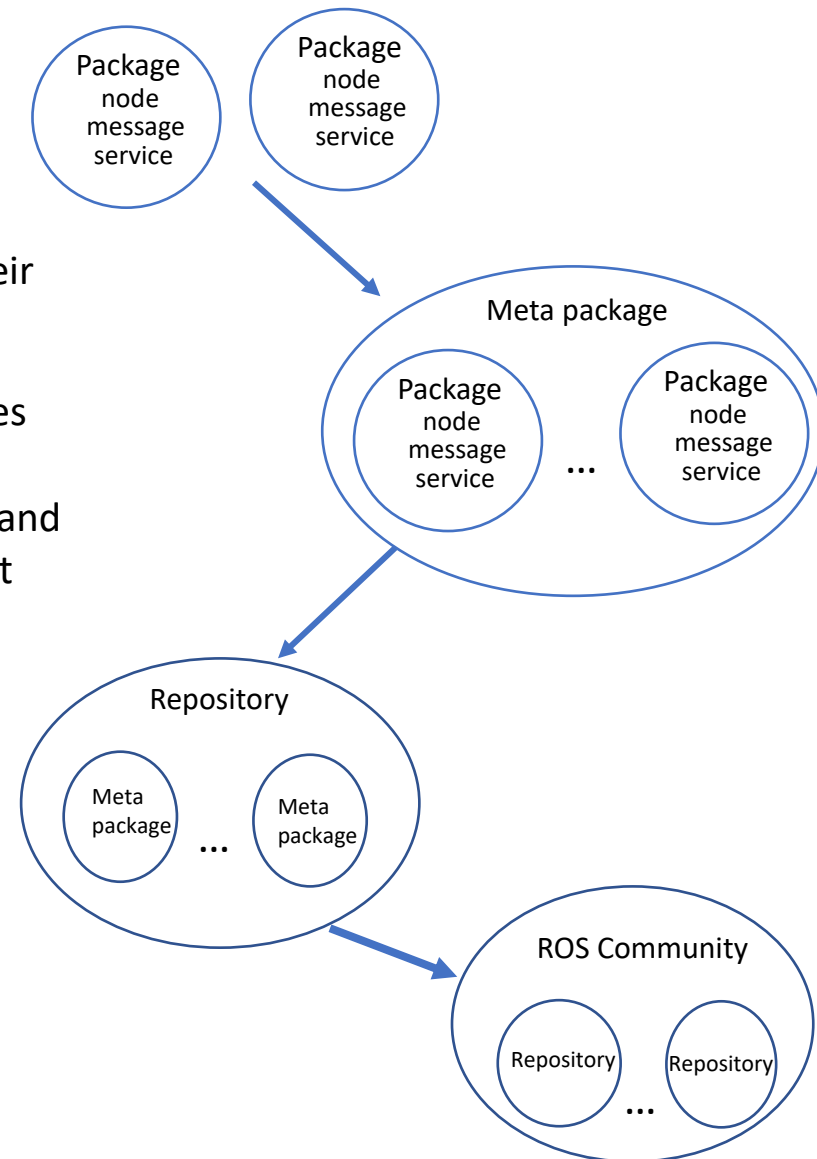
3. ROS wiki: The main forum for recording ROS information files

4. Mailing list: The main channel for exchanging ROS updates, and also for exchanging various questions about ROS development

5. ROS answer: Website for consulting ROS related issues

6. Blog: Post news, pictures, videos from the ROS community

www.ros.org



Install ROS on Ubuntu

Choose one distribution version

kinetic

ROS Kinetic Kame

ROS Kinetic Kame is the tenth [ROS distribution release](#). It was released May 23rd, 2016.

目录

1. [ROS Kinetic Kame](#)
 1. [Platforms](#)
 2. [Installation](#)
 3. [Release Planning](#)
 4. [Changes](#)



1. Platforms

ROS Kinetic Kame is primarily targeted at the Ubuntu 16.04 (Xenial) release, though other Linux systems as well as Mac OS X, Android, and Windows are supported to varying degrees. For more information on compatibility on other platforms, please see [REP 3: Target Platforms](#). It will also support Ubuntu 15.10 Wily and Debian Jessie.

2. Installation

Please see the [installation instructions](#). There are binary packages available for Ubuntu distributions Wily and Xenial for x86, x86_64, and armhf architectures.

3. Release Planning

See [Planning](#)

4. Changes

To get a better idea of the parts of ROS which have been changed in ROS Kinetic, please look at the [ROS Kinetic Migration](#) page.

ROS 2 Documentation

The ROS Wiki is for ROS 1. Are you using ROS 2 (Dashing/Foxy/Rolling)? [Check out the ROS 2 Documentation](#)

维基

[Distributions](#)
[ROS/Installation](#)
[ROS/Tutorials](#)
[RecentChanges](#)
[kinetic](#)

网页

[只读网页](#)

[信息](#)

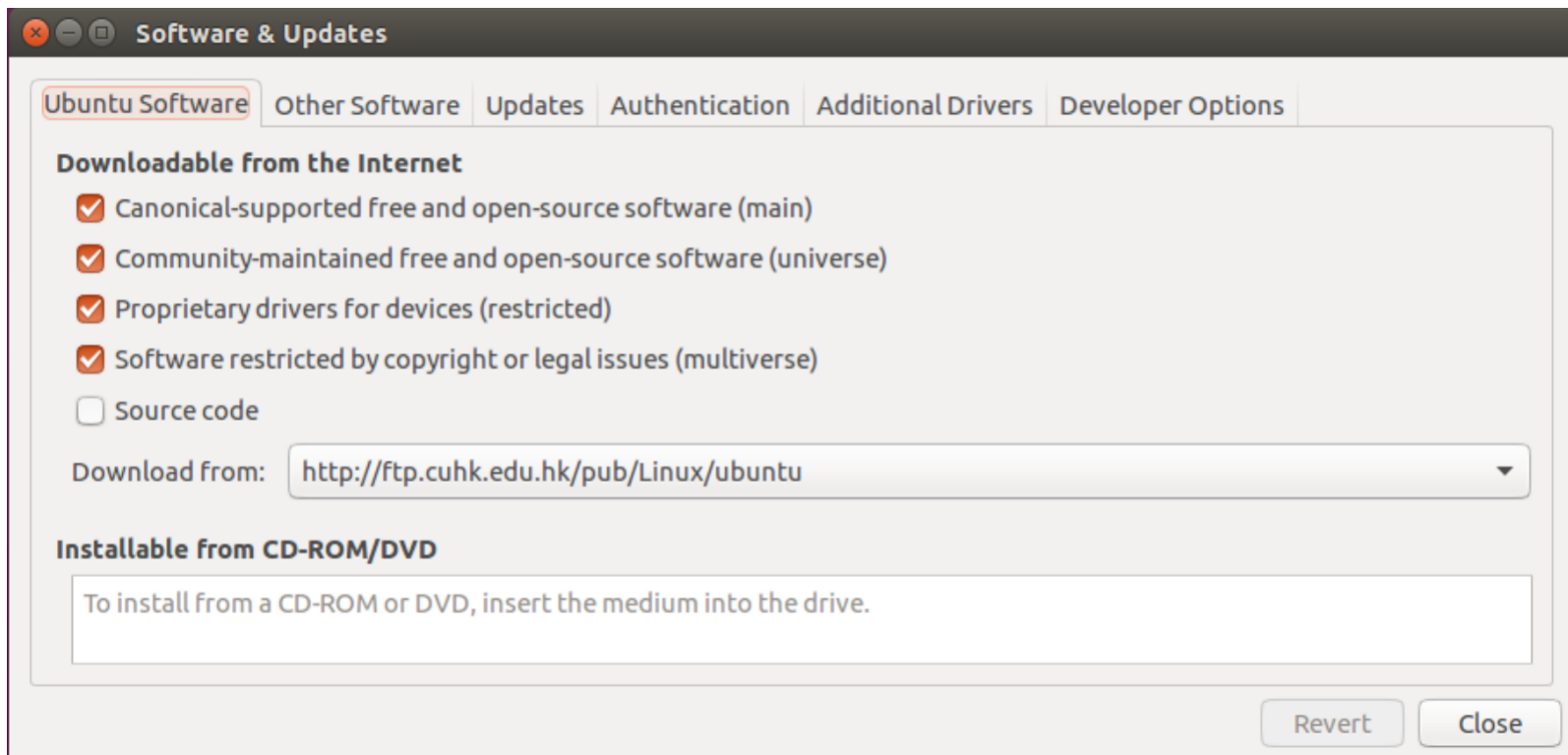
[附件](#)

更多操作:

用户

[登录](#)

Configure your Ubuntu repositories



Install ROS on Ubuntu

1. Setup your sources.list

```
sudo apt install python-rosdep python-rosinstall python-rosinstall-generator python-wstool build-essential
```

2. Set up your keys

```
sudo apt install curl # if you haven't already installed curl  
curl -s https://raw.githubusercontent.com/ros/rosdistro/master/ros.asc | sudo apt-key add -
```

3. Installation

```
sudo apt-get update  
sudo apt-get install ros-kinetic-desktop-full
```

4. Environment setup

```
echo "source /opt/ros/kinetic/setup.bash" >> ~/.bashrc  
source ~/.bashrc
```

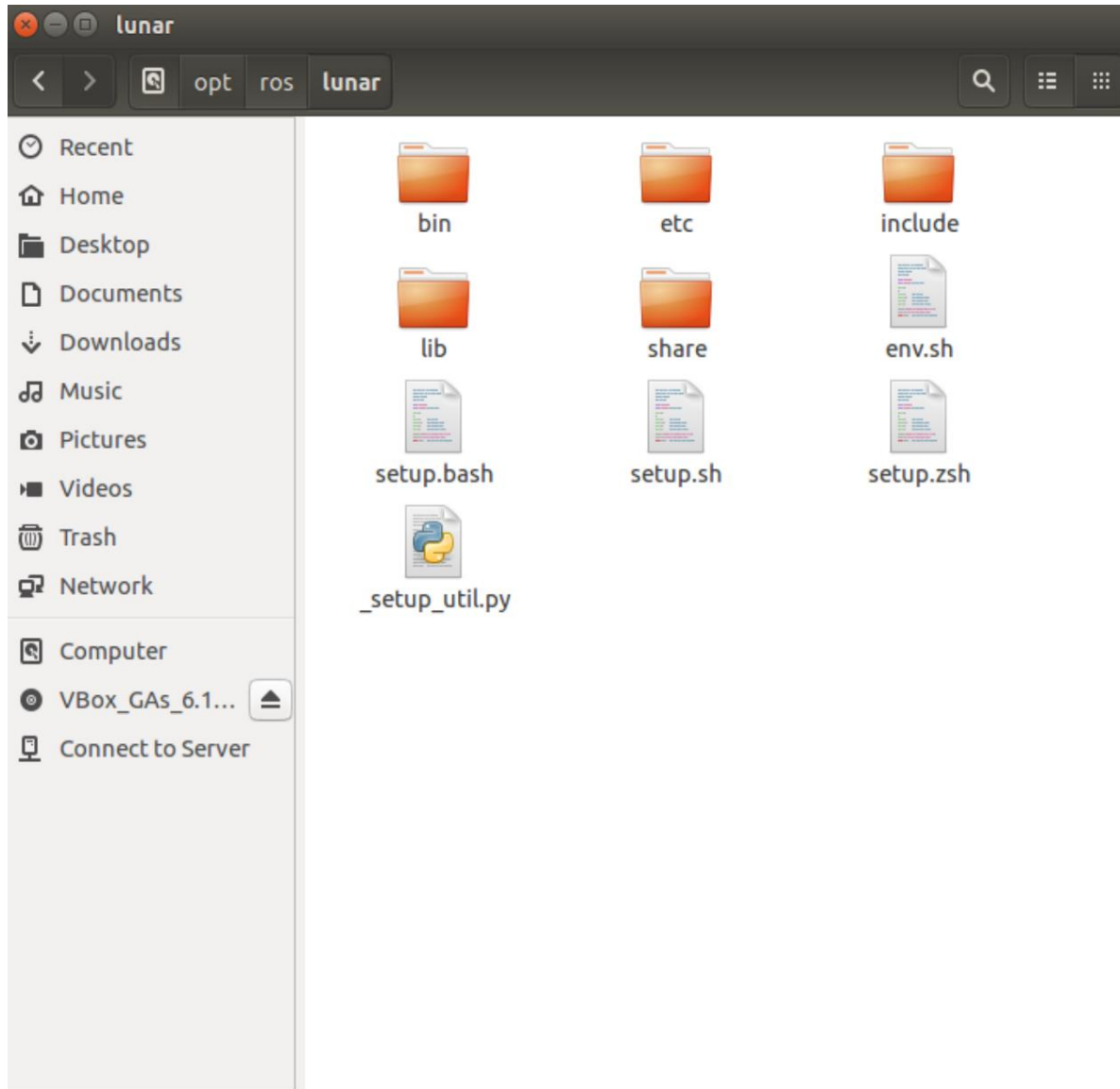
5. Dependencies for building packages

```
sudo apt install python-rosdep python-rosinstall python-rosinstall-generator python-wstool build-essential
```

6. Initialize rosdep

```
sudo rosdep init  
rosdep update
```

Install ROS on Ubuntu



Install ROS on Ubuntu

Run ROS Master



Run Turtlesim

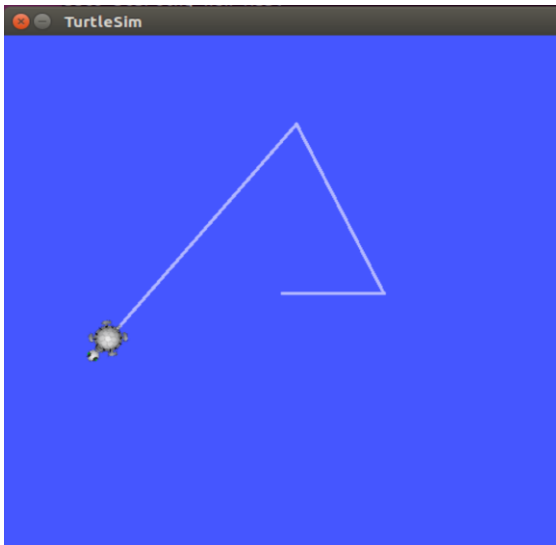


Run Turtlesim's control node

```
$ roscore
```

```
$ rosrun turtlesim turtlesim_node
```

```
$ rosrun turtlesim turtle_teleop_key
```



```
eleg@eleg-VirtualBox:~$ rosrun turtlesim turtlesim_node
[ INFO] [1631690510.668787374]: Starting turtlesim with node name /turtlesim
[ INFO] [1631690510.674501602]: Spawning turtle [turtle1] at x=[5.544445], y=[5.544445], theta=[0.000000]
XmbTextListToTextProperty result code -2
XmbTextListToTextProperty result code -2
XmbTextListToTextProperty result code -2
```

```
eleg@eleg-VirtualBox:~$ rosrun turtlesim turtle_teleop_key
Reading from keyboard
-----
Use arrow keys to move the turtle.
█
```

Core concepts in ROS

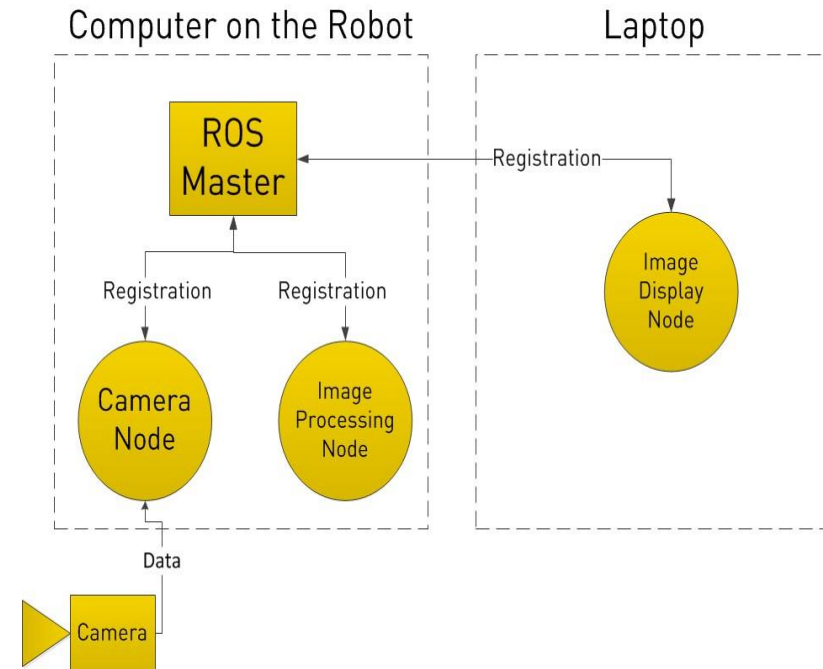
Core concepts in ROS

■ Node -- Execution Unit

- Processes that perform specific tasks, independently run executables
- Different nodes can use different programming languages and can be distributed to run on different hosts
- The name of the node must be unique in the system

■ ROS Master – Control center

- Provide naming and registration services for nodes
- Track and record topic/service communications to assist nodes in finding each other and establishing connections
- Provides a parameter server that nodes use to store and retrieve runtime parameters



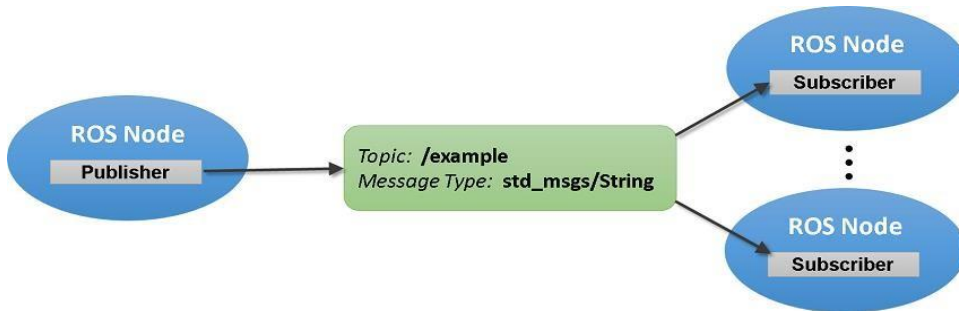
Core concepts in ROS

■ Topic -- Asynchronous communication

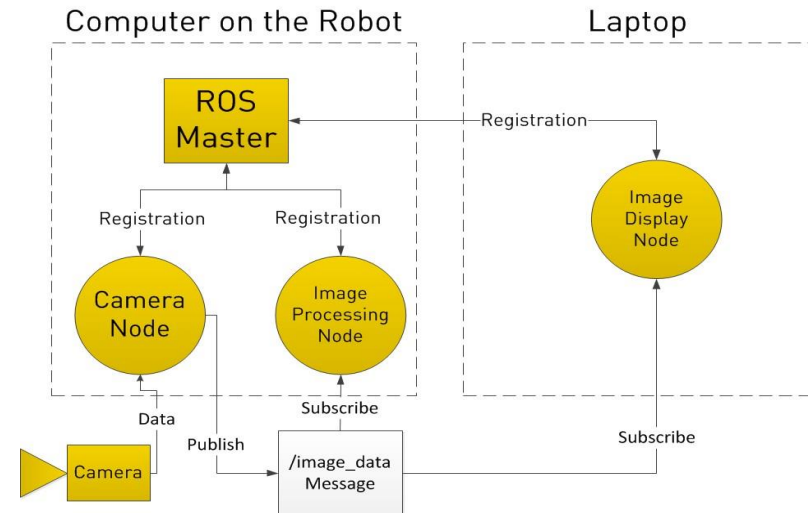
- Important bus used to transfer data between nodes
- Using the publish/subscribe model, data is transferred from publisher to subscriber, and publishers or subscribers of the same topic may not be unique

■ Message – Topic data

- Has certain types and data structures, including the standard types provided by ROS and user-defined types
- Use programming language-independent .msg file define message, the programming process generates the corresponding code files



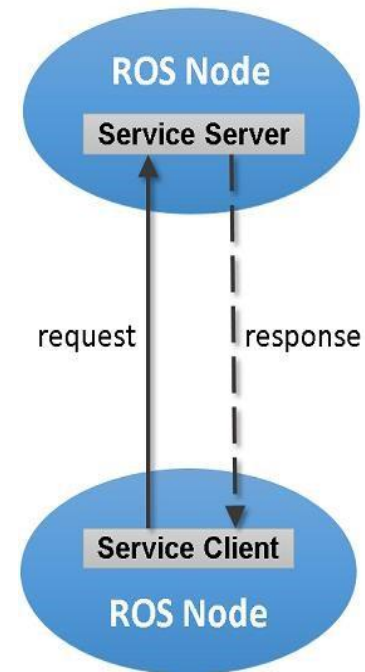
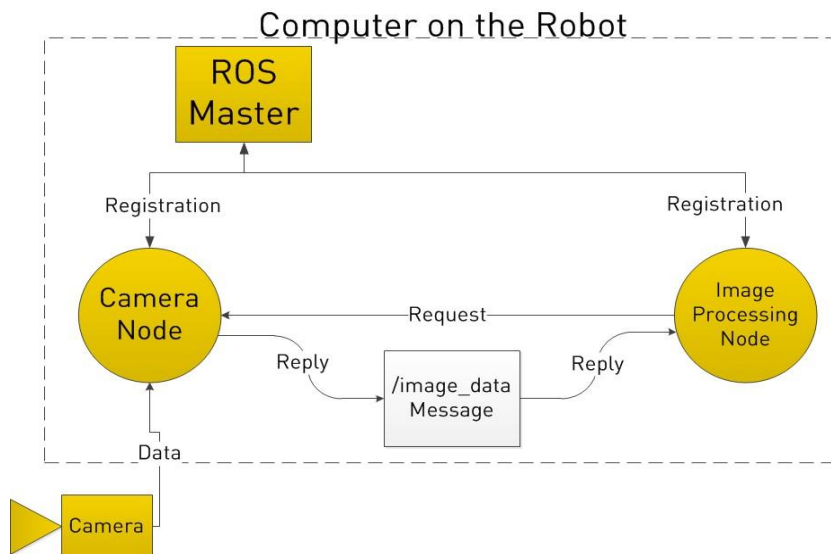
Topic Model(publish/subscribe)



Core concepts in ROS

■ Service -- Synchronous communication

- Using the client/server model(C/S), the client sends the request data, the server finishes processing and returns the answer data
- Different nodes can use different programming languages and can be distributed to run on different hosts
- Use programming language-independent .srv files to define request and response data structures, and generate the corresponding code files during the compile process



Service Model(request/response)

Core concepts in ROS

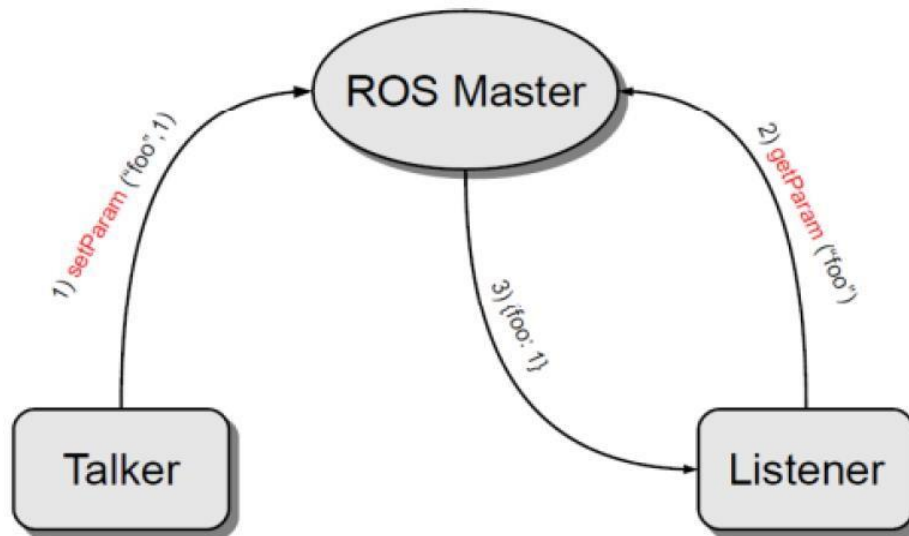
The difference between topics and services

	Topic	Service
Synchronicity	Asynchronous	Synchronous
Communication Model	Publish/subscribe	Request/response
Underlying protocols	ROSTCP/ROSUDP	ROSTCP/ROSUDP
Feedback mechanism	NO	YES
Buffer	YES	NO
Real-time	Weak	Strong
Node Relationships	Many-to-Many	One to many
Applicable scenarios	Data Transfer	Logic Processing

Core concepts in ROS

■ Parameter-- Global Shared Dictionary

- Net-accessible shared, multivariate dictionaries
- The node uses this server to store and retrieve runtime parameters
- Suitable for storing static, non-binary configuration parameters, not for storing dynamically configured data



Parameter Model(Global Shared Dictionary)

File System

■ Package

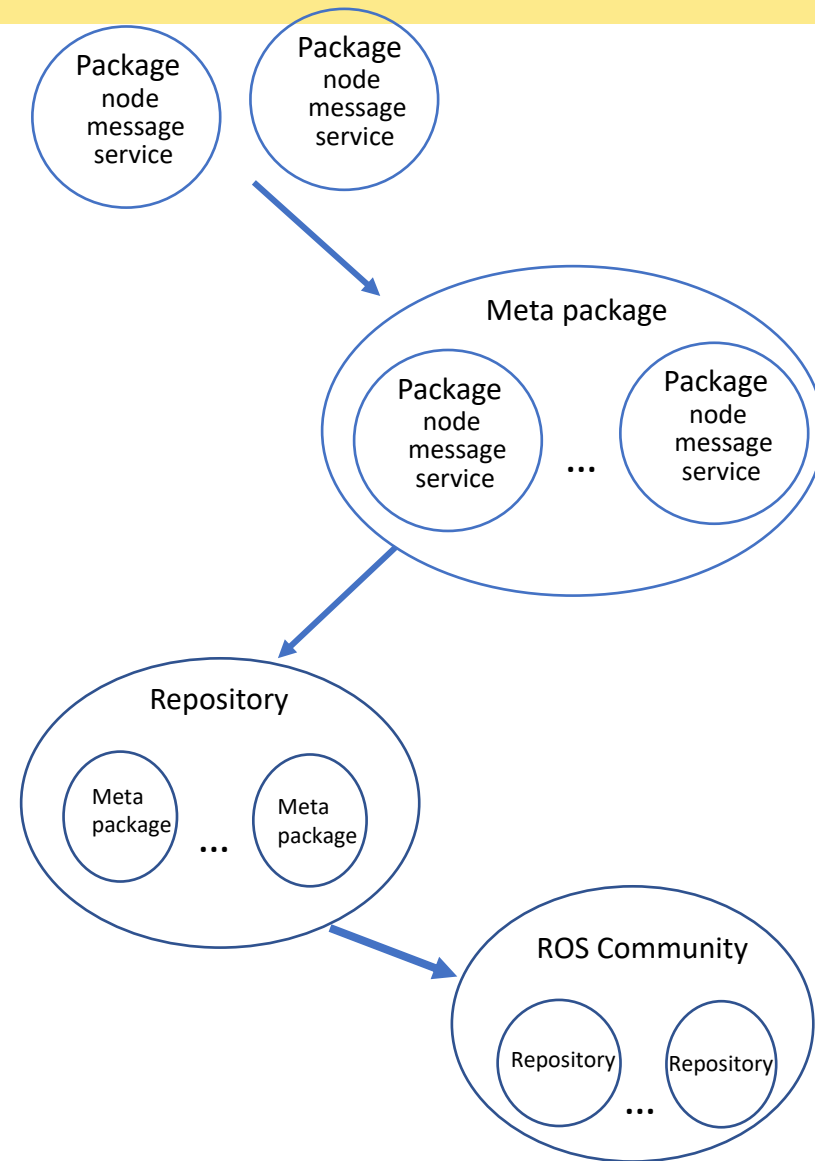
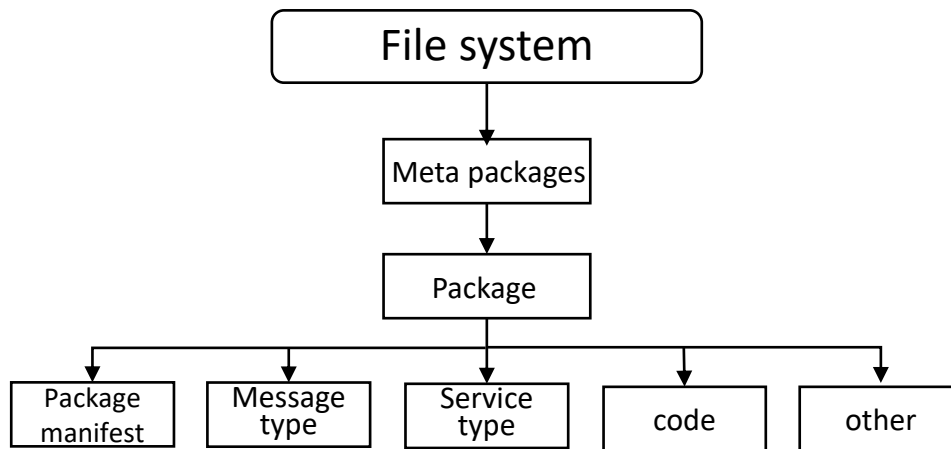
- The basic unit in ROS software, containing node source code, configuration files, data definitions

■ Package manifest

- Record basic information about the package, including author information, license information, dependency options, compilation flags, etc.

■ Meta Packages

- A collection of multiple functional packages for the same purpose



Use of command line tools

Use of command line tools

Common commands

- rostopic
- rosnod
- rosservice
- rosparm
- rosmg
- rossrv

WORKSPACES

Create Workspace

```
mkdir catkin_ws && cd catkin_ws
wstool init src
catkin_make
source devel/setup.bash
```

Add Repo to Workspace

```
roscd; cd ../src
wstool set repo_name \
--git http://github.com/org/repo_name.git \
--version=kinetic-devel
wstool up
```

Resolve Dependencies in Workspace

```
sudo rosdep init # only once
rosdep update
rosdep install --from-paths src --ignore-src \
--rosdistro=${ROS_DISTRO} -y
```

PACKAGES

Create a Package

```
catkin_create_pkg package_name [dependencies ...]
```

Package Folders

include/package_name	C++ header files
src	Source files. Python libraries in subdirectories
scripts	Python nodes and scripts
msg, srv, action	Message, Service, and Action definitions

Release Repo Packages

```
catkin_generate_changelog
# review & commit changelogs
catkin_prepare_release
bloom-release --track kinetic --ros-distro kinetic repo_name
```

Reminders

- Testable logic
- Publish diagnostics
- Desktop dependencies in a separate package

CMakeLists.txt

Skeleton

```
cmake_minimum_required(VERSION 2.8.3)
project(package_name)
find_package(catkin REQUIRED)
catkin_package()
```

Package Dependencies

To use headers or libraries in a package, or to use a package's exported CMake macros, express a build-time dependency:

```
find_package(catkin REQUIRED COMPONENTS roscpp)
```

Tell dependent packages what headers or libraries to pull in when your package is declared as a catkin component:

```
catkin_package(
  INCLUDE_DIRS include
  LIBRARIES ${PROJECT_NAME}
  CATKIN_DEPENDS roscpp)
```

Note that any packages listed as CATKIN_DEPENDS dependencies must also be declared as a <run_depend> in package.xml.

Messages, Services

These go after find_package(), but before catkin_package().

Example:

```
find_package(catkin REQUIRED COMPONENTS message_generation
std_msgs)
add_message_files(FILES MyMessage.msg)
add_service_files(FILES MyService.msg)
generate_messages(DEPENDENCIES std_msgs)
catkin_package(CATKIN_DEPENDS message_runtime std_msgs)w
```

Build Libraries, Executables

Goes after the catkin_package() call.

```
add_library(${PROJECT_NAME} src/main)
add_executable(${PROJECT_NAME}_node src/main)
target_link_libraries(
  ${PROJECT_NAME}_node ${catkin_LIBRARIES})
```

Installation

```
install(TARGETS ${PROJECT_NAME}
  DESTINATION ${CATKIN_PACKAGE_LIB_DESTINATION})
install(TARGETS ${PROJECT_NAME}_node
  DESTINATION ${CATKIN_PACKAGE_BIN_DESTINATION})
install(PROGRAMS scripts/myscript
  DESTINATION ${CATKIN_PACKAGE_BIN_DESTINATION})
install(DIRECTORY launch
  DESTINATION ${CATKIN_PACKAGE_SHARE_DESTINATION})
```

RUNNING SYSTEM

Run ROS using plain:
roscore

Alternatively, roslaunch will run its own roscore automatically if it can't find one:

```
roslaunch my_package package_launchfile.launch
```

Suppress this behaviour with the --wait flag.

Nodes, Topics, Messages

```
roscd list
rostopic list
rostopic echo cmd_vel
rostopic hz cmd_vel
rostopic info cmd_vel
rosmg show geometry_msgs/Twist
```

Remote Connection

Master's ROS environment:

- ROS_IP or ROS_HOSTNAME set to this machine's network address.
- ROS_MASTER_URI set to URI containing that IP or hostname.

Your environment:

- ROS_IP or ROS_HOSTNAME set to your machine's network address.
- ROS_MASTER_URI set to the URI from the master.

To debug, check ping from each side to the other, run roswhf on each side.

ROS Console

Adjust using rqt_logger_level and monitor via rqt_console. To enable debug output across sessions, edit the \$HOME/.ros/config/rosconsole.config

and add a line for your package:

```
log4j.logger.org.ros.package_name=DEBUG
```

And then add the following to your session:

```
export ROSCONSOLE_CONFIG_FILE=$HOME/.ros/config/rosconsole.config
```

Use the roslaunch --screen flag to force all node output to the screen, as if each declared <node> had the output="screen" attribute.



www.clearpathrobotics.com/ros-cheat-sheet
© 2015 Clearpath Robotics, Inc. All Rights Reserved.

Use of command line tools

Let's visit the below website to learn more

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

Example

Run ROS Master



Run Turtlesim

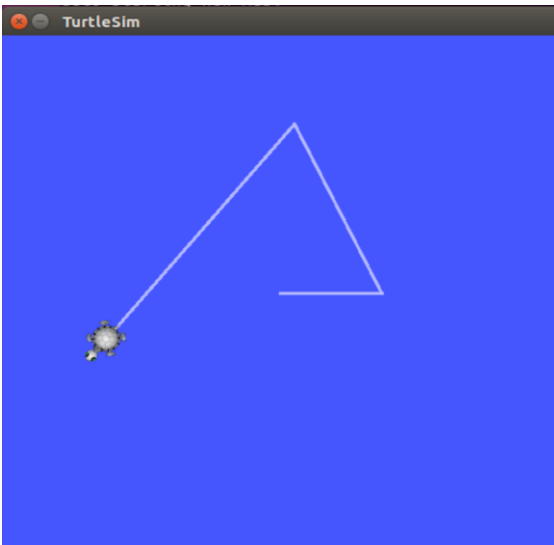


Run Turtlesim's control node

```
$ roscore
```

```
$ rosrun turtlesim turtlesim_node
```

```
$ rosrun turtlesim turtle_teleop_key
```



```
eleg@eleg-VirtualBox:~$ rosrun turtlesim turtlesim_node
[ INFO] [1631690510.668787374]: Starting turtlesim with node name /turtlesim
[ INFO] [1631690510.674501602]: Spawning turtle [turtle1] at x=[5.544445], y=[5.544445], theta=[0.000000]
XmbTextListToTextProperty result code -2
XmbTextListToTextProperty result code -2
XmbTextListToTextProperty result code -2
```

```
eleg@eleg-VirtualBox:~$ rosrun turtlesim turtle_teleop_key
Reading from keyboard
-----
Use arrow keys to move the turtle.
█
```

Example

View the list of topics

```
$ rosnodet list
```

Pub a topic message

```
$ rostopic pub /turtle1/cmd_vel geometry_msgs/Twist "liner:  
x:1.0  
y:0.0  
z:0.0  
angular:  
x:0.0  
y:0.0  
z:0.0"
```

Use tab button

Call a service request

```
$ rosservice call /spawn "x 5.0  
y: 5.0  
theater: 0.0  
name: 'turtle2'"
```


Workspace and package

Workspace

Workspace is a folder where engineering development related files are stored

- **src**: source space for source code
- **build**: build space for the files generated by the intermediate compile process
- **devel**: development space
- **install**: install space

```
workspace_folder/
src/
  CMakeLists.txt
  package_1/
    CMakeLists.txt
    package.xml
  ...
  package_n/
    CMakeLists.txt
    package.xml
  ...
build/
  CATKIN_IGNORE
devel/
  bin/
  etc/
  include/
  lib/
  share/
  .catkin
  env.bash
  setup.bash
  setup.sh
  ...
install/
  bin/
  etc/
  include/
  lib/
  share/
  .catkin
  env.bash
  setup.bash
  setup.sh
  ...
```

-- WORKSPACE
-- SOURCE SPACE
-- The 'toplevel' CMake file

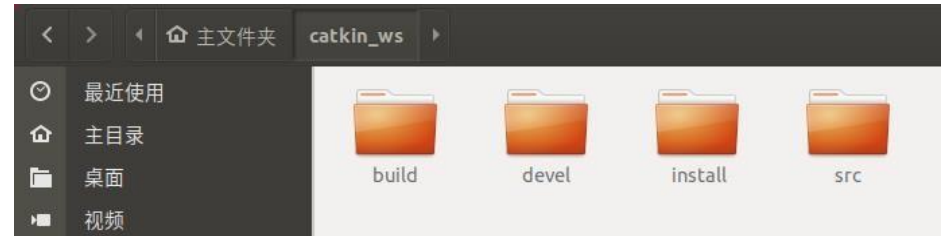
-- BUILD SPACE
-- Keeps catkin from walking this directory
-- DEVELOPMENT SPACE (set by CATKIN_DEVEL_PREFIX)

-- INSTALL SPACE (set by CMAKE_INSTALL_PREFIX)

Workspace

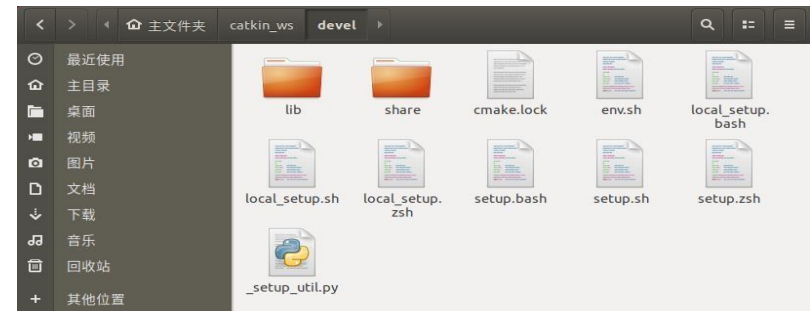
Create a ROS Workspace

```
$ mkdir -p ~/catkin_ws/src  
$ cd ~/catkin_ws/src  
$ catkin_init_workspace
```



Compile Workspace

```
$ cd ~/catkin_ws/  
$ catkin_make
```

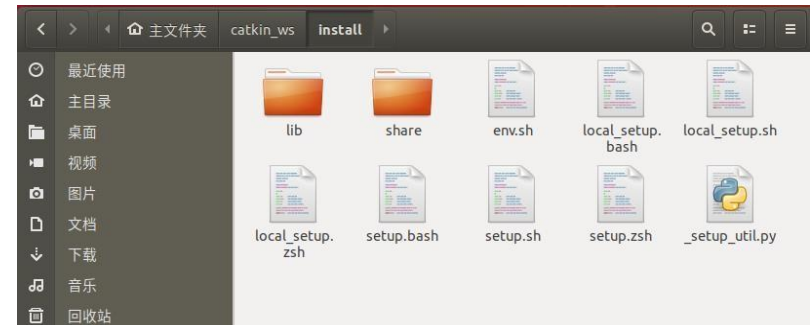


Environment setup

```
$ source devel/setup.bash
```

Check environment

```
$ echo $ROS_PACKAGE_PATH  
/home/youruser/catkin_ws/src:/opt/ros/kinetic/share
```



Package

The simplest possible package might have a structure which looks like this:

```
my_package/  
  CMakeLists.txt  
  package.xml
```

Creating a catkin Package

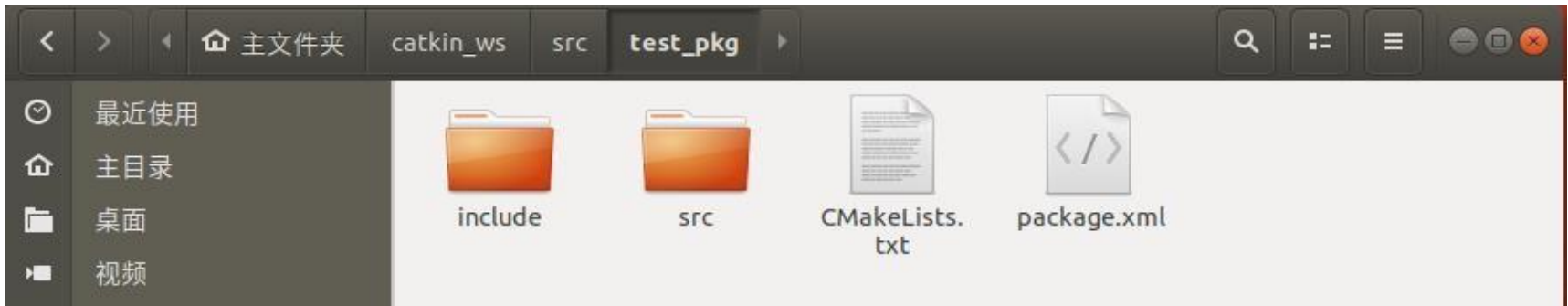
```
# catkin_create_pkg <package_name> [depend1] [depend2] [depend3]
```

```
$ cd ~/catkin_ws/src  
$ catkin_create_pkg beginner_tutorials std_msgs roscpp
```

Building a catkin workspace and sourcing the setup file

```
$ cd ~/catkin_ws  
$ catkin_make  
$ source ~/catkin_ws/devel/setup.bash
```

Package



```
打开(O)  CMakeLists.txt  保存(S)
~/catkin_ws/src/test_pkg

cmake_minimum_required(VERSION 2.8.3)
project(test_pkg)

## Compile as C++11, supported in ROS Kinetic and newer
# add_compile_options(-std=c++11)

## Find catkin macros and libraries
## if COMPONENTS list like find_package(catkin REQUIRED COMPONENTS xyz)
## is used, also find other catkin packages
find_package(catkin REQUIRED COMPONENTS
  roscpp
  rospy
  std_msgs
)

## System dependencies are found with CMake's conventions
# find_package(Boost REQUIRED COMPONENTS system)

## Uncomment this if the package has a setup.py. This macro ensures
## modules and global scripts declared therein get installed
## See http://ros.org/doc/api/catkin/html/user\_guide/setup\_dot\_py.html
# catkin_python_setup()

#####
## Declare ROS messages, services and actions ##
#####
```

```
打开(O)  package.xml  保存(S)
~/catkin_ws/src/test_pkg

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

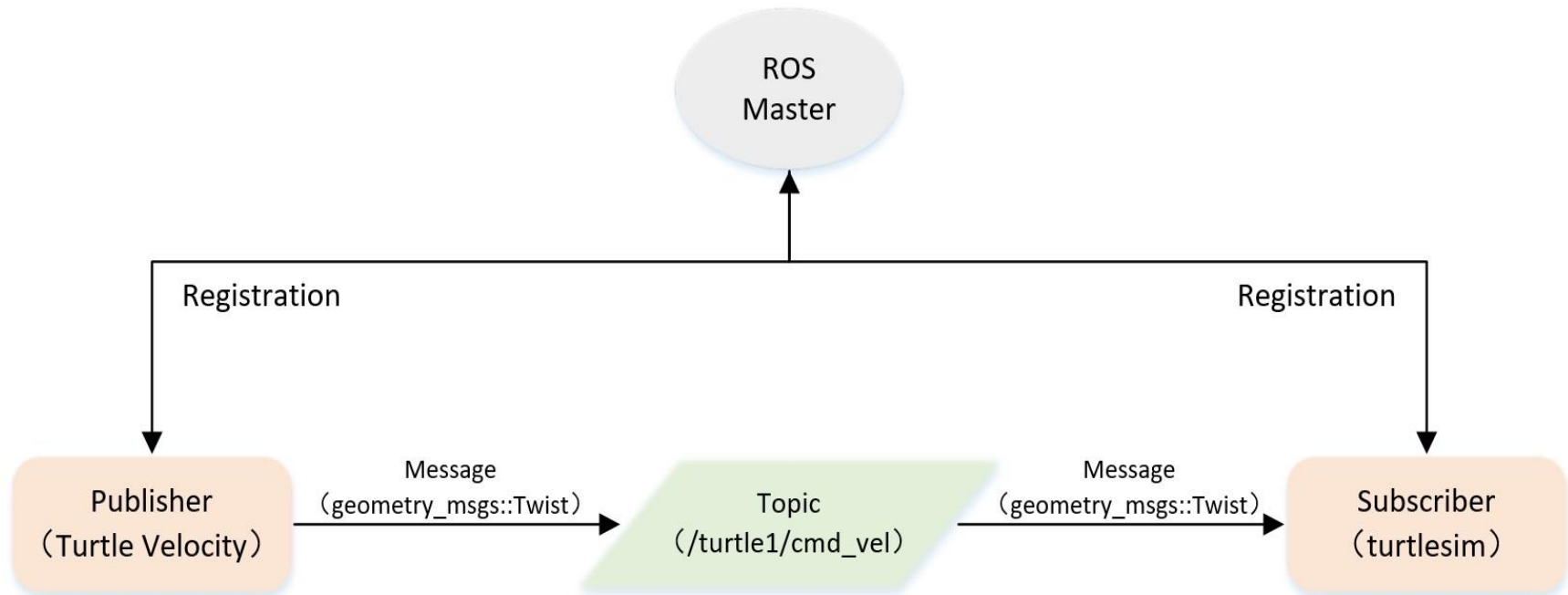
  <!-- One maintainer tag required, multiple allowed, one person per tag -->
  <!-- Example: -->
  <!-- <maintainer email="jane.doe@example.com">Jane Doe</maintainer> -->
  <maintainer email="hcx@todo.todo">hcx</maintainer>

  <!-- 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/test\_pkg</url> -->
```

Programming practice

Programming practice

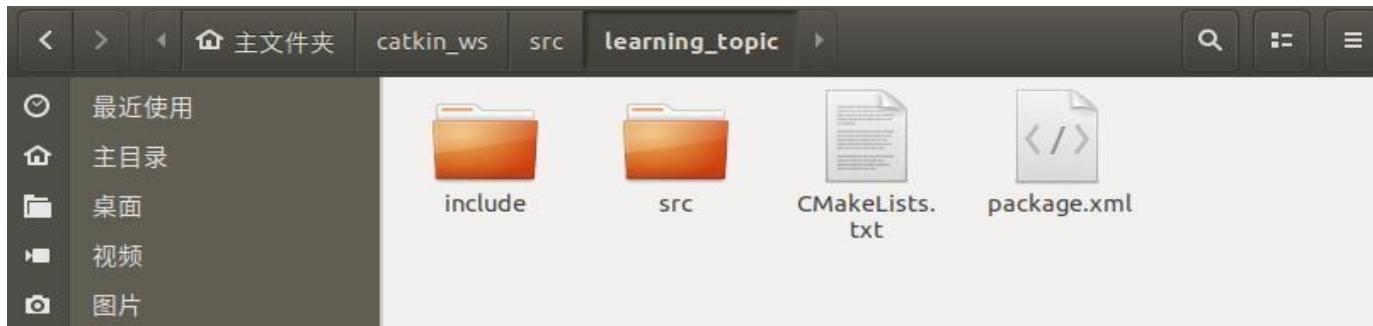


Topic Model(publish/subscribe)

Programming practice

```
$ cd ~/catkin_ws/src
```

```
$ catkin_create_pkg eleg_T03topic_yoursid std_msgs roscpp geometry_msgs turtlesim
```



Programming practice

```
1 #!/usr/bin/env python
2 # -*- coding: utf-8 -*-
3 #this example will pub turtle1/cmd_vel topic, message type is geometry_msgs::Twist, node name is velocity_publisher_last4 number of you SID,
4 #example: sid 1155135432 and the node name will be velocity_publisher_5432
5
6 # TODO 0: modify the package.xml file, add rospy dependence; set environment in ~/.bashrc file so that you do not need to source every time
7 # finish this task and show the result in the terminal.
8 import rospy
9 from geometry_msgs.msg import Twist
10
11 def velocity_publisher():
12     # TODO 0, ROS node initialize
13     rospy.init_node('node name', anonymous=True)
14
15     # create a Publisher, queue size is 10
16     # TODO 1: finish the code below
17     # reference: rospy.Publisher(topic_name, msg_class, queue_size)
18     turtle_vel_pub = rospy.Publisher(topic_name, msg_class, queue_size=10)
19
20     #set loop rate
21     rate = rospy.Rate(10)
22
23     while not rospy.is_shutdown():
24         # init geometry_msgs::Twist
25         vel_msg = Twist()
26         # TODO 2: draw a circle, the linear velocity is  $\pi$  m/s, and radius is 1 m
27         # modify the code below and import something at the start of the file, you should use the  $\pi$  in math library rather than 3.14
28         vel_msg.linear.x = 0
29         vel_msg.linear.y = 0
30         vel_msg.linear.z = 0
31         vel_msg.angular.x = 0
32         vel_msg.angular.y = 0
33         vel_msg.angular.z = 0
34
35         # publish
36         turtle_vel_pub.publish(vel_msg)
37
38         #TODO 3: modify the code below, let the terminal output velocity
39         rospy.loginfo("Publish turtle velocity command[% velocity m/s, %velocity radius]",
40             | vel_msg.linear.x, vel_msg.angular.z)
41
42         # delay as loop rate
43         rate.sleep()
44
45 if __name__ == '__main__':
46     try:
47         velocity_publisher()
48     except rospy.ROSInterruptException:
49         pass
50
```

How to implement a publisher

- Initialize the ROS node
- Register node information with ROS Master, including the name of the published topic and the type of messages in the topic
- Creating Message Data
- Post messages on a recurring basis with a certain frequency

Programming practice

```
$ cd ~/catkin_ws  
$ catkin_make  
$ source devel/setup.bash  
$ roscore  
$ rosrn turtlesim turtlesim_node  
# rosrn your_package your_ROS_node
```

```
eleg@eleg-VirtualBox: ~/catkin_ws  
[INFO] [1631848355.965107]: Publish turtle velocity command[1.000000 m/s, 1.000000 radius  
[INFO] [1631848356.062636]: Publish turtle velocity command[1.000000 m/s, 1.000000 radius  
[INFO] [1631848356.165153]: Publish turtle velocity command[1.000000 m/s, 1.000000 radius  
[INFO] [1631848356.262655]: Publish turtle velocity command[1.000000 m/s, 1.000000 radius  
[INFO] [1631848356.359284]: Publish turtle velocity command[1.000000 m/s, 1.000000 radius  
[INFO] [1631848356.459053]: Publish turtle velocity command[1.000000 m/s, 1.000000 radius  
[INFO] [1631848356.559593]: Publish turtle velocity command[1.000000 m/s, 1.000000 radius  
[INFO] [1631848356.659128]: Publish turtle velocity command[1.000000 m/s, 1.000000 radius  
[INFO] [1631848356.759708]: Publish turtle velocity command[1.000000 m/s, 1.000000 radius  
[INFO] [1631848356.859972]: Publish turtle velocity command[1.000000 m/s, 1.000000 radius  
[INFO] [1631848356.961572]: Publish turtle velocity command[1.000000 m/s, 1.000000 radius
```

