



ELEG4701

Intelligent Interactive Robot Practice

Lab 3: ROS – Installation & Basic Knowledge

Jiewen Lai

Research Assistant Professor

EE, CUHK

jiewen.lai@cuhk.edu.hk



Today's Agenda

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



What is ROS



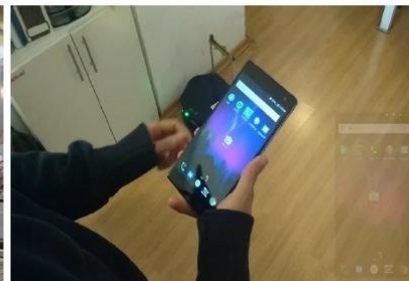
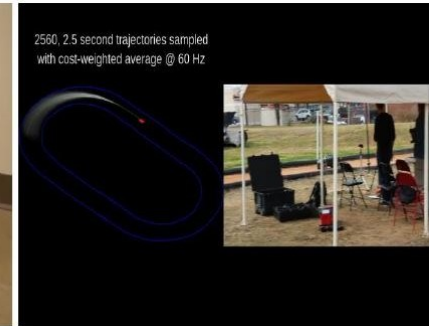
History of ROS

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)

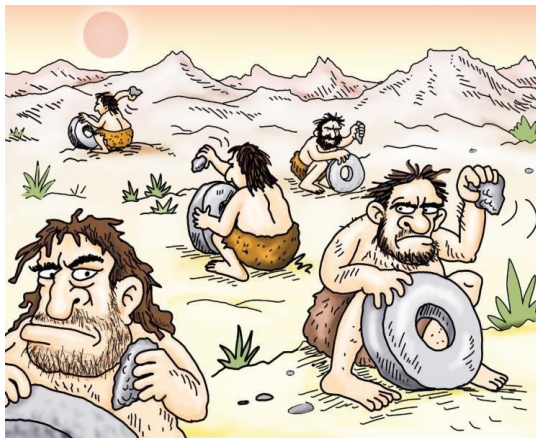
ROS Hydro Medusa	September 4th, 2013			May, 2015
ROS Groovy Galapagos	December 31, 2012			July, 2014
ROS Fuerte Turtle	April 23, 2012			--
ROS Electric Emys	August 30, 2011			--
ROS Diamondback	March 2, 2011			--
ROS C Turtle	August 2, 2010			--
ROS Box Turtle	March 2, 2010			--

Box Turtle

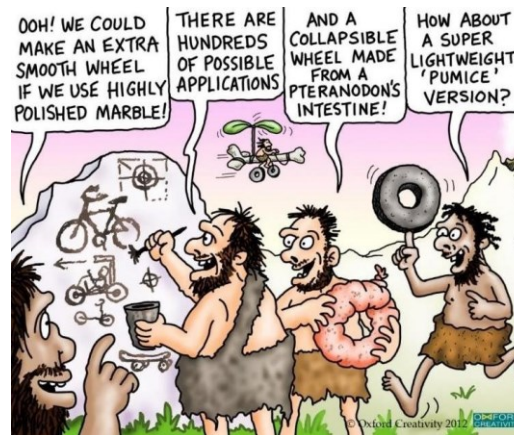
Application of ROS



What is ROS



Legacy Mode



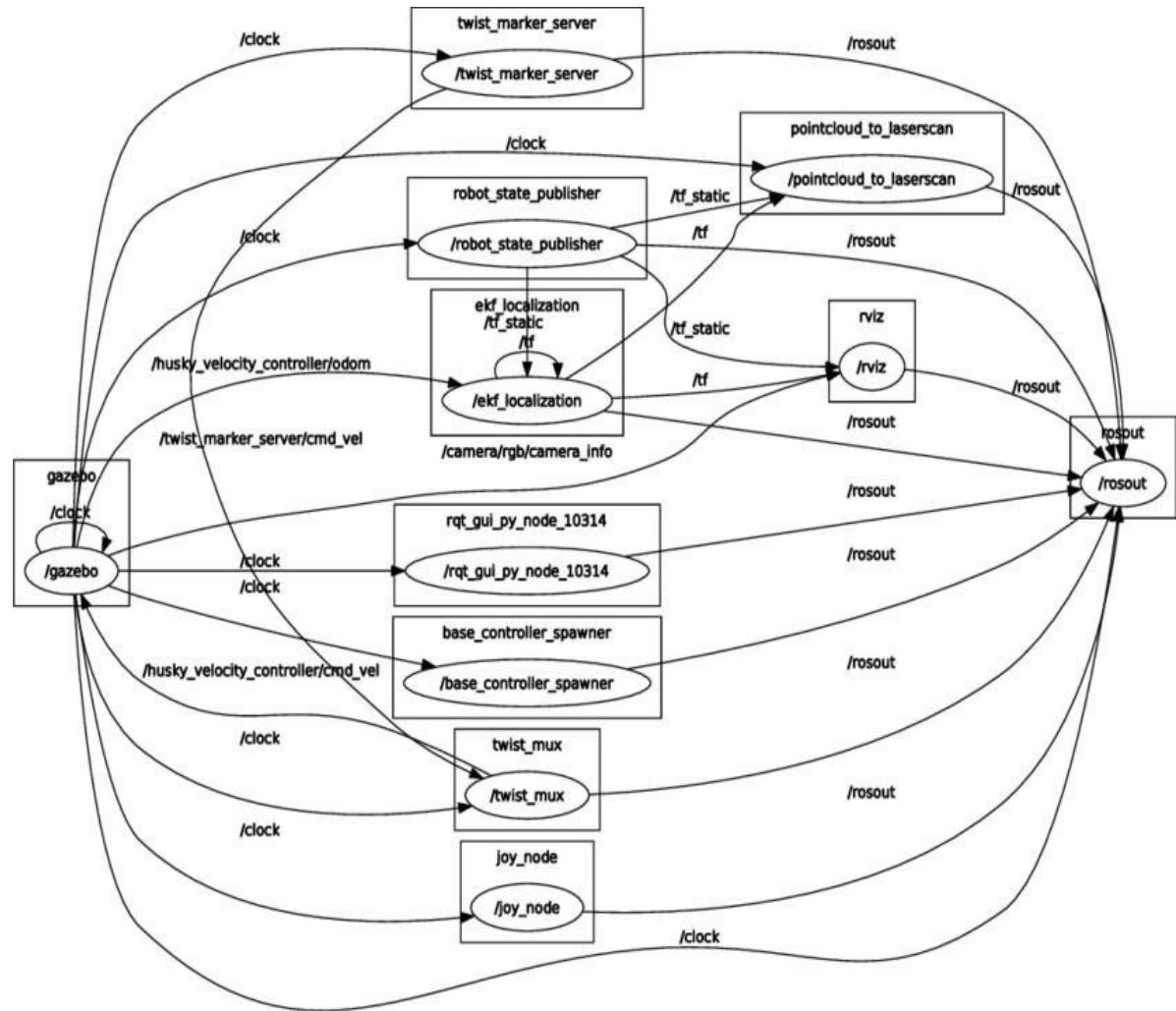
Modern Mode

Improve the **software reuse rate** in robotics research & 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

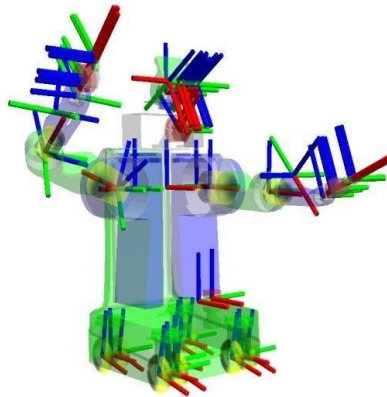
include/package_name	C++ header files
src	Source files. Python libraries in subdirectories
scripts	Python nodes and scripts
msg, srv, action	Message, Service, and Action definitions

Release Repo Packages

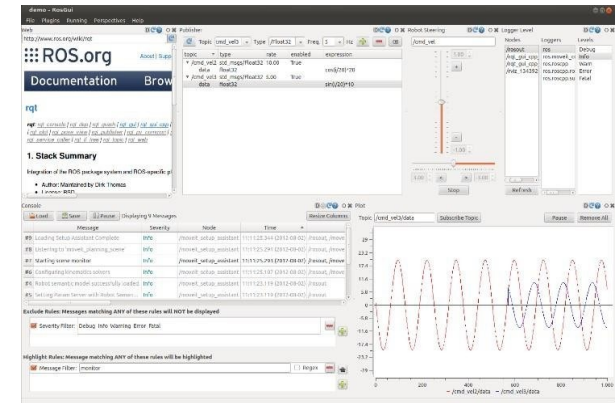
```
catkin_generate_changelog
# review & commit changelogs
catkin_prepare_release
bloom-release --track kinetic --ros-distro kinetic repo_name
```

Reminders

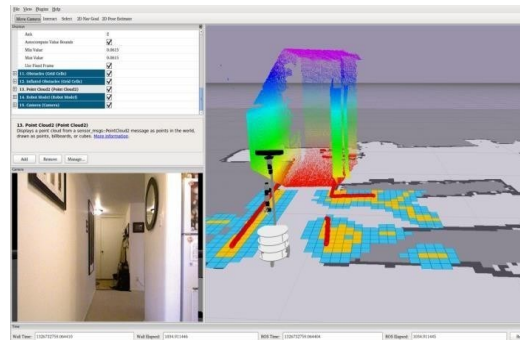
- Testable logic
- Publish diagnostics
- Desktop dependencies in a separate package



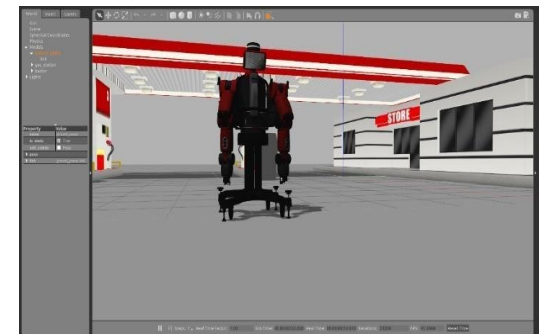
TF coordinate transformation



QT Toolbox



Rviz

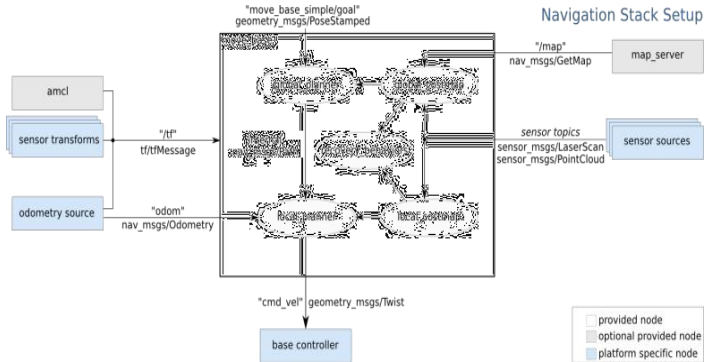


Gazebo

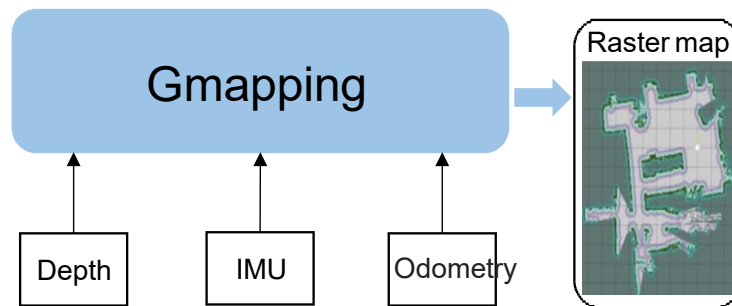
Command line and compiler



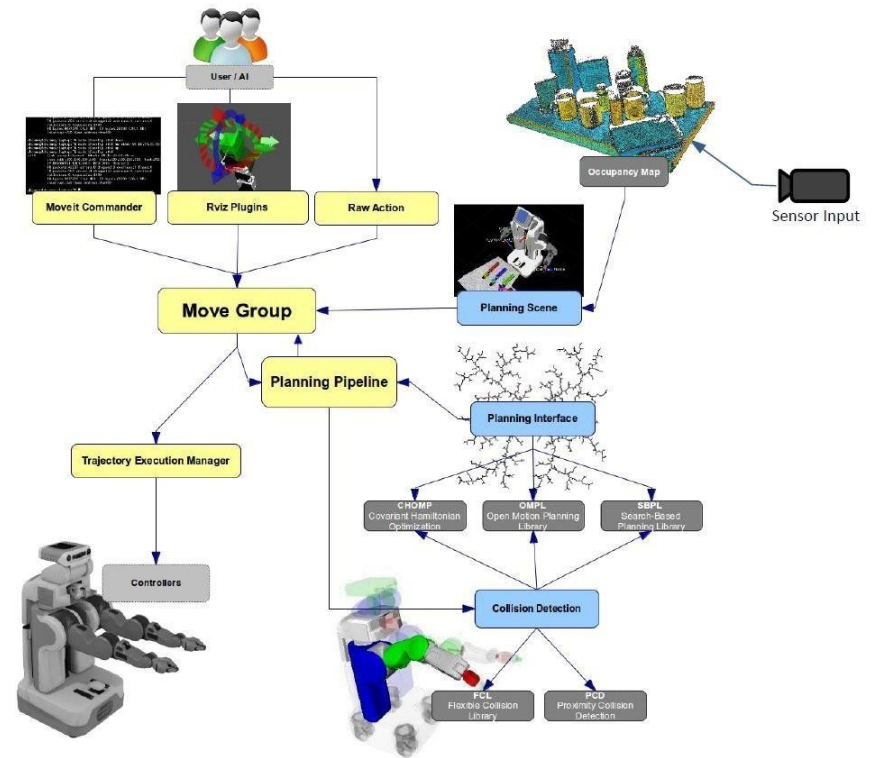
Applications



Navigation



SLAM



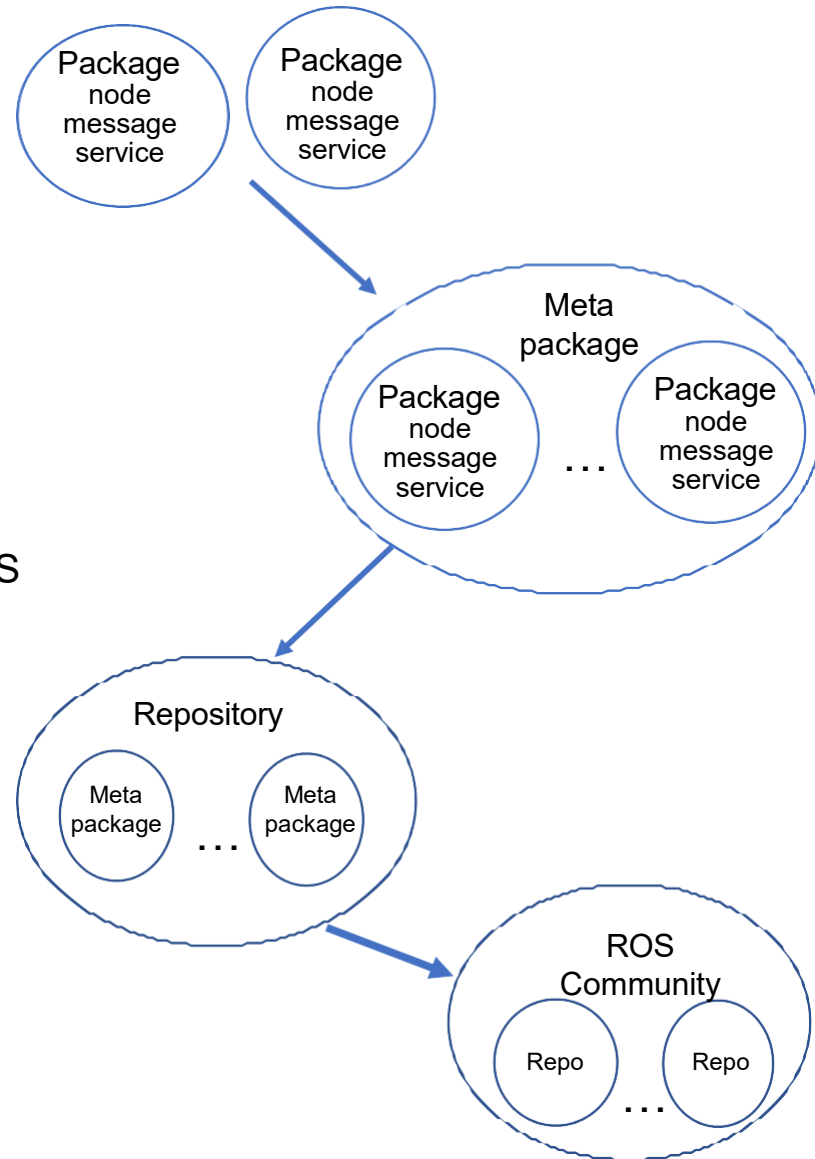
MoveIt



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 open source 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

ROS.org

[About](#) | [Support](#) | [Discussion Forum](#) | [Service Status](#) | [Q&A answers.ros.org](#)

Search:

Documentation

Browse Software

News

Download

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



ROS 2 Documentation

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

ROS.org

[About](#) | [Support](#) | [Discussion Forum](#) | [Index](#) | [Service Status](#) | [Q&A answers.ros.org](#)

Documentation

Browse Software

News

Download

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.

noetic

ROS Noetic Ninjemys

ROS Noetic Ninjemys is the thirteenth ROS distribution release. It was released on May 23rd, 2020.

Contents

1. ROS Noetic Ninjemys
1. Platforms
2. Installation
3. Release Planning
4. Changes

1. Platforms

ROS Noetic Ninjemys is primarily targeted at the Ubuntu 20.04 (Focal) release, though other systems are supported to varying degrees. For more information on compatibility on other platforms, please see [REP 3: Target Platforms](#).

2. Installation

See [ROS Noetic Ninjemys installation instructions](#).

3. Release Planning

See [Planning](#)

4. Changes

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



ROS 2 Documentation

The ROS Wiki is for ROS 1. Are you using ROS 2 (Humble, Iron, or Rolling)?

[Check out the ROS 2 Project Documentation](#)

Package specific documentation can be found on [index.ros.org](#)

Wiki

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

Page

[Immutable Page](#)
[Info](#)
[Attachments](#)

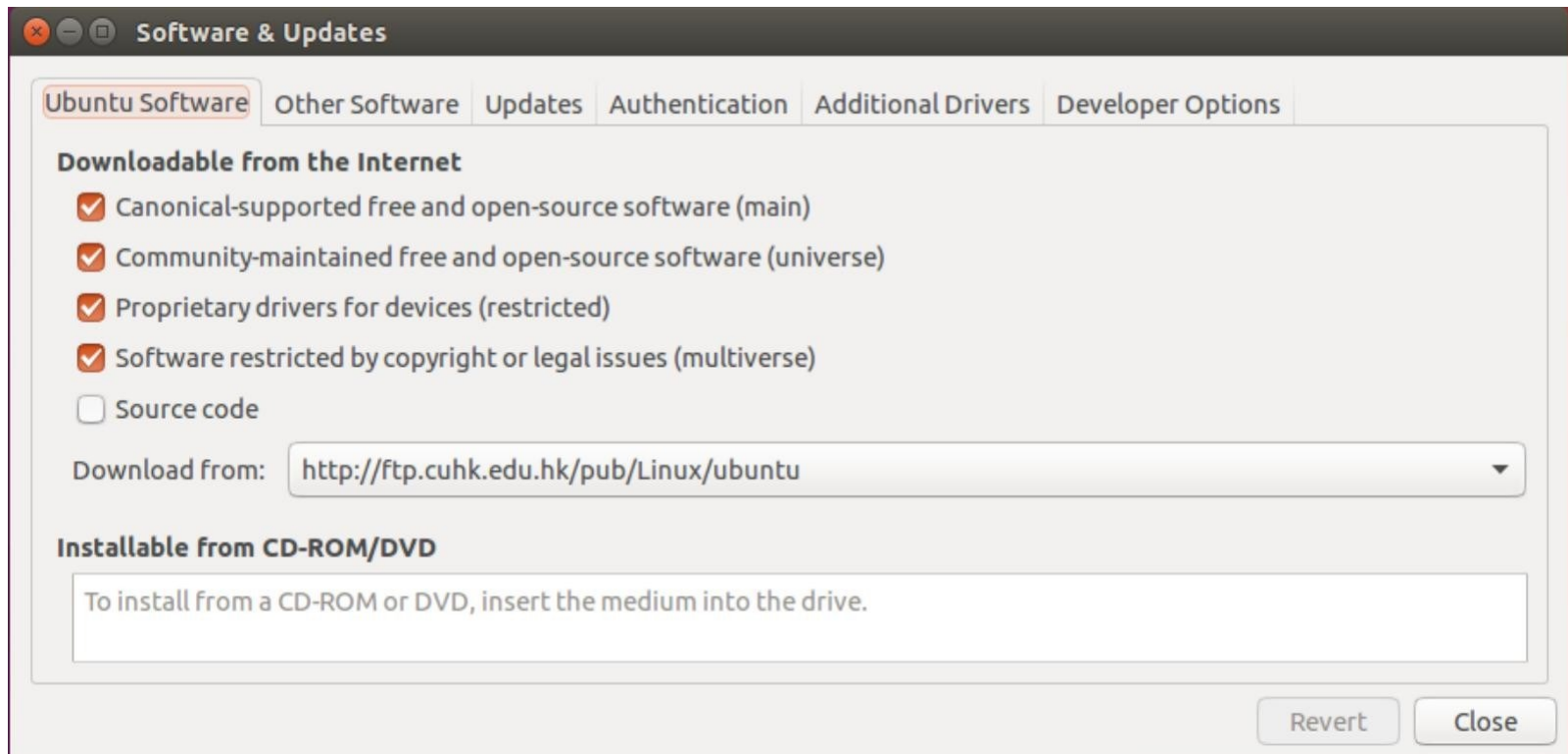
More Actions:

User

[Login](#)



Configure your Ubuntu repositories





Install ROS on Ubuntu

1. Setup your sources.list

- `$ sudo sh -c 'echo "deb http://packages.ros.org/ros/ubuntu $(lsb_release -sc) main" >/etc/apt/sources.list.d/ros-latest.list'`

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-noetic-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

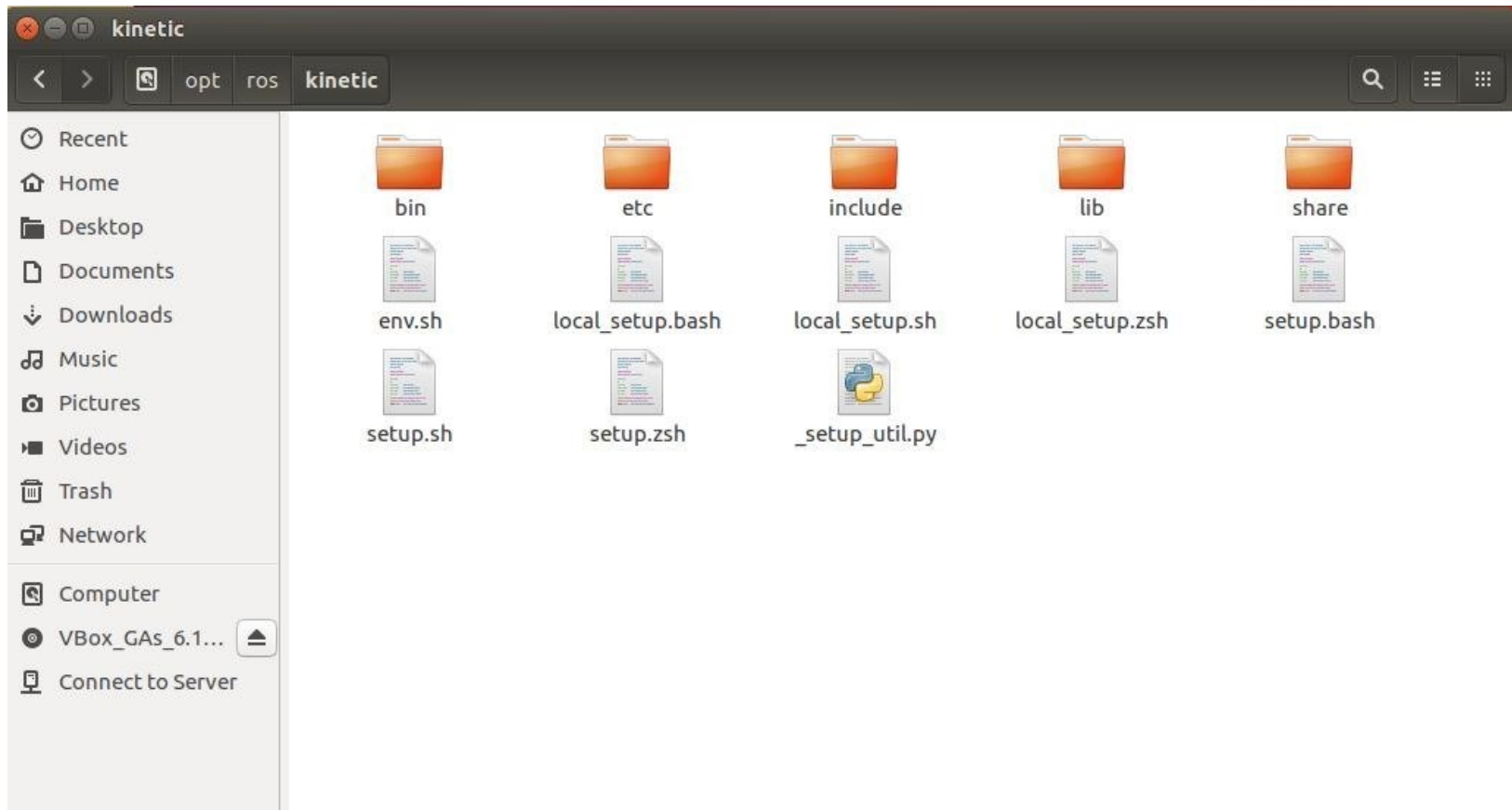
- `$ sudo rosdep init`
- `$ rosdep update`

ROS Wiki

<http://wiki.ros.org/Installation/Ubuntu>



Install ROS on Ubuntu





Install ROS on Ubuntu

Run ROS Master



Run Turtlesim

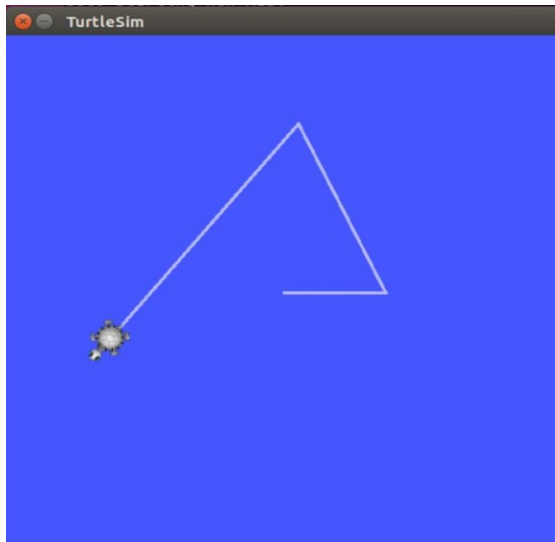


Run Turtlesim's control node

```
$ roscore
```

```
$ rosrunc turtlesim turtlesim_node
```

```
$ rosrunc turtlesim turtle_teleop_key
```



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



Core concepts in ROS



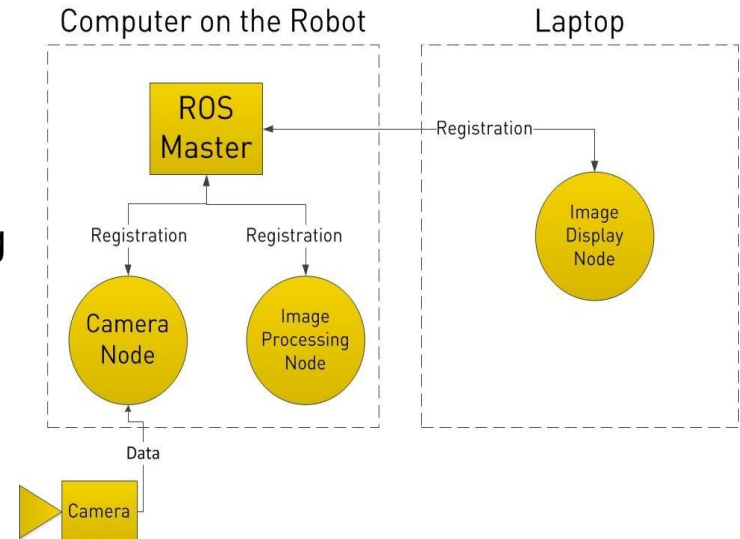
Core concepts in ROS

■ 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





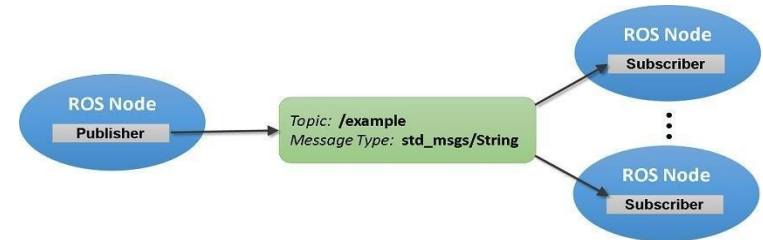
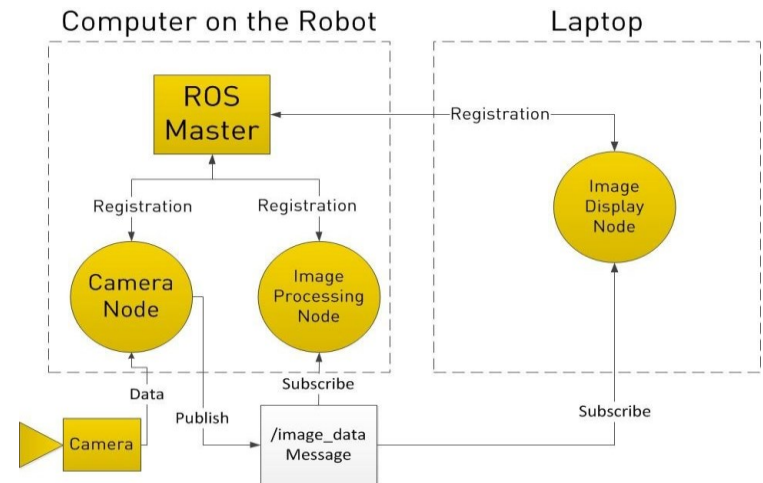
Core concepts in ROS

■ 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

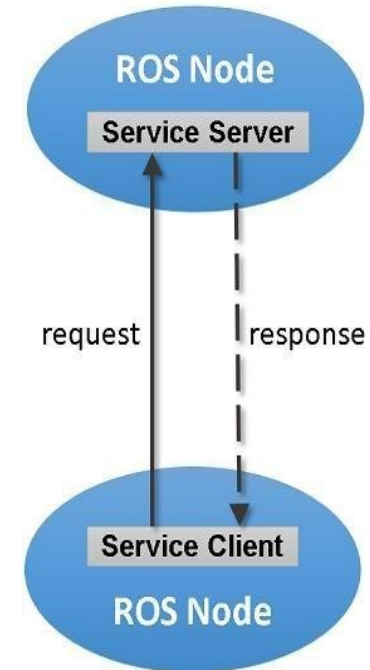
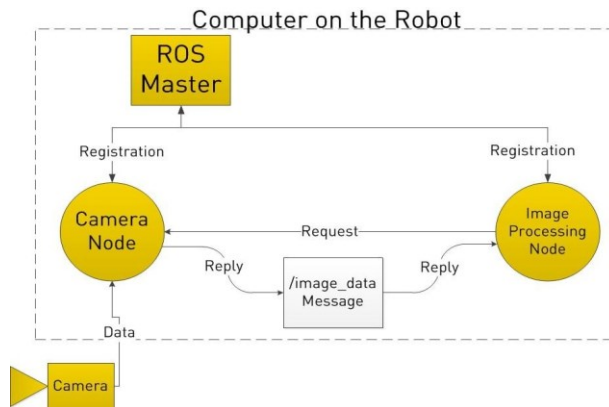


Topic Model (publish/subscribe)

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	Multiple-to-Multiple	One to Multiple
Applicable scenarios	Data Transfer	Logic Processing



File System

■ Package

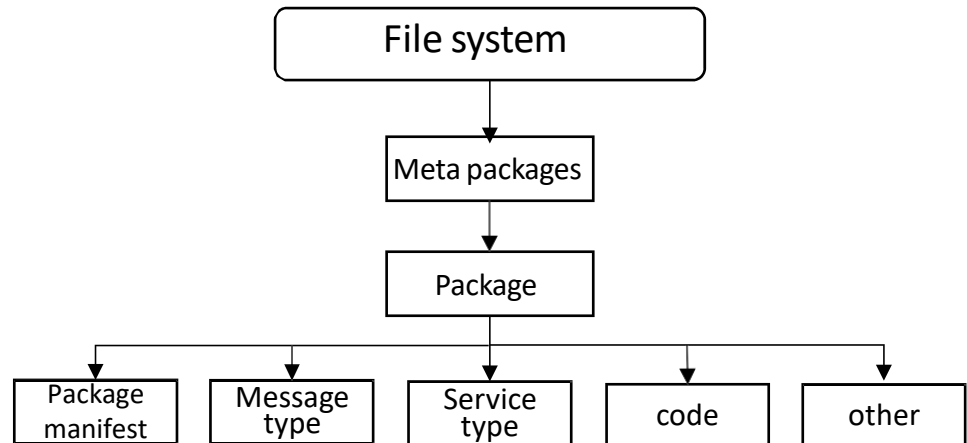
- The basic unit in ROS software, containing node source code, configuration files, data definitions

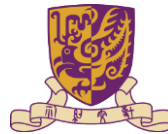
■ Package manifest

- 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

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; cd ../src
rostopic list
rostopic list
rostopic echo cmd_vel
rostopic hz cmd_vel
rostopic info cmd_vel
rostopic 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 rosnet 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.ros.package_name=DEBUG

And then add the following to your session:

```
export ROS_CONSOLE_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.

Common commands:

- rostopic
- roscd
- rosservice
- rosparm
- rosmg
- rossrv



Use of command line tools

Please visit the website below to learn more

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



Example

Run ROS Master



Run Turtlesim

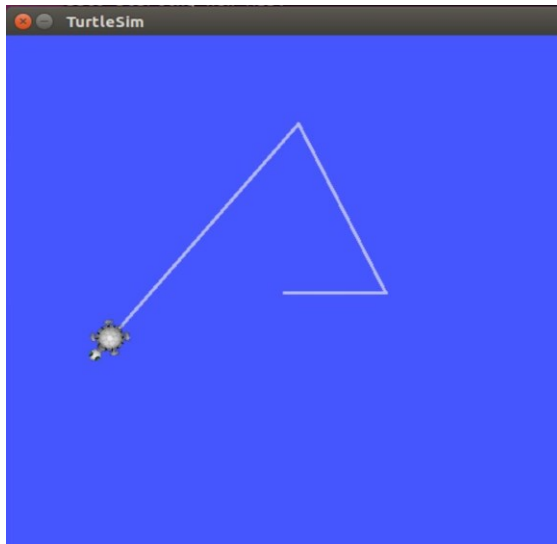


Run Turtlesim's control node

```
$ roscore
```

```
$ rosrunc turtlesim turtlesim_node
```

```
$ rosrunc turtlesim turtle_teleop_key
```



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




Example

View the list of topics

```
$ rosnode list
```

Pub a topic message

Use tab button

```
$ 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"
```

Call a service request

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



Workspace and package

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



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  
$ echo $ROS_PACKAGE_PATH
```

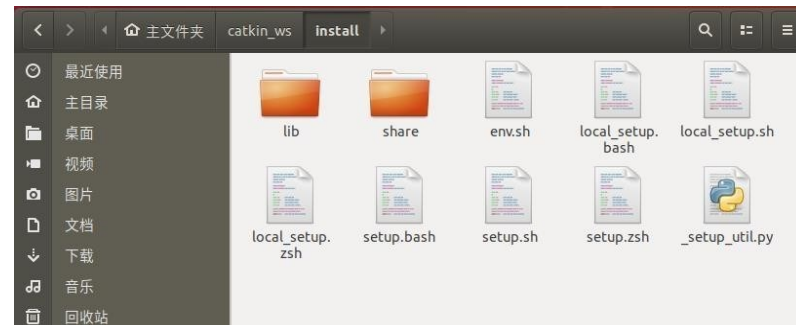
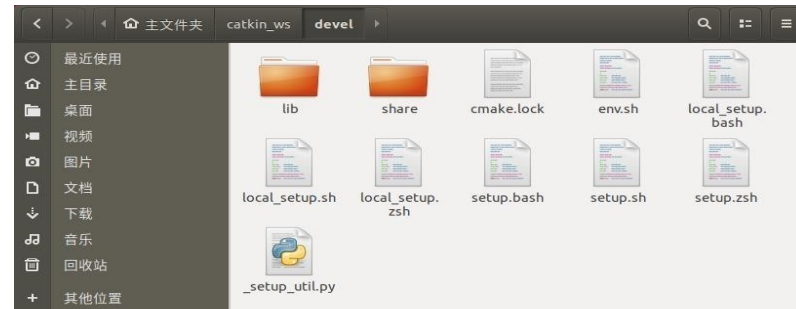
Environment setup

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

Check environment

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

```
/home/youruser/catkin_ws/src:/opt/ros/kinetic/share:/home/youruser/some_ws/src
```





Package

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

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

For a package to be considered a catkin package it must meet a few requirements:

- The package must contain a [catkin compliant package.xml](#) file.
 - That package.xml file provides meta information about the package.
- The package must contain a [CMakeLists.txt](#) which uses [catkin](#).
 - If it is a [catkin metapackage](#) it must have the relevant boilerplate CMakeLists.txt file.
- Each package must have its own folder
 - This means no nested packages nor multiple packages sharing the same directory.

Creating a catkin Package

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

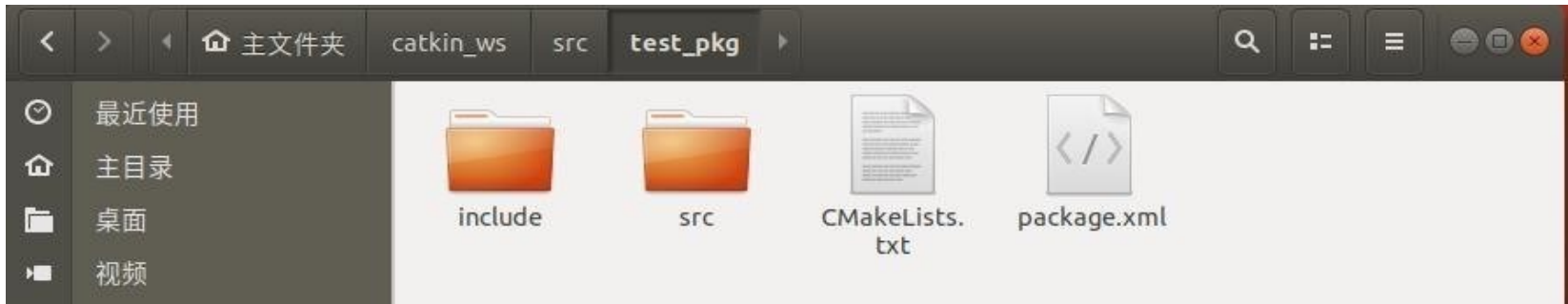
```
$ cd ~/catkin_ws/src  
$ catkin_create_pkg beginner_tutorials std_msgs rospy roscpp
```

Building a catkin workspace and sourcing the setup file

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



Package



```
CMakeLists.txt
~/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 ##
#####
```

```
package.xml
~/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>

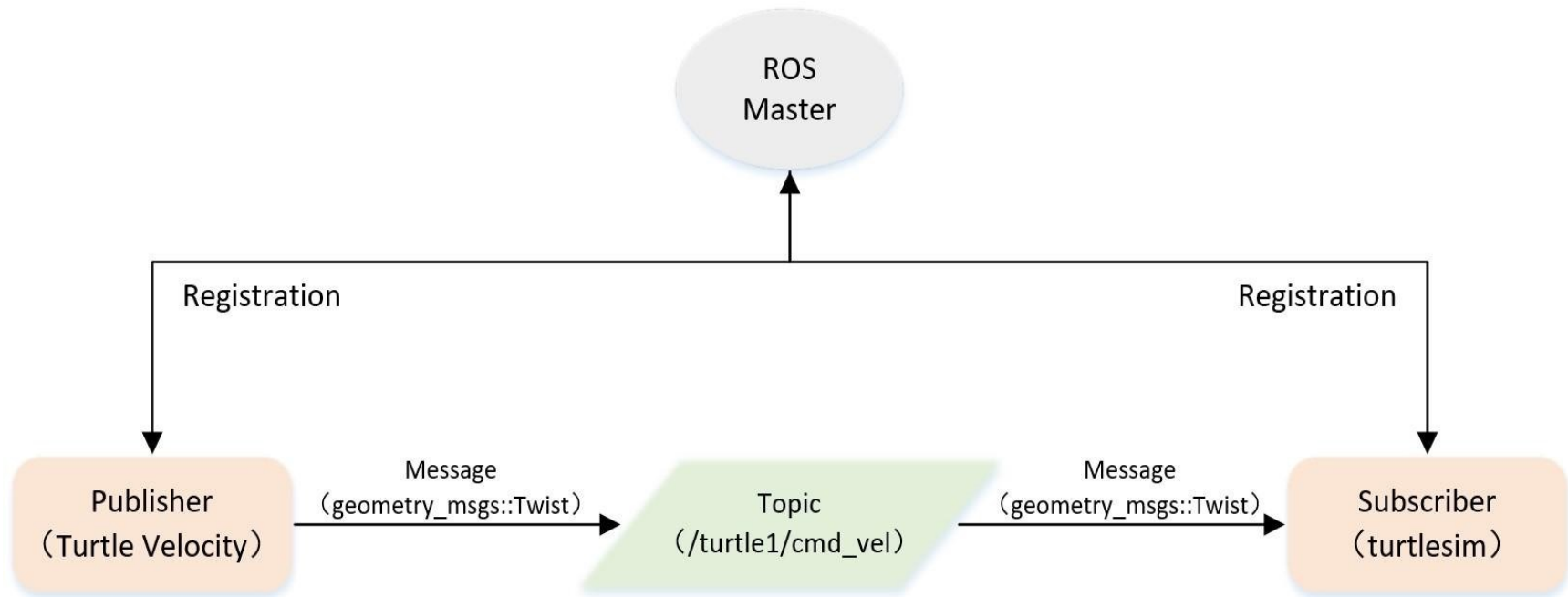
  <!-- Url tags are optional, but multiple are allowed, one per tag -->
  <!-- Optional attribute type can be: website, bugtracker, or repository -->
  <!-- Example: -->
  <!-- <url type="website">http://wiki.ros.org/test_pkg</url> -->
```




Programming practice (Task 3)



Programming practice

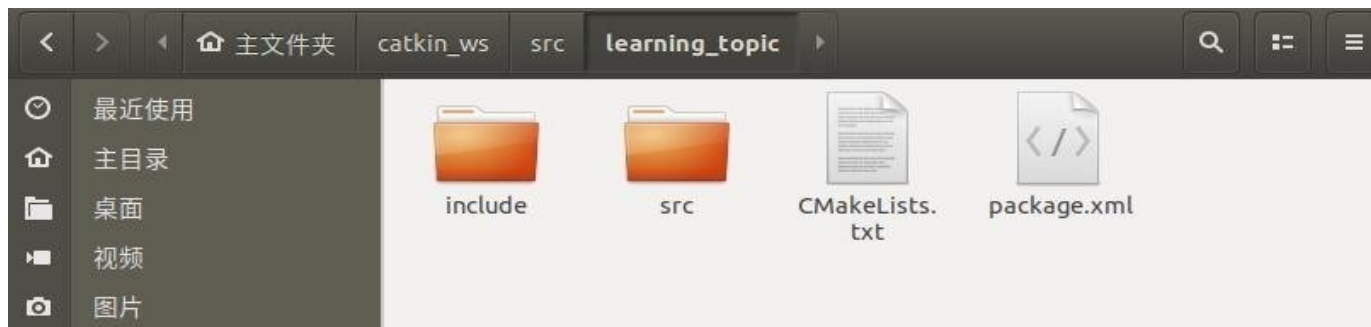


Topic Model (publish/subscribe)



Programming practice

```
$ cd ~/catkin_ws/src  
$ catkin_create_pkg eleg_t03topic_yoursid std_msgs roscpp geometry_msgs turtlesim
```





Programming practice

How to implement a publisher

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

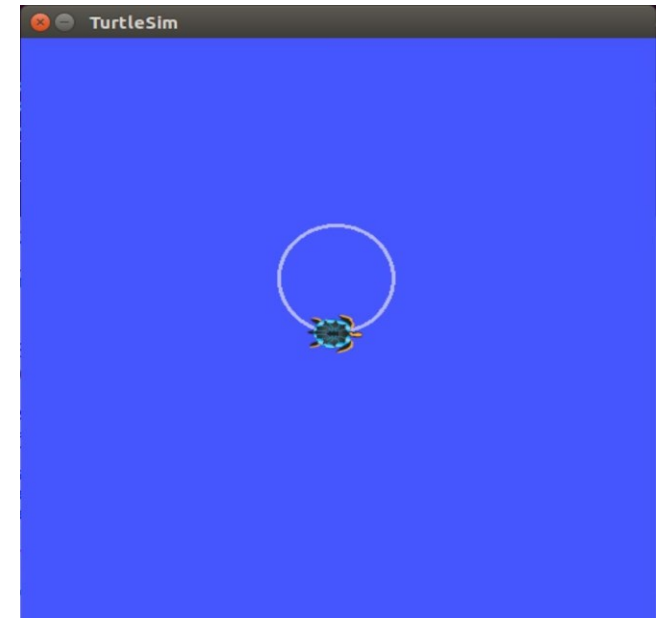
```
1 #!/usr/bin/env python
2 # -*- coding: utf-8 -*-
3 #this example will pub turtle1/cmd_vel topic, message type is geometry_msgs::Twist, node name is velocity_publisher_last 3 number of you SID,
4 #example: sid 1155135432 and the node name will be velocity_publisher_432
5
6 # Todo 0: modify the package.xml file, add rospys 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 1, ROS node initialize
13     rospy.init_node('node name', anonymous=True)
14     # create a Publisher, queue size is 10
15     # Todo 2: finish the code below
16     # reference: rospy.Publisher(topic_name, msg_class, queue_size)
17     turtle_vel_pub = rospy.Publisher(topic_name, msg_class, queue_size=10)
18     #set loop rate
19     rate = rospy.Rate(10)
20     while not rospy.is_shutdown():
21         # init geometry_msgs::Twist
22         vel_msg = Twist()
23         # Todo 3: draw a circle, the linear velocity is  $\pi$  m/s, and radius is 1 m
24         # 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
25         vel_msg.linear.x = 0
26         vel_msg.linear.y = 0
27         vel_msg.linear.z = 0
28         vel_msg.angular.x = 0
29         vel_msg.angular.y = 0
30         vel_msg.angular.z = 0
31         # publish
32         turtle_vel_pub.publish(vel_msg)
33         #1000 3: modify the code below, let the terminal output 2 velocities, hold 3 digits after the decimal point
34         rospy.loginfo("Publish turtle velocity command[ modify here]",
35                       vel_msg.linear.x, vel_msg.angular.z)
36         # delay as loop rate
37         rate.sleep()
38 if __name__ == '__main__':
39     try:
40         velocity_publisher()
41     except rospy.ROSInterruptException:
42         pass
```



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]: Pubsh turtle velocity command[1.000000 m/s, 1.000000 radius  
[INFO] [1631848356.062636]: Pubsh turtle velocity command[1.000000 m/s, 1.000000 radius  
[INFO] [1631848356.165153]: Pubsh turtle velocity command[1.000000 m/s, 1.000000 radius  
[INFO] [1631848356.262655]: Pubsh turtle velocity command[1.000000 m/s, 1.000000 radius  
[INFO] [1631848356.359284]: Pubsh turtle velocity command[1.000000 m/s, 1.000000 radius  
[INFO] [1631848356.459053]: Pubsh turtle velocity command[1.000000 m/s, 1.000000 radius  
[INFO] [1631848356.559593]: Pubsh turtle velocity command[1.000000 m/s, 1.000000 radius  
[INFO] [1631848356.659128]: Pubsh turtle velocity command[1.000000 m/s, 1.000000 radius  
[INFO] [1631848356.759708]: Pubsh turtle velocity command[1.000000 m/s, 1.000000 radius  
[INFO] [1631848356.859972]: Pubsh turtle velocity command[1.000000 m/s, 1.000000 radius  
[INFO] [1631848356.961572]: Pubsh turtle velocity command[1.000000 m/s, 1.000000 radius
```





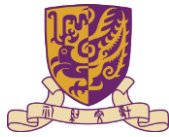
Programming practice

```
1  #!/usr/bin/env python
2  # -*- coding: utf-8 -*-
3
4  import sys
5  import rospy as ros
6  # if you use "import rospy", you must use rospy.**. example: rospy.loginfo(), there is a
7  # mistake in this code file related to this issue, please find and correct it TODO 0
8
9  from geometry_msgs.msg import Twist
10 from turtlesim.msg import Pose
11
12 ROBOT_X = 0
13
14 def pose_callback(pose):
15     global ROBOT_X
16     ros.loginfo("Robot X = %f: Y=%f: Z = %f\n", pose.x, pose.y, pose.theta)
17     ROBOT_X = pose.x
18
19 # TODO 0: modify the code below so that you can pub velocity and subscribe the pose
20 def move_turtle(modify_here, modify_here, modify_here): #modify code in this line
21     global ROBOT_X
22     ros.init_node('move_turtle', anonymous=False)
23     pub = ros.Publisher( topic_name, msg_class, queue_size=10) #modify code in this line
24     ros.Subscriber( topic_name, msg_class, pose_callback) #modify code in this line
25
26     rate = rospy.Rate(10)
27     vel = Twist()
28     while not ros.is_shutdown():
29         # TOTO 1: draw a circle, the linear velocity is  $\pi$  m/s, and radius is 1 m
30         # 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
31         # if the x positon larger than 8, stop to draw, you need to input 3 parameters in terminal
```



Programming practice

```
32
33     vel.linear.x = 0
34     vel.linear.y = 0
35     vel.linear.z = 0
36     vel.angular.x = 0
37     vel.angular.y = 0
38     vel.angular.z = 0
39     ros.loginfo("Linear Vel = %f: Angular Vel = % f", modify_here, modify_here) #modify code in this line
40     if ROBOT_X >= modify_here: #modify code in this line
41         ros.loginfo('Robot exercises finished.')
42         ros.logwarn('stopping robot')
43         break
44     pub.publish(vel)
45     rate.sleep()
46
47 if __name__ == '__main__':
48     try:
49         move_turtle(float(sys.argv[1]), float(sys.argv[2]), float(sys.argv[3]))
50     except ros.ROSInterruptException:
51         pass
```



Programming practice

```
eleg@eleg-VirtualBox: ~/catkin_ws

[INFO] [1632051795.680394]: Robot X = 7.755259: Y=6.479162: Z = 0.784000
[INFO] [1632051795.703936]: Robot X = 7.790277: Y=6.515218: Z = 0.800000
[INFO] [1632051795.709409]: Linear Vel = 3.141400: Angular Vel = 1.000000
[INFO] [1632051795.711725]: Robot X = 7.824714: Y=6.551829: Z = 0.816000
[INFO] [1632051795.729382]: Robot X = 7.858560: Y=6.588987: Z = 0.832000
[INFO] [1632051795.743914]: Robot X = 7.891808: Y=6.626682: Z = 0.848000
[INFO] [1632051795.760546]: Robot X = 7.924448: Y=6.664904: Z = 0.864000
[INFO] [1632051795.776566]: Robot X = 7.956473: Y=6.703643: Z = 0.880000
[INFO] [1632051795.792842]: Robot X = 7.987874: Y=6.742890: Z = 0.896000
[INFO] [1632051795.807698]: Robot X = 8.018643: Y=6.782634: Z = 0.912000
[INFO] [1632051795.808259]: Linear Vel = 3.141400: Angular Vel = 1.000000
[INFO] [1632051795.808435]: Robot exercises finished.
[WARN] [1632051795.808617]: stopping robot
eleg@eleg-VirtualBox:~/catkin_ws$
```





Programming practice

```
sudo apt install git  
git clone https://github.com/jiewen-lai/eleg4701-t03.git
```