

---

---

# EE405

# Robotics Operating System (ROS)

(TA) Hyunki Seong, Sungwon Nah  
School of Electrical Engineering  
KAIST

Sep 08, 2023

{hynkis, sw.nah}@kaist.ac.kr

# Contents

---

1. Ubuntu Setup (Recap)
2. ROS Background (Recap)
3. ROS Tutorials
  - 1) Installation & Configuration of ROS
  - 2) ROS Package Tutorial
  - 3) ROS Topic, Command
4. ROS Programming
5. # Assignment 1: Simple ROS topic publisher & subscriber
6. Appendix (ROS Tools, ROS Messages)

---

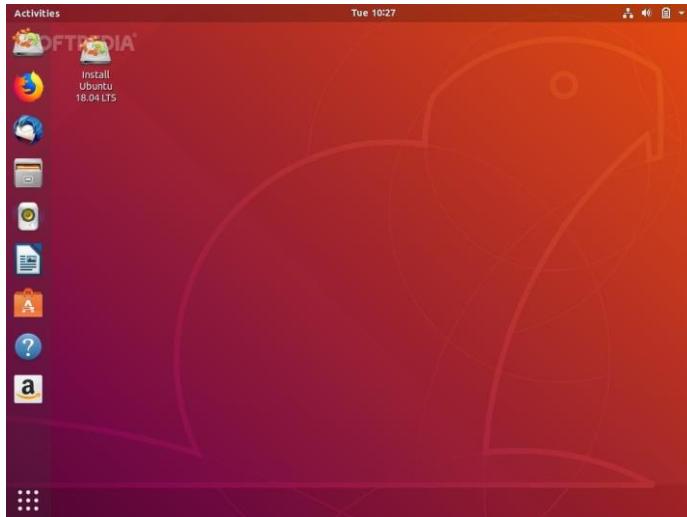
---

# Ubuntu Setup

# Ubuntu Installation

## ❖ Ubuntu 18.04

- ❑ Ubuntu is a Linux kernel-based Operating System(OS).
- ❑ It is one of the **most popular developer OS** for personal desktops and laptops.
- ❑ For ROS1 framework, **Ubuntu 18.04** release version is suitable for use.  
(Recently, ROS2 is getting utilized for industrial area.)



Several figures from the Ubuntu 18.04 desktop

# Ubuntu Installation

## ❖ Ubuntu 18.04 Installation

- ❑ There are two ways of Ubuntu Installation in your PC.

### 1. Install the Ubuntu alongside with Window **in Dual Boot**.

- Link(Eng) : <https://www.itzgeek.com/how-tos/linux/ubuntu-how-tos/how-to-install-ubuntu-18-04-alongside-with-windows-10-or-8-in-dual-boot.html>
- Link(Kor) : <https://coding-factory.tistory.com/494>

### 2. Install the Ubuntu **in a Virtual Machine**.

- Link(Eng) : <https://codebots.com/library/techies/ubuntu-18-04-virtual-machine-setup>

# Ubuntu Installation

## ❖ Ubuntu 18.04 Installation

- ❑ There are two ways of Ubuntu Installation in your PC.

### 3. Install the Ubuntu **in a Docker container**.

(Ubuntu-based)

- A Guide to Docker and ROS: <https://roboticseabass.com/2021/04/21/docker-and-ros/>
- ROS Docker GUI for Windows and Linux: <https://www.youtube.com/watch?v=qWuudNxFGOQ>

(Window-based)

- Setting up ROS in Windows through Docker: <https://jack.kawell.us/posts/ros-windows-docker/>
- ROS with Docker on Windows 10: <https://rickyvanrijn.nl/2021/01/19/ros-with-docker-on-windows-10/>
- WSL2 + DockerでROS1をセットアップ: [https://zenn.dev/takahiro\\_kawai/scrap/695cb148a2e043](https://zenn.dev/takahiro_kawai/scrap/695cb148a2e043) (this use WSL)



# Ubuntu Installation

## ❖ Ubuntu 18.04 Installation

- ❑ Docker-based Ubuntu 18.04 setup.

Following: <https://roboticseabass.com/2021/04/21/docker-and-ros/>

1. Install docker in your Ubuntu (e.g., Ubuntu 20.04).

- If dependency error occurs: <https://askubuntu.com/questions/1409192/cannot-install-docker-desktop-on-ubuntu-22-04>
- If permission error occurs: <https://technote.kr/369>  
sudo usermod -a -G docker \$USER

2. Run Ubuntu 18.04 by using Docker

1. Share X server for GUI

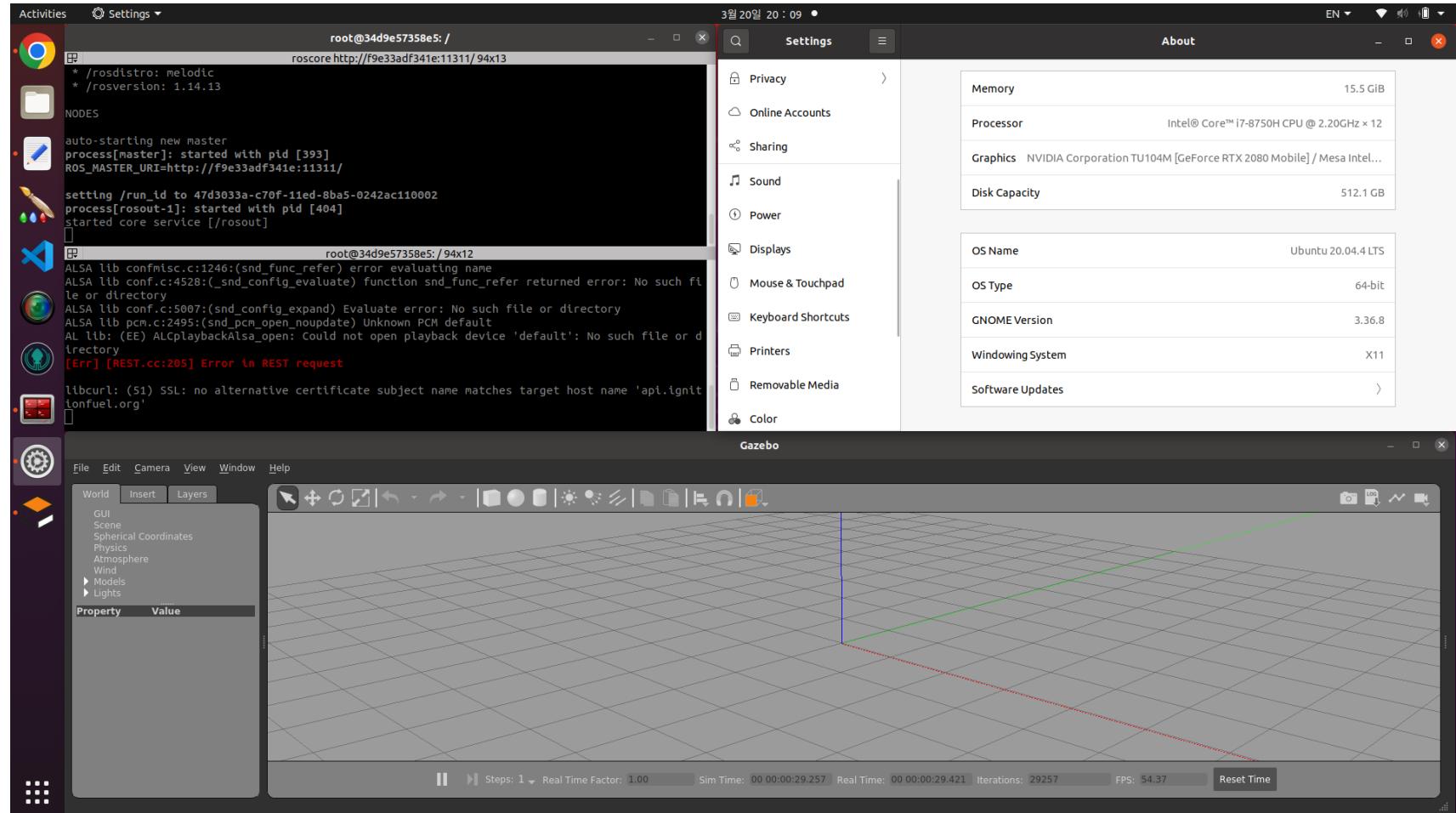
```
xhost +
```

2. Pull & Run docker image (ROS melodic, Ubuntu 18.04).

```
docker run --rm -v /tmp/.X11-unix:/tmp/.X11-unix -e DISPLAY=$DISPLAY \
-e TZ=$(cat /etc/timezone) osrf/ros:melodic-desktop-full bash
```

# Ubuntu Installation

## ❖ Ubuntu 18.04 Installation



# Ubuntu Installation

## ❖ Ubuntu 18.04 Installation

### 3. (Optional) Run a customized ROS docker container

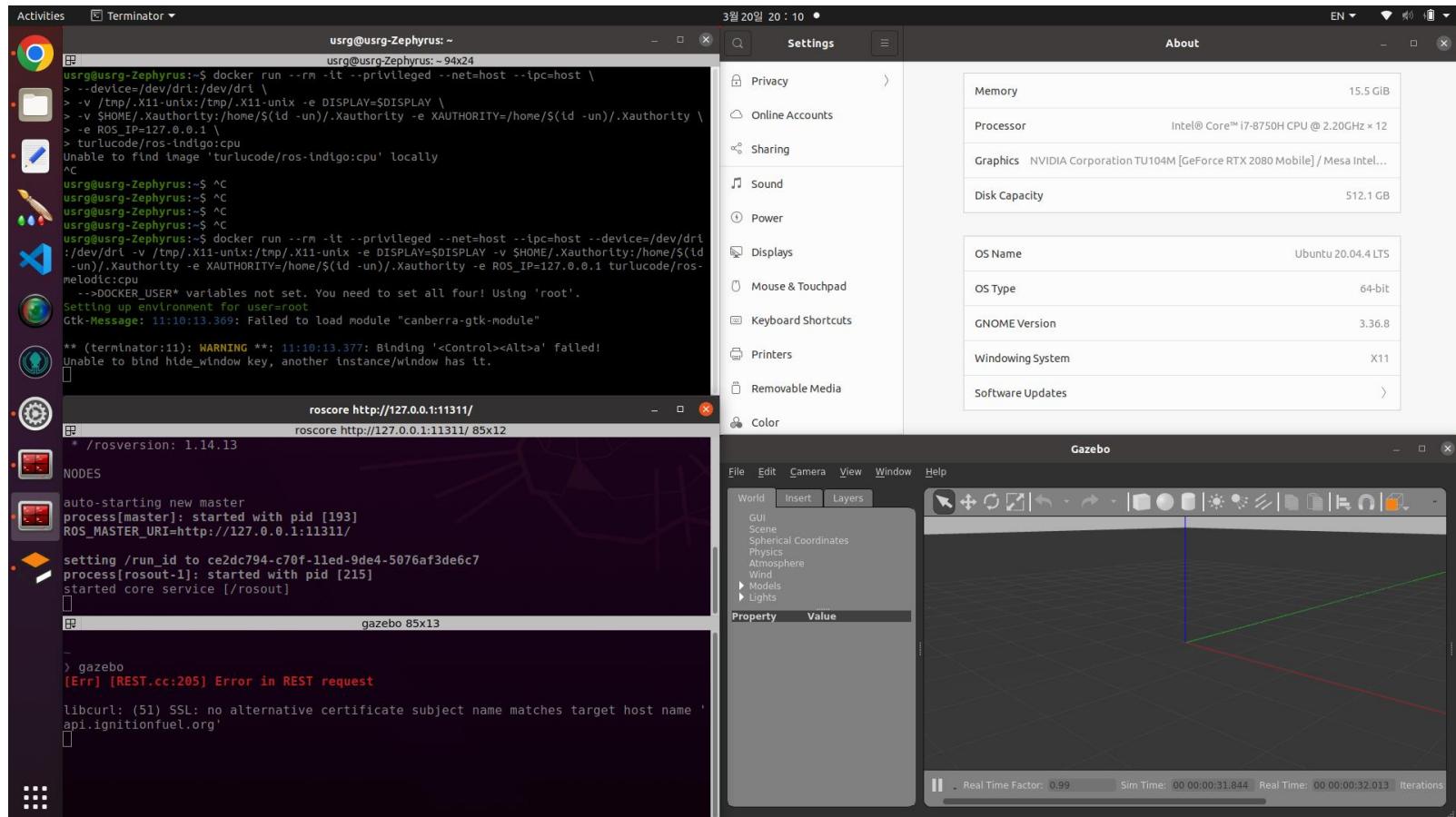
<https://github.com/turlucode/ros-docker-gui>

- Install docker.
- git clone <https://github.com/turlucode/ros-docker-gui>
- cd ros-docker-gui
- make cpu\_ros\_melodic
- docker run --rm -it --privileged --net=host --ipc=host --device=/dev/dri:/dev/dri \  
-v /tmp/.X11-unix:/tmp/.X11-unix -e DISPLAY=\$DISPLAY \  
-v \$HOME/.Xauthority:/home/\$(id -un)/.Xauthority -e XAUTHORITY=/home/\$(id -un)/.Xauthority -e ROS\_IP=127.0.0.1 turlucode/ros-melodic:cpu

# Ubuntu Installation

## ❖ Ubuntu 18.04 Installation

### 3. (Optional) Run a customized ROS docker container



---

---

# ROS

# Background

# Robotics Operating System(ROS)

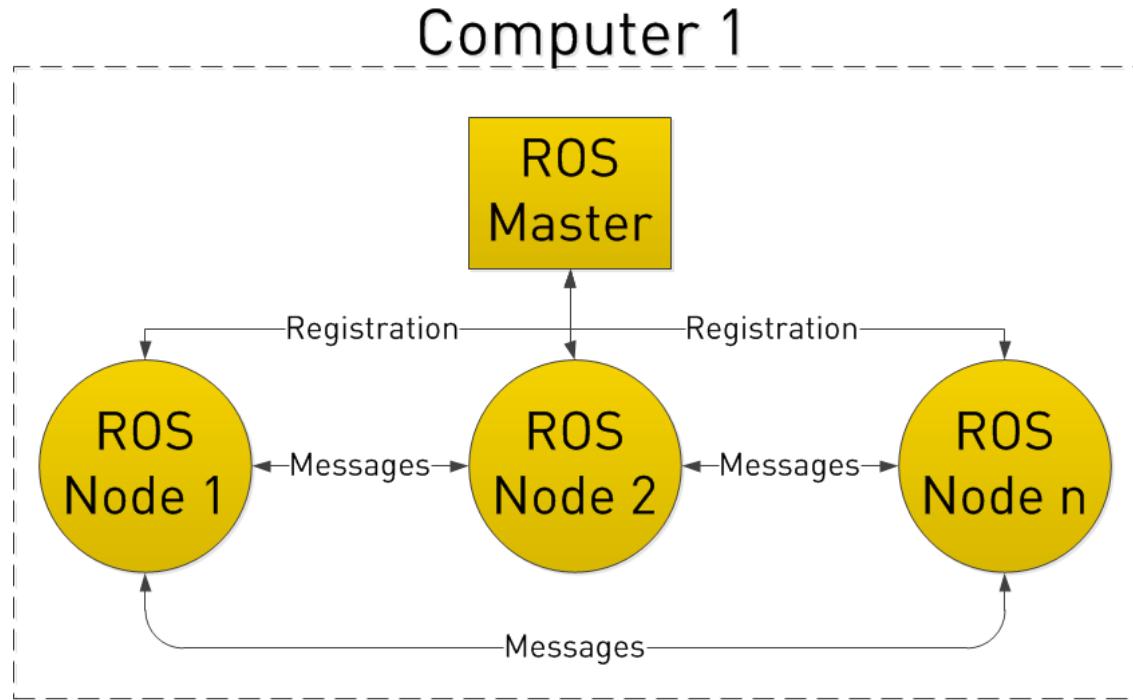
## ❖ What is ROS?

- ❑ The Robot Operating System(ROS) is a flexible framework **for writing robot software**.
- ❑ It is a collection of **tools, libraries, and conventions** that aim to **simplify the task of creating complex and robust robot behavior** across a wide variety of robotic platforms.
- ❑ It has a large **ecosystem** to encourage **collaborative robotics software development**.



ROS distributions for Ubuntu 16.04(Kinetic), 18.04(Melodic), 20.04(Noetic)

# Robotics Operating System(ROS)



- ❖ **ROS Master** stores topics registration information for ROS nodes. Nodes communicate with the Master to report their registration information.
  - ※ **registration info** : node name, msg info to publish, msg info to subscribe, ...
- **Nodes** are connected to other nodes directly; the Master only provides lookup information, much like a DNS server.

Reference : <http://www.clearpathrobotics.com/assets/guides/kinetic/ros/Intro%20to%20the%20Robot%20Operating%20System.html>

# Robotics Operating System(ROS)

## ❖ Important concepts in ROS

- ❑ ROS is processed in the **peer-to-peer network** that are processing data together.
- ❑ Important terminologies: **Node, Master, Message, Topic, Bag, ...**
- ❑ **Nodes** : Nodes are processes that perform computation. A robotic system will usually comprise many nodes; one node to control a laser range-finder, one node to control wheel motors, one node to perform localization, control, and so on.
- ❑ **Master** : Master provides name registration of all nodes and lookup information, much like a DNS server.
- ❑ **Message** : Nodes communicate with each other by **passing messages**. A message is simply a data structure, comprising typed fields.
- ❑ **Topic** : The topic is a **name** that is used to identify the content of the message.
- ❑ **Bags** : Bags are a format for **saving and playing back ROS message data**. Bags are important mechanism for storing data, such as sensor data, that can be difficult to collect but is necessary for developing and testing algorithms.

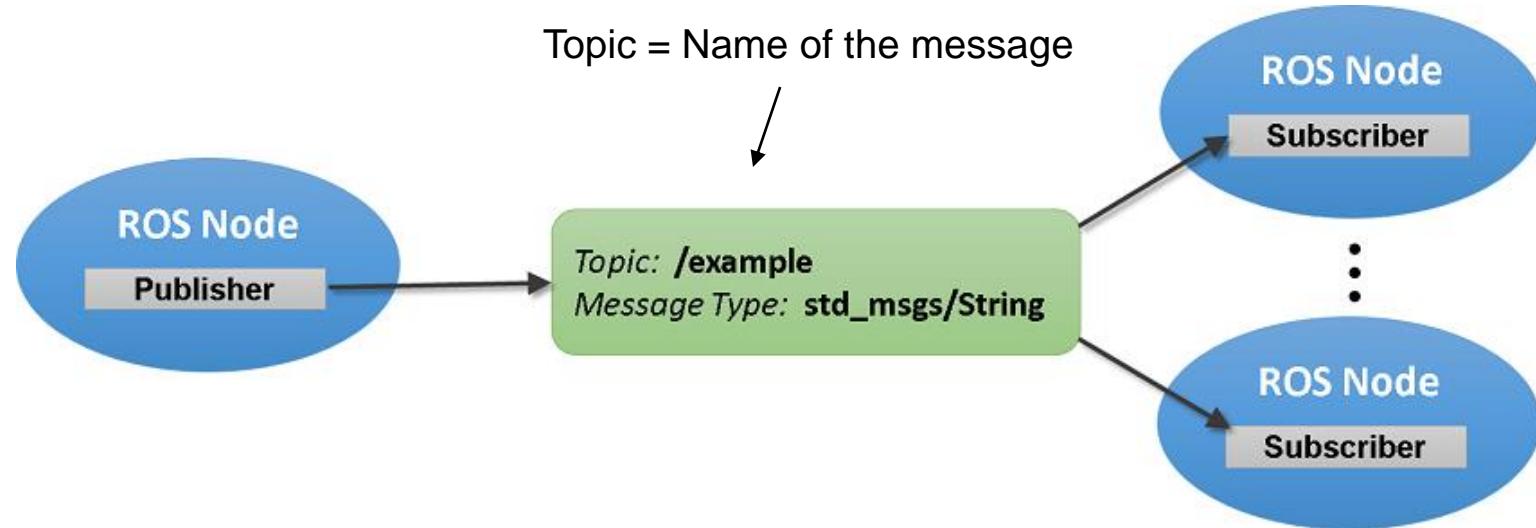
Reference : [http://library.isr.ist.utl.pt/docs/roswiki/ROS\(2f\)Concepts.html](http://library.isr.ist.utl.pt/docs/roswiki/ROS(2f)Concepts.html)



# Robotics Operating System(ROS)

## ❖ Message, Topic

- ❑ Messages are routed via a transport system with **publish / subscribe** semantics.
- ❑ In message type, standard primitive types (Integer, Floating point, Boolean, etc.) are supported, as are arrays of primitives types.
- ❑ Messages can be arbitrary nested structures and arrays (much like **structs in programming languages**).

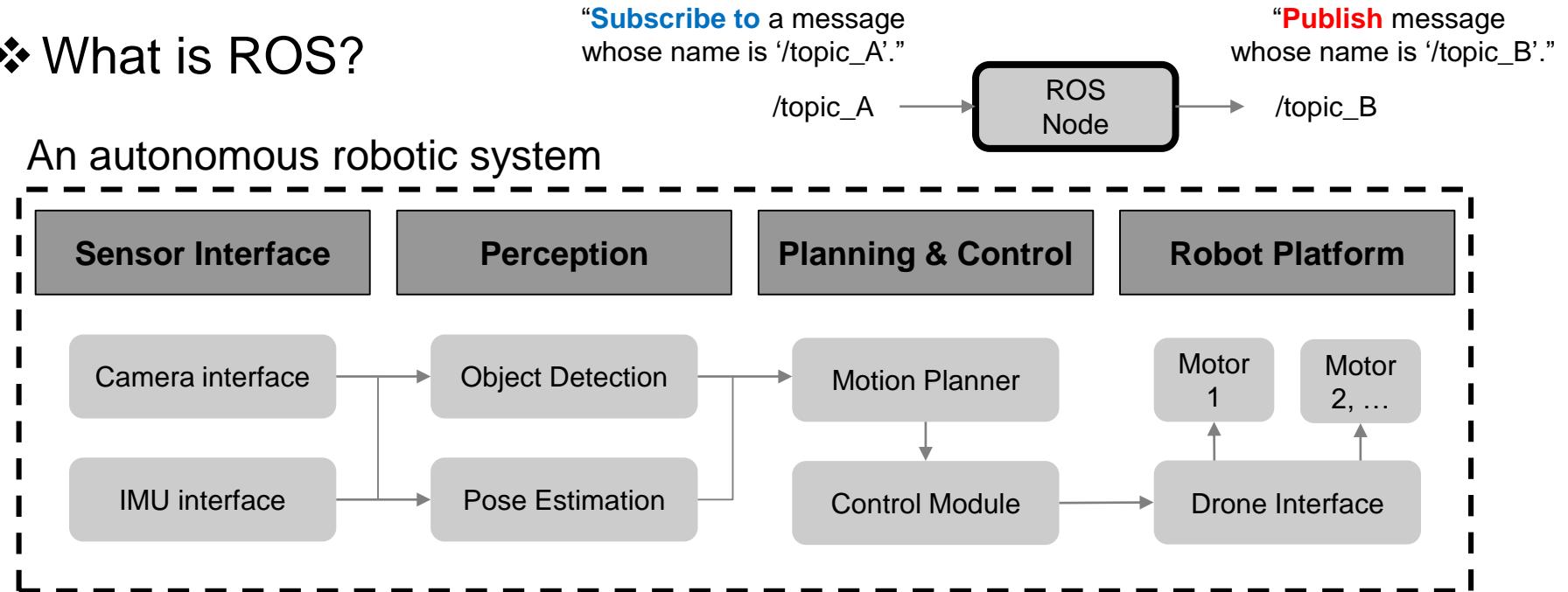


Reference : <https://kr.mathworks.com/help/ros/ug/exchange-data-with-ros-publishers-and-subscribers.html>

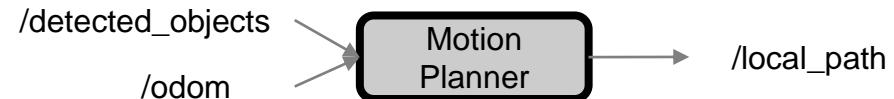
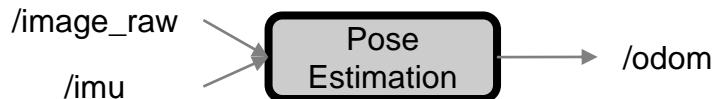
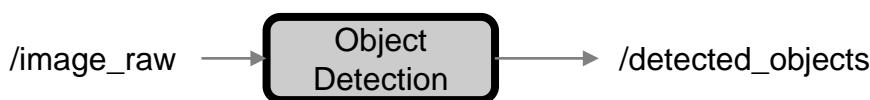
# Robotics Operating System(ROS)

## ❖ What is ROS?

An autonomous robotic system



Examples of ROS nodes for robotic modules



---

---

# ROS Tutorial

1. Installation & Configuration of ROS
2. ROS Package Tutorial
3. ROS Topic, Commands



# Installation & Configuration of ROS

- ❖ Simply follow the homepage installation tutorial.
- ❖ Install homepage: <http://wiki.ros.org/ROS/Installation>.

[ROS.org](#)

About | Support | Discussion Forum | Index | Service Status | Q&A answers.ros.org

Documentation    Browse Software    News    Download

### ROS/ Installation

Available Translations: German | Spanish | French | Italian | Japanese | Korean | Brazilian Portuguese | Portuguese | Русский (Russian) | Thai | Turkish | 简体中文 | Ukrainian | Vietnamese | العربية

#### 1. See Also:

1. [ROS/Installation](#) (this page)
2. [Distributions](#)
3. [Installation](#)

### ROS Installation Options

There is more than one ROS distribution supported at a time. Some are older releases with long term support, making them more stable, while others are newer with shorter support life times, but with binaries for more recent platforms and more recent versions of the ROS packages that make them up. See the [Distributions](#) page for more details. We recommend one of the versions below:

<a href="#">ROS Melodic Morenia</a> Released May, 2018 LTS, supported until May, 2023 Recommended for Ubuntu 18.04	<a href="#">ROS Noetic Nijenjems</a> Released May, 2020 Latest LTS, supported until May, 2025 Recommended for Ubuntu 20.04
---	---



#### ROS 2 Documentation

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

Wiki

Distributions

[ROS/Installation](#)

ROS/Tutorials

RecentChanges

---

Page

Immutable Page

Info

Attachments

More Actions:

Raw Text

Do

---

User

Login

**ROS Melodic (for Ubuntu 18.04)**

### ROS 2 Installation

If you're looking for ROS 2 installation pages please see [docs.ros.org](#)

# Installation & Configuration of ROS

- ❖ Simply follow the homepage installation tutorial.
- ❖ Install homepage: <http://wiki.ros.org/ROS/Installation>.

## melodic/ Installation

### ROS Melodic installation instructions

These instructions will install the ROS Melodic Morenia distribution, which is available for Ubuntu Artful (17.10), Bionic (18.04 LTS) and Debian Stretch, among other platform options.

To install our previous long-term support release, ROS Kinetic Kame, please see the [Kinetic installation instructions](#).

The links below contain instructions for installing ROS Melodic Morenia on various operating systems.

#### Select Your Platform

Supported:



Source installation

Experimental:



The following links are referring to previous ROS distributions installation instructions and have not been updated since.



Wiki

- Distributions
- ROS/Installation
- ROS/Tutorials
- RecentChanges
- melodic/installation

Page

- Immutable Page
- Info
- Attachments
- More Actions:

Raw Text

Do

Login



ROS.org

About | Support | Discussion Forum | Service Status | Q&A answers.ros.org

Search:  Submit

Documentation    Browse Software    News    Download

melodic/ Installation/ Ubuntu

### Ubuntu install of ROS Melodic

We are building Debian packages for several Ubuntu platforms, listed below. These packages are more efficient than source-based builds and are our preferred installation method for Ubuntu. Note that there are also packages available from Ubuntu upstream. Please see [UpstreamPackages](#) to understand the difference.

Ubuntu packages are built for the following distros and architectures.

Distro	amd64	arm64	armhf
Artful	X		
Bionic	X	X	X

If you need to install from source (not recommended), please see [source \(download-and-compile\) installation instructions](#).

If you rely on these packages, please support OSRF.

These packages are built and hosted on infrastructure maintained and paid for by the [Open Source Robotics Foundation](#), a 501(c)(3) non-profit organization. If OSRF were to receive one penny for each downloaded package for just two months, we could cover our annual costs to manage, update, and host all of our online services. Please consider [donating to OSRF today](#).

Contents

- 1 Ubuntu install of ROS Melodic
- 1 Installation
  - 1 Configure your Ubuntu repositories
  - 2 Set up your sources list
  - 3 Set up your keys
  - 4 Installation
  - 5 Environment setup
  - 6 Dependencies for building packages
  - 7 Build farm status
- 2 Tutorials

#### 1. Installation

##### 1.1 Configure your Ubuntu repositories

# Installation & Configuration of ROS

## 1. Installation

### 1.1 Configure your Ubuntu repositories

Configure your Ubuntu repositories to allow "restricted," "universe," and "multiverse." You can [follow the Ubuntu guide](#) for instructions on doing this.

### 1.2 Setup your sources.list

Setup your computer to accept software from packages.ros.org.

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

Mirrors Source Debs are also available

### 1.3 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 -
```

### 1.4 Installation

First, make sure your Debian package index is up-to-date:

```
sudo apt update
```

There are many different libraries and tools in ROS. We provided four default configurations to get you started. You can also install ROS packages individually.

In case of problems with the next step, you can use following repositories instead of the ones mentioned above [ros-shadow-fixed](#)

Desktop-Full Install: (Recommended) : ROS, [rqt](#), [rviz](#), robot-generic libraries, 2D/3D simulators and 2D/3D perception

```
sudo apt install ros-melodic-desktop-full
```

or [click here](#)

## Reference :

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

### ➤ Setup your sources.list

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

### ➤ Set up your keys

```
sudo apt install curl
```

```
curl -s https://raw.githubusercontent.com/ros/rosdistro/master/ros.asc | sudo apt-key add -
```

### ➤ Update your Ubuntu package index

```
sudo apt update
```

### ➤ Desktop-Full install

```
sudo apt install ros-melodic-desktop-full
```

# Installation & Configuration of ROS

## 1.5 Environment setup

It's convenient if the ROS environment variables are automatically added to your bash session every time a new shell is launched:

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

If you have more than one ROS distribution installed, `~/.bashrc` must only source the `setup.bash` for the version you are currently using.

If you just want to change the environment of your current shell, instead of the above you can type:

```
source /opt/ros/melodic/setup.bash
```

If you use zsh instead of bash you need to run the following commands to set up your shell:

```
echo "source /opt/ros/melodic/setup.zsh" >> ~/.zshrc  
source ~/.zshrc
```

## 1.6 Dependencies for building packages

Up to now you have installed what you need to run the core ROS packages. To create and manage your own ROS workspaces, there are various tools and requirements that are distributed separately. For example, `rosinstall` is a frequently used command-line tool that enables you to easily download many source trees for ROS packages with one command.

To install this tool and other dependencies for building ROS packages, run:

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

### 1.6.1 Initialize rosdep

Before you can use many ROS tools, you will need to initialize rosdep. rosdep enables you to easily install system dependencies for source you want to compile and is required to run some core components in ROS. If you have not yet installed rosdep, do so as follows.

```
sudo apt install python-rosdep
```

With the following, you can initialize rosdep.

```
sudo rosdep init  
rosdep update
```

## 1.7 Build farm status

The packages that you installed were built by the [ROS build farm](#). You can check the status of individual packages [here](#).

## Reference :

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

### ➤ Environment setup

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

You can check the above added line as below

```
gedit ~/.bashrc
```

※ `'~/.bashrc'` is same with `'/home/user/.bashrc'`

### ➤ Install dependencies for building packages

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

### ➤ Initialize rosdep

```
sudo apt install python-rosdep
```

```
sudo rosdep init  
rosdep update
```

# Installation & Configuration of ROS

## 3. Create a ROS Workspace

catkin rosbuild

These instructions are for ROS Groovy and later. For ROS Fuerte and earlier, select rosbuild.

Let's create and build a [catkin workspace](#):

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

The [catkin\\_make](#) command is a convenience tool for working with [catkin workspaces](#). Running it the first time in your workspace, it will create a CMakeLists.txt link in your 'src' folder.

Python 3 users in ROS Melodic and earlier: note, if you are building ROS from source to achieve Python 3 compatibility, and have setup your system appropriately (ie: have the Python 3 versions of all the required ROS Python packages installed, such as catkin) the first [catkin\\_make](#) command in a clean [catkin workspace](#) must be:

```
$ catkin_make -DPYTHON_EXECUTABLE=/usr/bin/python3
```

This will configure [catkin\\_make](#) with Python 3. You may then proceed to use just [catkin\\_make](#) for subsequent builds.

Additionally, if you look in your current directory you should now have a 'build' and 'devel' folder. Inside the 'devel' folder you can see that there are now several setup.\*sh files. Sourcing any of these files will overlay this workspace on top of your environment. To understand more about this see the general catkin documentation: [catkin](#). Before continuing source your new setup.\*sh file:

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

To make sure your workspace is properly overlayed by the setup script, make sure `ROS_PACKAGE_PATH` environment variable includes the directory you're in.

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

### Reference :

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

#### ➤ Create and build a catkin workspace (~/[catkin\\_ws](#))

cd ~  
mkdir -p ~/catkin\_ws/src  
"go to home directory"  
"make a workspace (catkin\_ws) and source (src) directory"  
  
cd ~/catkin\_ws  
catkin\_make  
"go back to workspace"  
"do catkin\_make"

※ 'catkin\_make' command should be executed in workspace (~/[catkin\\_ws](#)), not src directory.

#### ➤ Sourcing the files in devel after [catkin\\_make](#) process

source devel/setup.bash

※ Usually, people add above line in '~/.bashrc' so that they don't need do 'source devel/setup.bash' every [catkin\\_make](#) process.  
Open the bashrc file (gedit ~/.bashrc) and add the line (source devel/setup.bash) in it.

# ROS Package Tutorial – TurtleSim

**Reference :**

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

## ➤ Prerequisites

sudo apt-get install ros-melodic-ros-tutorials

or

sudo apt-get install ros-kinetic-ros-tutorials

"for Ubuntu 18.04"

"for Ubuntu 16.04"

## 1. Prerequisites

For this tutorial we'll use a lightweight simulator, to install it run the following command:

```
$ sudo apt-get install ros-<distro>-ros-tutorials
```

Replace '<distro>' with the name of your ROS distribution (e.g. indigo, jade, kinetic)

## ➤ Run roscore (in terminal 1)

roscore

## ➤ Run tutorial ros package (in another terminal 2)

rosrun turtlesim turtlesim\_node

※ rosrun [package\_name] [node\_name]



# ROS Package Tutorial – TurtleSim

## 1. Setup

### 1.1 roscore

Let's start by making sure that we have roscore running. **In a new terminal:**

```
$ roscore
```

If you left roscore running from the last tutorial, you may get the error message:

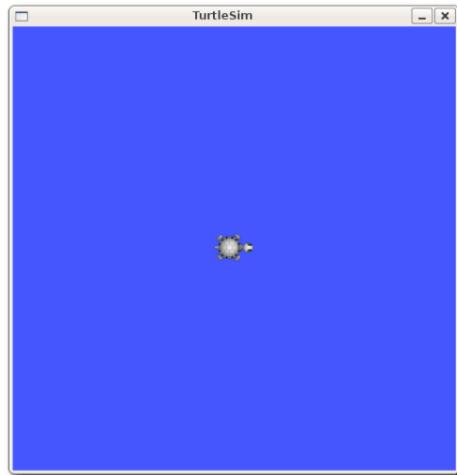
```
roscore cannot run as another roscore/master is already running.  
please kill other roscore/master processes before relaunching
```

This is fine. Only one roscore needs to be running.

### 1.2 turtlesim

For this tutorial we will also use turtlesim. Please run **in a new terminal:**

```
$ rosrun turtlesim turtlesim_node
```



## Reference :

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

### ➤ Prerequisites

sudo apt-get install ros-melodic-ros-tutorials  
or  
sudo apt-get install ros-kinetic-ros-tutorials

"for Ubuntu 18.04"

"for Ubuntu 16.04"

### ➤ Run roscore (in terminal 1)

roscore

### ➤ Run tutorial ros package (in another terminal 2)

rosrun turtlesim turtlesim\_node

※ rosrun [package\_name] [node\_name]



# ROS Package Tutorial – TurtleSim

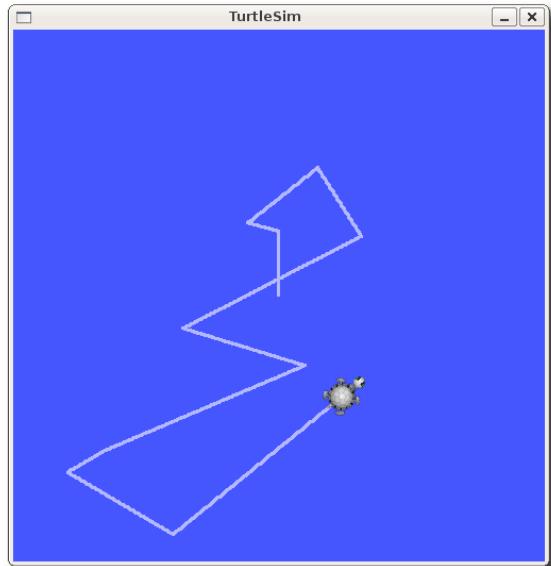
## 1.3 turtle keyboard teleoperation

We'll also need something to drive the turtle around with. Please run in a new terminal:

```
$ rosrun turtlesim turtle_teleop_key
```

```
[ INFO] 1254264546.878445000: Started node [/teleop_turtle], pid [5528], bound on [tcp], xm  
l rpc port [43918], tcpros port [55936], logging to [~/ros/ros/log/teleop_turtle_5528.log],  
using [real] time  
Reading from keyboard  
-----  
Use arrow keys to move the turtle.
```

Now you can use the arrow keys of the keyboard to drive the turtle around. If you can not drive the turtle **select the terminal window of the turtle\_teleop\_key** to make sure that the keys that you type are recorded.



### Reference :

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

### ➤ Run turtle keyboard package (in a new terminal 3)

```
rosrun turtlesim turtle_teleop_key
```

You can use the arrow keys of keyboard to  
drive the turtle around!

# ROS Topic



Reference :

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

- Run rqt\_graph for checking the nodes and topics currently running (in a new terminal 4)

rosrun rqt\_graph rqt\_graph

or simply

rqt\_graph

- You can see the name of the publisher node (/teleop\_turtle) and subscriber node (/turtlesim)

- The topic message name (/turtle1/command\_velocity) is represented as an arrow from the publisher to subscriber

# ROS Topic

## 2.2 Introducing rostopic

The rostopic tool allows you to get information about ROS topics.

You can use the help option to get the available sub-commands for rostopic

```
$ rostopic -h
rostopic bw      display bandwidth used by topic
rostopic echo    print messages to screen
rostopic hz      display publishing rate of topic
rostopic list    print information about active topics
rostopic pub     publish data to topic
rostopic type   print topic type
```

Or pressing tab key after rostopic prints the possible sub-commands:

```
$ rostopic
bw  echo  find  hz  info  list  pub  type
```

Let's use some of these topic sub-commands to examine turtlesim.

## 2.3 Using rostopic echo

rostopic echo shows the data published on a topic.

Usage:

```
rostopic echo [topic]
```

### Reference :

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

➤ “**rostopic**” allows you to get information about ROS topics

- rostopic list : print the list of active topics
- rostopic echo : print messages of specific topic
- rostopic pub : publish message data to topic
- rostopic type : print message type of specific topic
- rostopic hz : display publishing rate of topic

※ In order of frequent use



# ROS Topic

## 2.3 Using rostopic echo

rostopic echo shows the data published on a topic.

Usage:

```
rostopic echo [topic]
```

Let's look at the command velocity data published by the turtle\_teleop\_key node.

For ROS Hydro and later, this data is published on the /turtle1/cmd\_vel topic. In a new terminal, run:

```
$ rostopic echo /turtle1/cmd_vel
```

For ROS Groovy and earlier, this data is published on the /turtle1/command\_velocity topic. In a new terminal, run:

```
$ rostopic echo /turtle1/command_velocity
```

You probably won't see anything happen because no data is being published on the topic. Let's make turtle\_teleop\_key publish data by pressing the arrow keys. Remember if the turtle isn't moving you need to select the turtle\_teleop\_key terminal again.

For ROS Hydro and later, you should now see the following when you press the up key:

```
linear:  
  x: 2.0  
  y: 0.0  
  z: 0.0  
angular:  
  x: 0.0  
  y: 0.0  
  z: 0.0  
---  
linear:  
  x: 2.0  
  y: 0.0  
  z: 0.0  
angular:  
  x: 0.0  
  y: 0.0  
  z: 0.0  
---
```

## Reference :

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

- Use “rostopic echo” for displaying the command velocity topic from the turtle keyboard package, turtle\_teleop\_key.

rostopic echo /turtle1/cmd\_vel

※ rostopic echo {topic\_name}

- When you press the arrow keys, you will see the message published from the turtle\_teleop\_key package.

```
linear:  
  x: 2.0  
  y: 0.0  
  z: 0.0  
angular:  
  x: 0.0  
  y: 0.0  
  z: 0.0
```



# ROS Commands

➤ **roscore** : run the ROS Master Node.

➤ **`rosrun [package_name] [node_name]`**: run a ROS package node.

- rosrun turtlesim turtle\_sim\_node
  - rosrun turtlesim turtle\_teleop\_key

- ✓ Do not need to memorize commands.
- ✓ Just use double ‘Tap key’ to display available commands or auto complete.

➤ **rostopic** : allows you to get information about ROS topics.

- **rostopic -h** : display the sub-commands for 'rostopic'.  
(bw, echo, find, hz, info, list, pub, type)
- **rostopic list** : print information about active topics.
- **rostopic echo [topic\_name]** : print messages to screen.
- **rostopic pub [topic\_name] [message\_type] [data]**  
: publish data to topic.

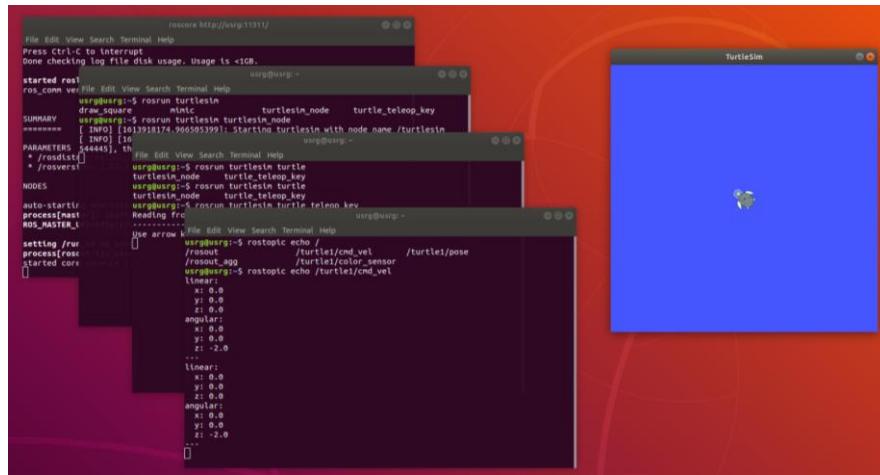


# Tips – Terminator

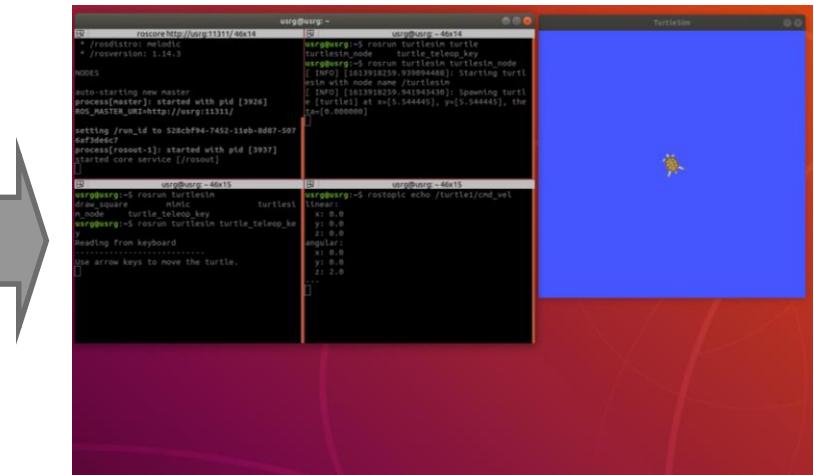
## ➤ Recommend Terminal Program in Ubuntu: Terminator

- In many case, programmer needs to **process multiple programs** in Ubuntu.
- Opening new terminal window whenever you need to run new one is inconvenient.
- Terminator has a function to **split a terminal window so that we can run multiple programs in one window**.
- **Installation:**

```
sudo apt-get install terminator
```



Default Terminal Windows



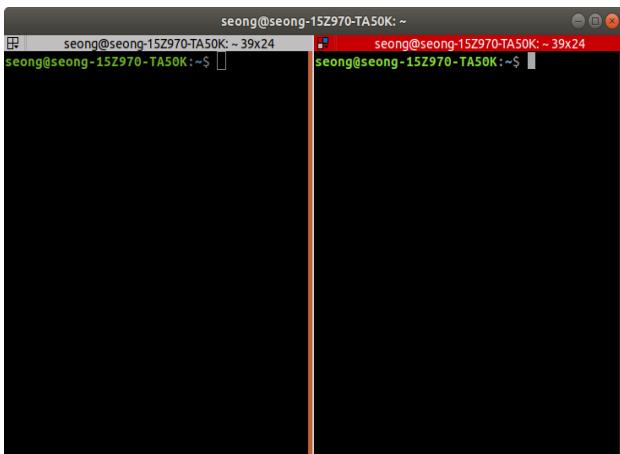
Terminator window



# Tips – Terminator

## ➤ Recommend Terminal Program in Ubuntu: Terminator

- **Ctrl + Alt + T** : open a terminal window (same with default terminal of Ubuntu).
- **Ctrl + Shift + E** : split terminal window vertically.
- **Ctrl + Shift + O** : split terminal window horizontally.
- **Ctrl + Shift + T** : add new terminal window tab.  
→ Convenient to run and control multiple ros packages in one window!



seong@seong-15Z970-TA50K: ~

seong@seong-15Z970-TA50K: ~ 39x24

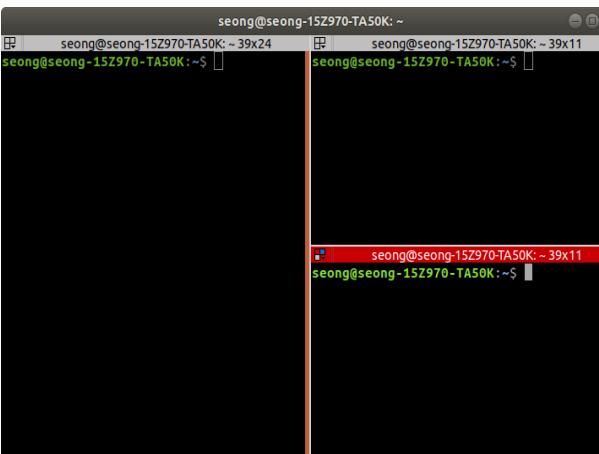
seong@seong-15Z970-TA50K: ~\$

seong@seong-15Z970-TA50K: ~ 39x24

seong@seong-15Z970-TA50K: ~\$

seong@seong-15Z970-TA50K: ~\$

Ctrl + Shift + E



seong@seong-15Z970-TA50K: ~

seong@seong-15Z970-TA50K: ~ 39x24

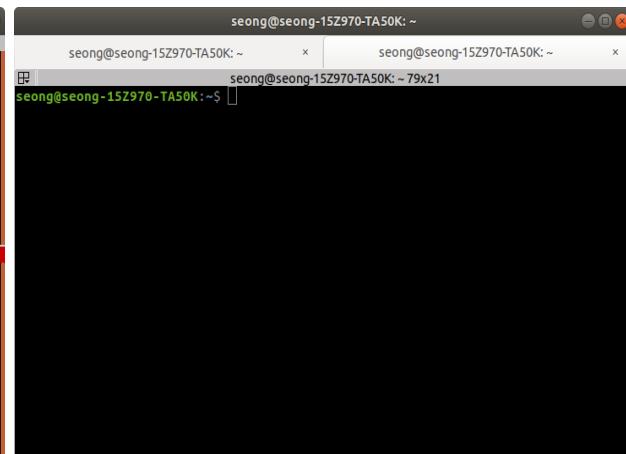
seong@seong-15Z970-TA50K: ~\$

seong@seong-15Z970-TA50K: ~ 39x11

seong@seong-15Z970-TA50K: ~\$

seong@seong-15Z970-TA50K: ~\$

Ctrl + Shift + O



seong@seong-15Z970-TA50K: ~

seong@seong-15Z970-TA50K: ~

seong@seong-15Z970-TA50K: ~ 79x21

seong@seong-15Z970-TA50K: ~\$

Ctrl + Shift + T



---

---

# ROS Programming

# ROS Programming

## ➤ Creating a catkin package

Reference :

<http://wiki.ros.org/catkin/Tutorials/CreatingPackage>

### 3. Creating a catkin Package

This tutorial will demonstrate how to use the `catkin_create_pkg` script to create a new catkin package, and what you can do with it after it has been created.

First change to the source space directory of the catkin workspace you created in the [Creating a Workspace for catkin tutorial](#):

```
# You should have created this in the Creating a Workspace Tutorial  
$ cd ~/catkin_ws/src
```

Now use the `catkin_create_pkg` script to create a new package called 'beginner\_tutorials' which depends on std\_msgs, roscpp, and rospy:

```
$ catkin_create_pkg beginner_tutorials std_msgs rospy roscpp
```

This will create a `beginner_tutorials` folder which contains a `package.xml` and a `CMakeLists.txt`, which have been partially filled out with the information you gave `catkin_create_pkg`.

`catkin_create_pkg` requires that you give it a `package_name` and optionally a list of dependencies on which that package depends:

```
# This is an example, do not try to run this  
# catkin_create_pkg <package_name> [depend1] [depend2] [depend3]
```

`catkin_create_pkg` also has more advanced functionalities which are described in [catkin/commands/catkin\\_create\\_pkg](#).

### ➤ Go to the catkin workspace directory

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

### ➤ Create a ROS package

```
catkin_create_pkg beginner_tutorial  
std_msgs rospy
```

※ `catkin_create_pkg [package_name]  
[dependent] [dependent] ...`

# ROS Programming

## ➤ Writing a Publisher Node

```
1 #!/usr/bin/env python ←  
2 # license removed for brevity  
3 import rospy ←  
4 from std_msgs.msg import String ←  
5  
6 def talker():  
7     pub = rospy.Publisher('chatter', String, queue_size=10)  
8     rospy.init_node('talker', anonymous=True)  
9     rate = rospy.Rate(10) # 10hz  
10    while not rospy.is_shutdown():  
11        hello_str = "hello world %s" % rospy.get_time()  
12        rospy.loginfo(hello_str)  
13        pub.publish(hello_str)  
14        rate.sleep()  
15  
16 if __name__ == '__main__':  
17     try:  
18         talker()  
19     except rospy.ROSInterruptException:  
20         pass
```

Every Python ROS node will have this declaration at the top.  
(This make sure your script is executed as a Python script.)

Import 'rospy' library

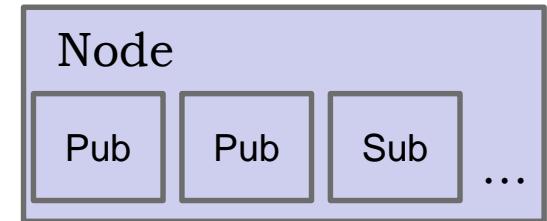
Import String message type

Reference : <http://wiki.ros.org/ROS/Tutorials/WritingPublisherSubscriber%28python%29>

# ROS Programming

## ➤ Writing a Publisher Node

```
1 #!/usr/bin/env python
2 # license removed for brevity
3 import rospy
4 from std_msgs.msg import String
5
6 def talker():
7     pub = rospy.Publisher('chatter', String, queue_size=10) ←
8     rospy.init_node('talker', anonymous=True) ←
9     rate = rospy.Rate(10) # 10hz ←
10    while not rospy.is_shutdown():
11        hello_str = "hello world %s" % rospy.get_time()
12        rospy.loginfo(hello_str)
13        pub.publish(hello_str)
14        rate.sleep() ←
15
16 if __name__ == '__main__':
17     try:
18         talker()
19     except rospy.ROSInterruptException:
20         pass
```



Define a publisher to publish a topic message 'chatter'

Define a node named 'talker'

Set node process rate

Reference : <http://wiki.ros.org/ROS/Tutorials/WritingPublisherSubscriber%28python%29>

# ROS Programming

## ➤ Writing a Subscriber Node

```
1 #!/usr/bin/env python
2 import rospy
3 from std_msgs.msg import String
4
5 def callback(data):
6     rospy.loginfo(rospy.get_caller_id() + "I heard %s", data.data)
7
8 def listener():
9
10    # In ROS, nodes are uniquely named. If two nodes with the same
11    # name are launched, the previous one is kicked off. The
12    # anonymous=True flag means that rospy will choose a unique
13    # name for our 'listener' node so that multiple listeners can
14    # run simultaneously.
15    rospy.init_node('listener', anonymous=True)
16
17    rospy.Subscriber("chatter", String, callback)
18
19    # spin() simply keeps python from exiting until this node is stopped
20    rospy.spin()
21
22 if __name__ == '__main__':
23     listener()
```

Every Python ROS node will have this declaration at the top.  
(This make sure your script is executed as a Python script.)

Import 'rospy' library

Import String message type

Reference : <http://wiki.ros.org/ROS/Tutorials/WritingPublisherSubscriber%28python%29>

# ROS Programming

## ➤ Writing a Subscriber Node

```
1 #!/usr/bin/env python
2 import rospy
3 from std_msgs.msg import String
4
5 def callback(data):
6     rospy.loginfo(rospy.get_caller_id() + "I heard %s", data.data)
7
8 def listener():
9
10    # In ROS, nodes are uniquely named. If two nodes with the same
11    # name are launched, the previous one is kicked off. The
12    # anonymous=True flag means that rospy will choose a unique
13    # name for our 'listener' node so that multiple listeners can
14    # run simultaneously.
15    rospy.init_node('listener', anonymous=True)
16
17    rospy.Subscriber("chatter", String, callback)
18
19    # spin() simply keeps python from exiting until this node is stopped
20    rospy.spin()
21
22 if __name__ == '__main__':
23     listener()
```

Define a node named ‘listener’

Define a subscriber to subscribe the topic message “chatter”

Keep the process until this node is stopped

You can change this to a **while loop with rate.sleep** similar with the ones in the Publisher node

Reference : <http://wiki.ros.org/ROS/Tutorials/WritingPublisherSubscriber%28python%29>

# ROS Programming

## ➤ Writing a Subscriber Node

```
1 #!/usr/bin/env python
2 import rospy
3 from std_msgs.msg import String
4
5 def callback(data):
6     rospy.loginfo(rospy.get_caller_id() + "I heard %s", data.data)
7
8 def listener():
9
10    # In ROS, nodes are uniquely named. If two nodes with the same
11    # name are launched, the previous one is kicked off. The
12    # anonymous=True flag means that rospy will choose a unique
13    # name for our 'listener' node so that multiple listeners can
14    # run simultaneously.
15    rospy.init_node('listener', anonymous=True)
16
17    rospy.Subscriber("chatter", String, callback)
18
19    # spin() simply keeps python from exiting until this node is stopped
20    rospy.spin()
21
22 if __name__ == '__main__':
23     listener()
```

Callback process to subscribe a topic message 'chatter'

A callback function is a function which is:

- passed as an **argument** to another **function**
- **is invoked** after some kind of **event**.  
→
- passed a **topic message** to **callback function**
- **is invoked** after **receiving a message**

Reference :

<https://stackoverflow.com/questions/824234/what-is-a-callback-function>

Reference : <http://wiki.ros.org/ROS/Tutorials/WritingPublisherSubscriber%28python%29>

# ROS Programming

## ➤ Writing a Publisher + Subscriber Node

```
1  #!/usr/bin/env python ← Every Python ROS node will
2  # license removed for brevity have this declaration at the
3  import rospy ← top.
4  from std_msgs.msg import String ← Import 'rospy' library
5
6  class ROS_pub_sub():
7      def __init__(self):
8          # Init ros node
9          rospy.init_node('talker_listener', anonymous=True)
10
11         # Define publisher and subscriber
12         self.sub_chatter_1 = rospy.Subscriber('/chatter', String, self.callback_chatter)
13         self.pub_chatter_2 = rospy.Publisher('/chatter_2', String, queue_size=10)
14         self.pub_processed = rospy.Publisher('/chatter_processed', String, queue_size=10)
15
16         # Define ros node rate
17         self.rate = rospy.Rate(5) # 5hz
18
19     def callback_chatter(self, msg):
20         # Parse the string data in the message
21         chat_data = msg.data
22         rospy.loginfo("I heard %s", chat_data)
23
24         # Process
25         processed_chat_data = chat_data + "_processed"
26
27         # Publish a processed message
28         msg_processed = String()
29         msg_processed.data = processed_chat_data
30         self.pub_processed.publish(msg_processed)
31
```

Every Python ROS node will have this declaration at the top.

Import 'rospy' library

Import String message type

Reference code : [https://github.com/Guri-cccc/EE405A\\_2023/blob/main/Week2/Materials/test\\_package/script/test\\_pub\\_sub.py](https://github.com/Guri-cccc/EE405A_2023/blob/main/Week2/Materials/test_package/script/test_pub_sub.py)

# ROS Programming

## ➤ Writing a Publisher + Subscriber Node

```
1  #!/usr/bin/env python
2  # license removed for brevity
3  import rospy
4  from std_msgs.msg import String
5
6  class ROS_pub_sub():
7      def __init__(self):
8          # Init ros node
9          rospy.init_node('talker_listener', anonymous=True)
10
11         # Define publisher and subscriber
12         self.sub_chatter_1 = rospy.Subscriber('/chatter', String, self.callback_chatter)
13         self.pub_chatter_2 = rospy.Publisher('/chatter_2', String, queue_size=10)
14         self.pub_processed = rospy.Publisher('/chatter_processed', String, queue_size=10)
15
16         # Define ros node rate
17         self.rate = rospy.Rate(5) # 5hz
18
19     def callback_chatter(self, msg):
20         # Parse the string data in the message
21         chat_data = msg.data
22         rospy.loginfo("I heard %s", chat_data)
23
24         # Process
25         processed_chat_data = chat_data + "_processed"
26
27         # Publish a processed message
28         msg_processed = String()
29         msg_processed.data = processed_chat_data
30         self.pub_processed.publish(msg_processed)
31
```

Create a class

Define a node named  
'talker\_listener'

Define two publishers and  
a subscriber

Set node process rate

Reference code : [https://github.com/Guri-cccc/EE405A\\_2023/blob/main/Week2/Materials/test\\_package/script/test\\_pub\\_sub.py](https://github.com/Guri-cccc/EE405A_2023/blob/main/Week2/Materials/test_package/script/test_pub_sub.py)

# ROS Programming

## ➤ Writing a Publisher + Subscriber Node

```
1  #!/usr/bin/env python
2  # license removed for brevity
3  import rospy
4  from std_msgs.msg import String
5
6  class ROS_pub_sub():
7      def __init__(self):
8          # Init ros node
9          rospy.init_node('talker_listener', anonymous=True)
10
11         # Define publisher and subscriber
12         self.sub_chatter_1 = rospy.Subscriber('/chatter', String, self.callback_chatter)
13         self.pub_chatter_2 = rospy.Publisher('/chatter_2', String, queue_size=10)
14         self.pub_processed = rospy.Publisher('/chatter_processed', String, queue_size=10)
15
16         # Define ros node rate
17         self.rate = rospy.Rate(5) # 5hz
18
19     def callback_chatter(self, msg):
20         # Parse the string data in the message
21         chat_data = msg.data
22         rospy.loginfo("I heard %s", chat_data)
23
24         # Process
25         processed_chat_data = chat_data + "_processed"
26
27         # Publish a processed message
28         msg_processed = String()
29         msg_processed.data = processed_chat_data
30         self.pub_processed.publish(msg_processed)
```

The code illustrates a ROS node structure. It starts with initializing the ROS node and defining publishers and subscribers. The `callback_chatter` function handles incoming messages by parsing the data, processing it (adding a suffix), and then publishing the processed data. Annotations explain this flow: 'Parse the message data' points to the message parsing logic; 'Process' points to the modification of the data; and 'For publishing the processed data' points to the final publication step.

Callback process to subscribe to a topic message '**chatter**' and publish a topic message '**chatter\_processed**'

Reference code : [https://github.com/Guri-cccc/EE405A\\_2023/blob/main/Week2/Materials/test\\_package/script/test\\_pub\\_sub.py](https://github.com/Guri-cccc/EE405A_2023/blob/main/Week2/Materials/test_package/script/test_pub_sub.py)

# ROS Programming

## ➤ Writing a Publisher + Subscriber Node

```
31 def main(): ← Define a main function
32     # Create a class instance ← Create a class instance
33     pub_sub_node = ROS_pub_sub()
34
35
36     # Main loop
37     while not rospy.is_shutdown():
38         # Publish a message, chatter_2
39         msg_chatter_2_data = "hello_world v2"
40         msg_chatter_2 = String()
41         msg_chatter_2.data = msg_chatter_2_data } Define a message to publish
42         pub_sub_node.pub_chatter_2.publish(msg_chatter_2) ← Publish the message
43         rospy.loginfo("I sent %s", msg_chatter_2.data)
44
45         # Rate control
46         pub_sub_node.rate.sleep() ← Rate control (5Hz)
47
48 if __name__ == '__main__':
49     main() ← Run the main function
```

Reference code : [https://github.com/Guri-cccc/EE405A\\_2023/blob/main/Week2/Materials/test\\_package/script/test\\_pub\\_sub.py](https://github.com/Guri-cccc/EE405A_2023/blob/main/Week2/Materials/test_package/script/test_pub_sub.py)

# ROS Programming

## ➤ **rosrun / roslaunch**

- **rosrun** command for running a ROS node.
- After implementation, build your packages:
  - ✓ cd ~/catkin\_ws
  - ✓ catkin\_make
  - ✓ Source devel/setup.bash
- Use ‘rosrun’ command for each ROS node.
  - ✓ rosrun test\_package test\_publisher.py
  - ✓ rosrun test\_package test\_subscriber.py
  - ✓ rosrun test\_package test\_pub\_sub.py

※ **rosrun [package\_name] [node\_name]**

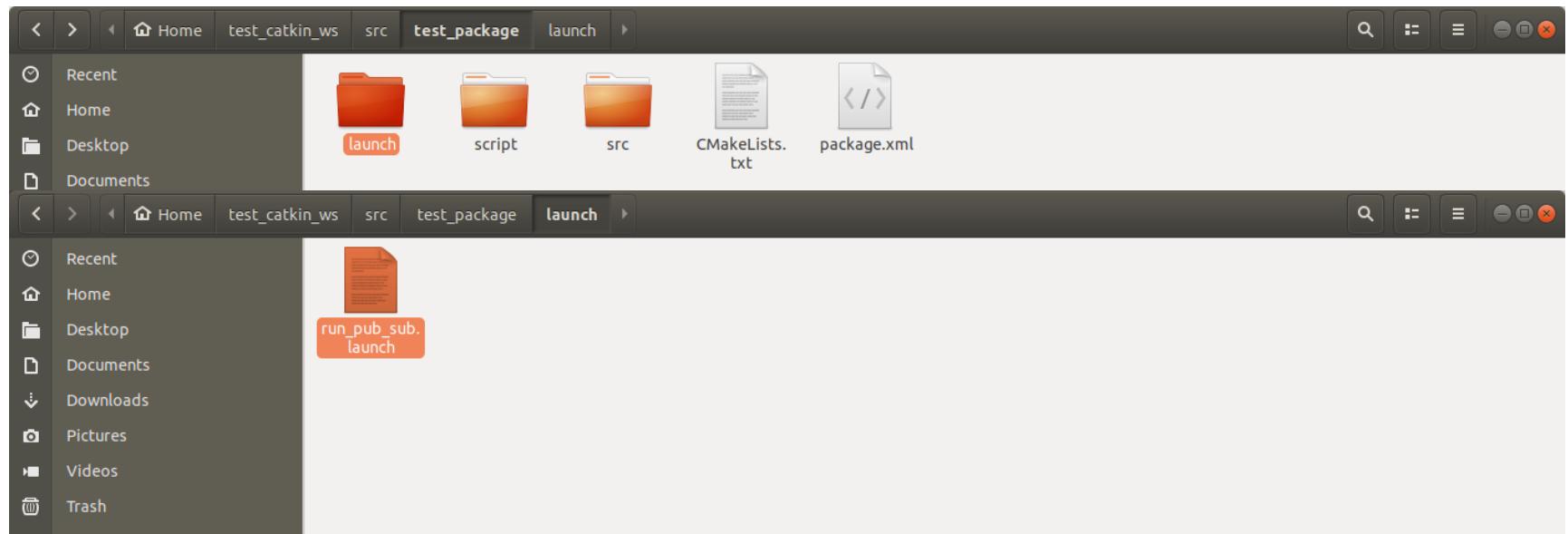
※ if your packages are not recognized, try ‘**rospack profile**’

Reference for ‘rosrun’ : <http://wiki.ros.org/ROS/Tutorials/UnderstandingNodes>

# ROS Programming

## ➤ rosrun / roslaunch

- ❑ **roslaunch** command for **running multiple ros nodes** at once.
- ❑ Make a '**launch**' **directory** in your package folder.
- ❑ Create a '**.launch**' **script** in the launch folder.



Reference for 'roslaunch' : <http://wiki.ros.org/roslaunch>

# ROS Programming

## ➤ rosrun / roslaunch

- You need to write a .launch script for using the ‘roslaunch’ command.

```
1  <launch>
2
3  <!-- Run test_publisher.py -->
4  <node name="talker" pkg="test_package" type="test_publisher.py" output="screen">
5  </node>
6
7  <!-- Run test_pub_sub.py -->
8  <node name="test_pub_sub" pkg="test_package" type="test_pub_sub.py" output="screen">
9  </node>
10
11 </launch>
```

Annotations for the launch file:

- Node name
- Package name
- Node file
- Whether display at window or not

- Use ‘roslaunch’ command for running multiple ROS nodes.

✓ rosrun test\_package run\_pub\_sub.launch

※ [rosrun \[package\\_name\] \[launch\\_file\]](#)

Reference for ‘roslaunch’ : <http://wiki.ros.org/roslaunch>

Reference launch script :

[https://github.com/Guri-cccc/EE405A\\_2023/blob/main/Week2/Materials/test\\_package/launch/run\\_pub\\_sub.launch](https://github.com/Guri-cccc/EE405A_2023/blob/main/Week2/Materials/test_package/launch/run_pub_sub.launch)

# References for ROS

- **Official ROS Tutorials**

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

- **Programming for Robotics (ROS) Course**

- Youtube videos for ROS introductions.

<https://www.youtube.com/watch?v=0BxVPCInS3M&list=PLE-BQwvVGf8HOvwXPgtDfWoxd4Cc6ghiP>

- **The Construct: A Platform to Learn ROS-based Advanced Robotics Online**

- A linux VM-based MOOC platform.
  - Several courses are not free.

<https://www.theconstructsim.com/>

- **Hello (Real) World with ROS – Robot Operating System**

- A MOOC course for ROS in Edx.
  - You can take the course for free by accessing to the audit track.

<https://www.edx.org/course/hello-real-world-with-ros-robot-operating-system>



---

---

# Assignments

# Programming Assignment

## ❖ Create a ROS package with following functions:

- ❑ Create a ROS package
- ❑ Write a ROS publisher node ('simple\_publisher\_node.py')
  - Publish a data whose
    - topic name is '/YourName' (Set topic name as your name)
    - message type is std\_msgs/Int32 (Set the data value as student ID number.)
    - Set publish rate is 30Hz
- ❑ Write a ROS subscriber node ('data\_processor.py')
  - Subscribe to the above publishing topic
  - Using the received sensor data, re-publish a processed data at the callback as
    - topic name is "/Processed\_YourName"
    - message type is std\_msgs/Int32 (You can set the processed data arbitrary.)
- ❑ Run both publisher and subscriber nodes using 'roslaunch'
  - Create a .launch script to run the publisher('simple\_publisher\_node.py') and subscriber node('data\_processor.py').



# Programming Assignment

## ❖ Assignment should include:

- Explain roscore, rosnode, rosrun and rostopic (each 0.25 points)
  - Capture code screen for the publisher and subscriber. (pub. & sub. each 1 points)
  - Capture terminal screen of “rostopic echo /YourName”. (0.5 point)
  - Capture terminal screen of “rostopic info /YourName”. (0.5 point)
  - Capture “rosaunch file” code (1.0 point)
- 
- Please **zip your ROS package and Report** with the following filename.  
EE405A\_[Student ID]\_[Full name]  
(e.g., EE405A\_20215169\_Hyunki\_Seong.zip)

Please submit the assignment in KLMS as a zip file (PDF + codes).

Due date: 2023-09-15 ( 23:59 PM, Friday).



---

---

# Appendix (ROS Tools, Messages)

---

---

# ROS Tools

# ROS Tools

## ❖ Rviz

- ❑ Rviz is a **3D visualization tool** for ROS applications.
- ❑ It provides a view of your robot model, capture sensor information from robot sensors, and replay captured data.
- ❑ It can **display data** from camera, lasers, from 3D and 2D devices including pictures and point clouds.

## ❖ Rosbag

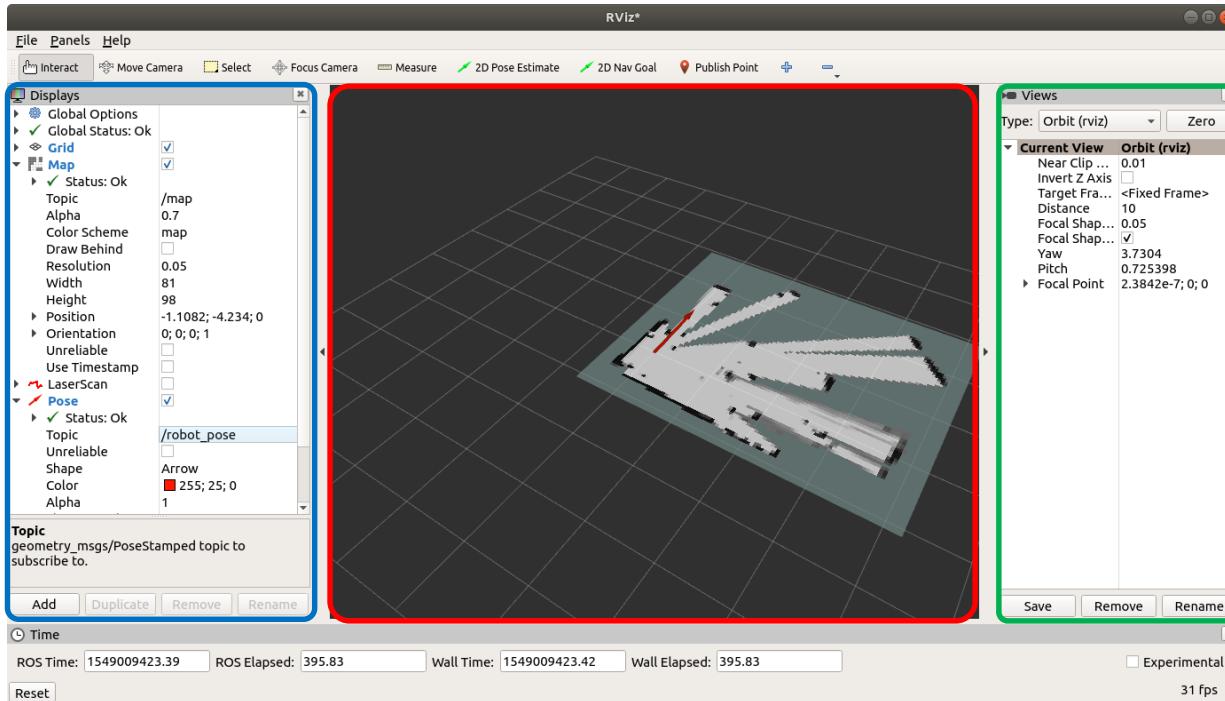
- ❑ This is a set of tools for **recording from and playing back to ROS topics**.
- ❑ The rosbag package provides a command-line tool for working with bags as well as code APIs for reading/writing bags in C++ and Python.
- ❑ It can **record** a bag, **republish** the messages from one or more bags, summarize the contents of a bag, etc.



# ROS Tools

## ➤ Rviz

- Start command : **rosrun rviz rviz** (or simply, **rviz**)



### Displays

- Add messages to display.
- **Topic name** should be matched.
- **Toggle** message displaying on/off.

### Views

- Change the view type.
- Go to initial zero point view.

### 3D View

- Display 3D data.
- Rotate and shift the 3D view.

# ROS Tools

## ➤ Rosbag

### ❑ **rosbag -h**

: display the sub-commands for ‘rosbag’ command.

### ❑ **rosbag record [topic\_name\_1] [topic\_name\_2] ... [topic\_name\_n]**

: record a bag file with the contents of specific topics.

i.e., rosbag record /image/raw /scan /imu /odom

### ❑ **rosbag info [.bag file]**

: summarize the contents of one or more bag files.

i.e., rosbag info test\_210312.bag

### ❑ **rosbag play [.bag file]**

: play back the contents of one of more bag files with time-synchronization.

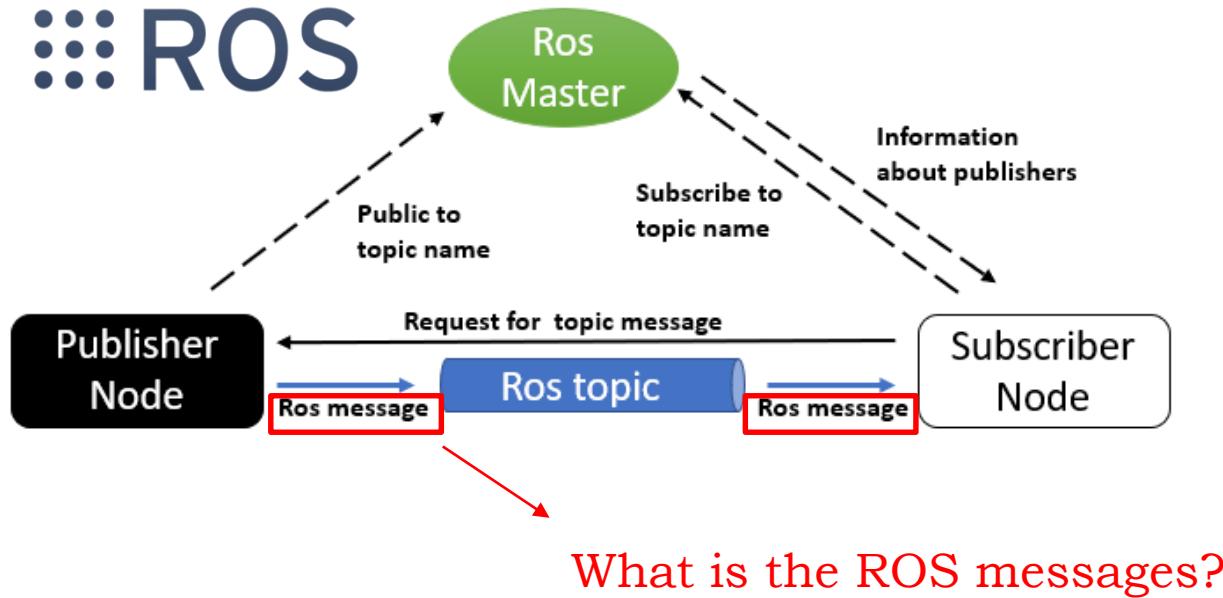
i.e., rosbag play test\_210312.bag

---

---

# ROS Messages

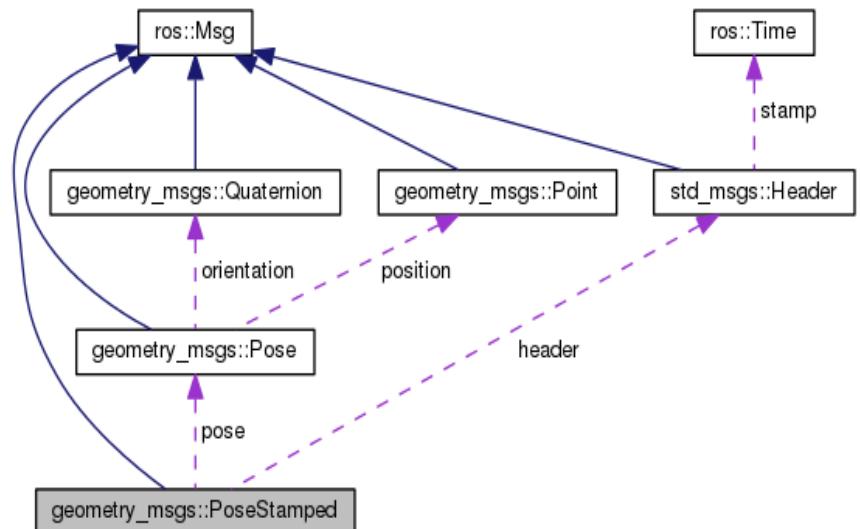
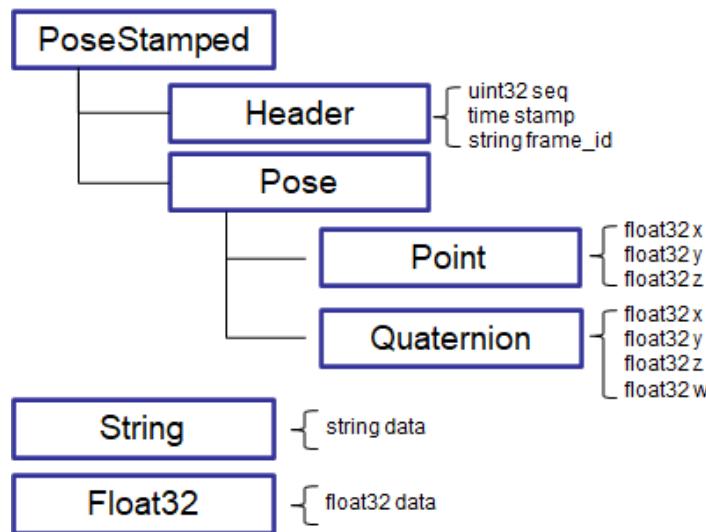
# ROS Messages



# ROS Messages

## ❖ ROS Message Type

- ❑ A message is a simple data structure, comprising typed fields.
- ❑ In message type, standard primitive types (Integer, Floating point, Boolean, etc.) are supported, as are arrays of primitives types.
- ❑ Messages can be arbitrary nested structures and arrays (much like C structs).



Reference : [http://api.kittcar.com/kitt\\_platform/html/classgeometry\\_\\_msgs\\_1\\_1PoseStamped.html](http://api.kittcar.com/kitt_platform/html/classgeometry__msgs_1_1PoseStamped.html)

# ROS Messages

## ❖ ROS Message Type

- ❑ Hierarchical characteristics of the ROS message type.

### geometry\_msgs/PoseStamped Message

File: `geometry_msgs/PoseStamped.msg`

Raw Message Definition

```
# A Pose with reference coordinate frame and timestamp
Header header
Pose pose
```

Compact Message Definition

```
std_msgs/Header header
geometry_msgs/Pose pose
```

### geometry\_msgs/Pose Message

File: `geometry_msgs/Pose.msg`

Raw Message Definition

```
# A representation of pose in free space, composed of position and orientation.
Point position
Quaternion orientation
```

Compact Message Definition

```
geometry_msgs/Point position
geometry_msgs/Quaternion orientation
```



### geometry\_msgs/Point Message

File: `geometry_msgs/Point.msg`

Raw Message Definition

```
# This contains the position of a point in free space
float64 x
float64 y
float64 z
```

Compact Message Definition

```
float64 x
float64 y
float64 z
```

### geometry\_msgs/Quaternion Message

File: `geometry_msgs/Quaternion.msg`

Raw Message Definition

```
# This represents an orientation in free space in quaternion form.
float64 x
float64 y
float64 z
float64 w
```

Compact Message Definition

```
float64 x
float64 y
float64 z
float64 w
```

Reference : [http://docs.ros.org/en/melodic/api/geometry\\_msgs/html/msg/PoseStamped.html](http://docs.ros.org/en/melodic/api/geometry_msgs/html/msg/PoseStamped.html)



# ROS Messages

## ❖ Accessing ROS Message Type Information

- ‘rosmsg show’ command
  - Add ‘-r’ option to display the raw msg definition
- Visit or Google ROS message documentation
  - [http://wiki.ros.org/common\\_msgs](http://wiki.ros.org/common_msgs)

```
sw@sw-Blade:~$ rosmsg show geometry_msgs/PoseStamped
std_msgs/Header header
  uint32 seq
  time stamp
  string frame_id
geometry_msgs/Pose pose
  geometry_msgs/Point position
    float64 x
    float64 y
    float64 z
  geometry_msgs/Quaternion orientation
    float64 x
    float64 y
    float64 z
    float64 w
```

## ❖ Discovering the Message Type of a ROS Topic

- ‘rostopic info’ command
  - Type
  - Publishing Nodes
  - Subscribing Nodes

```
sw@sw-Blade:~$ rostopic info /scout/mavros/vision_pose/pose
Type: geometry_msgs/PoseStamped
Publishers:
* /scout/tf_listener (http://localhost:37495/)
Subscribers: None
```

what is this topic named  
“/scout/mavros/vision\_pose/pose”?

Reference : [http://docs.ros.org/en/melodic/api/geometry\\_msgs/html/msg/PoseStamped.html](http://docs.ros.org/en/melodic/api/geometry_msgs/html/msg/PoseStamped.html)



# ROS Messages

## ❖ ROS messages(rosmsg)

Reference : <http://wiki.ros.org/msg>

- ❑ ROS uses a simplified messages description language for describing the data values.
- ❑ This description makes it easy for ROS tools to automatically generate source code for the message type in several target languages.
- ❑ The format of this language is simple: a message description is a **list of data field descriptions**

Reference : [http://docs.ros.org/en/diamondback/api/std\\_msgs/html/index-msg.html](http://docs.ros.org/en/diamondback/api/std_msgs/html/index-msg.html)

### Built-in types

Primitive Type	Serialization	C++	Python2	Python3
bool (1)	unsigned 8-bit int	uint8_t (2)	bool	
int8	signed 8-bit int	int8_t	int	
uint8	unsigned 8-bit int	uint8_t	int (3)	
int16	signed 16-bit int	int16_t	int	
uint16	unsigned 16-bit int	uint16_t	int	
int32	signed 32-bit int	int32_t	int	
uint32	unsigned 32-bit int	uint32_t	int	
int64	signed 64-bit int	int64_t	long	int
uint64	unsigned 64-bit int	uint64_t	long	int
float32	32-bit IEEE float	float	float	
float64	64-bit IEEE float	double	float	
string	ascii string (4)	std::string	str	bytes
time	secs/nsecs unsigned 32-bit ints	ros::Time	rospy.Time	
duration	secs/nsecs signed 32-bit ints	ros::Duration	rospy.Duration	

### common ROS messages

- std\_msgs
- geometry\_msgs
- nav\_msgs
- visualization\_msgs

### custom ROS messages

- my\_msgs
- your\_msgs
- ee405\_msgs
- kaist\_robot\_msgs

# ROS Messages

## ❖ geometry\_msgs

- geometry\_msgs provides messages for common geometric primitives such as points, vectors, and poses. These primitives are designed to provide a common data type and facilitate interoperability throughout the system.

Reference :

[http://docs.ros.org/en/diamondback/api/geometry\\_msgs/html/index-msg.html](http://docs.ros.org/en/diamondback/api/geometry_msgs/html/index-msg.html)

Message	Basic Information	Description	rviz
geometry_msgs/PoseStamped	Header header Pose pose  geometry_msgs/Point position geometry_msgs/Quaternion orientation	A Pose with reference coordinate frame and timestamp	
geometry_msgs/Quaternion	float64 x float64 y float64 z float64 w	This represents an orientation in free space in quaternion form.	-
geometry_msgs/Point	float64 x float64 y float64 z	This contains the position of a point in free space	-
geometry_msgs/Twist	Vector3 linear Vector3 angular	This expresses velocity in free space broken into it's linear and angular parts.	-

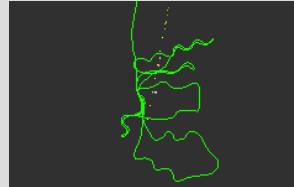
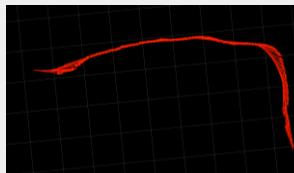
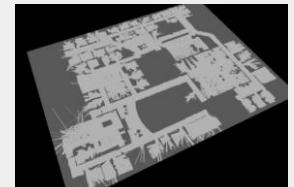
# ROS Messages

## ❖ nav\_msgs

- nav\_msgs defines the common messages used to interact with the navigation stack.

Reference :

[http://docs.ros.org/en/diamondback/api/nav\\_msgs/html/index-msg.html](http://docs.ros.org/en/diamondback/api/nav_msgs/html/index-msg.html)

Message	Basic Information	Description	rviz
nav_msgs/Path	Header header geometry_msgs/PoseStamped[] poses	An array of poses that represents a Path for a robot to follow	
nav_msgs/Odometry	Header header string child_frame_id geometry_msgs/PoseWithCovariance pose geometry_msgs/TwistWithCovariance twist	This represents an estimate of a position and velocity in free space.	
nav_msgs/MapMetaData	time map_load_time float32 resolution uint32 width uint32 height geometry_msgs/Pose origin	This hold basic information about the characteristics of the OccupancyGrid	-
nav_msgs/OccupancyGrid	Header header MapMetaData info int8[] data	This represents a 2-D grid map, in which each cell represents the probability of occupancy.	

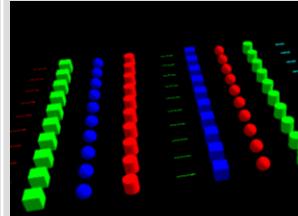
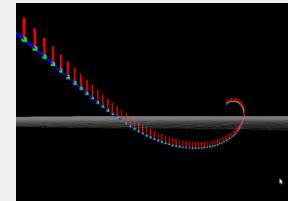
# ROS Messages

## ❖ visualization\_msgs

- ❑ visualization\_msgs is a set of messages used by higher level packages, such as rviz, that deal in visualization-specific data.

Reference :

[http://docs.ros.org/en/melodic/api/visualization\\_msgs/html/index-msg.html](http://docs.ros.org/en/melodic/api/visualization_msgs/html/index-msg.html)

Message	Basic Information	Description	rviz
Visualization_msgs/Marker	std_msgs/Header header string ns int32 id int32 type int32 action geometry_msgs/Pose pose geometry_msgs/Vector3 scale std_msgs/ColorRGBA color duration lifetime bool frame_locked geometry_msgs/Point[] points std_msgs/ColorRGBA[] colors string text string mesh_resource bool mesh_use_embedded_materials	The Markers display allows programmatic addition of various primitive shapes to the 3D view by sending a visualization_msgs/Marker or visualization_msgs/MarkerArray message.	
Visualization_msgs/MarkerArray	visualization_msgs/Marker[] markers	For array of the Marker type.	

# ROS Messages

## ❖ sensor\_msgs

- ❑ sensor\_msgs is a set of messages made for delivering various input information, ranging from camera, lidar, imu and even joystick.

Message	Basic Information	Description	rviz
sensor_msgs/CompressedImage	std_msgs/Header header string format uint8[] data	Header timestamp should be acquisition time of image Header frame_id should be optical frame of camera origin of frame should be optical center of camera +x should point to the right in the image +y should point down in the image +z should point into to plane of the image	
sensor_msgs/LaserScan	std_msgs/Header header float32 angle_min float32 angle_max float32 angle_increment float32 time_increment float32 scan_time float32 range_min float32 range_max float32[] ranges float32[] intensities	Single scan from a planar laser range-finder	

---

---

# Make Custom Messages



# ROS Messages (Custom)

## ❖ Building .msg Files

Reference : <http://wiki.ros.org/msg>

- ❑ The ROS Client Libraries implement message generators that translate .msg files into source code.
- ❑ These message generators must be invoked from your build script(package.xml, CMakeList.txt)

Open package.xml, and make sure these two lines are in it:

```
<build_depend>message_generation</build_depend>
<run_depend>message_runtime</run_depend>
```

Description in the wiki is  
old version

Note that at build time, we need "message\_generation", while at runtime, we only need "message\_runtime".

Open CMakeLists.txt in your favorite text editor ([rosed](#) from the previous tutorial is a good option).

Add the message\_generation dependency to the find package call which already exists in your CMakeLists.txt so that you can generate messages. You can do this by simply adding message\_generation to the list of COMPONENTS such that it looks like this:

```
# Do not just add this line to your CMakeLists.txt, modify the existing line
find_package(catkin REQUIRED COMPONENTS roscpp rospy std_msgs message_generation)
```

You may notice sometimes your project builds fine even if you did not call find\_package with all dependencies. This is because catkin combines all your projects into one, so if an earlier project calls find\_package, yours is configured with the same values. But forgetting the call means your project can easily break when build in isolation.

Also make sure you export the message runtime dependency.

```
catkin_package(
...
CATKIN_DEPENDS message_runtime ...
...)
```

➤ Open package.xml and add these lines :

```
<build_depend>message_generation</build_depend>
<build_export_depend>message_runtime</build_export_depend>
<exec_depend>message_runtime</exec_depend>
```

➤ Open CMakeLists.txt and add  
“message\_generation”  
in the list of find\_package()

➤ Open CMakeLists.txt and add  
“message\_runtime”  
in the list of catkin\_package()

# ROS Messages (Custom)

## ❖ Building .msg Files (Cont')

Find the following block of code:

```
# add_message_files()  
# FILES  
# Message1.msg  
# Message2.msg  
# )
```

Uncomment it by removing the # symbols and then replace the stand in Message\*.msg files with your .msg file, such that it looks like this:

```
add_message_files()  
FILES  
Num.msg  
)
```

Find the following block of code:

```
# generate_messages()  
# DEPENDENCIES  
# std_msgs # Or other packages containing msgs  
# )
```

Uncomment it by removing the # symbols and then replace std\_msgs with the messages your messages depend on, such that it looks like this:

```
generate_messages()  
DEPENDENCIES  
std_msgs  
)
```

Reference : <http://wiki.ros.org/msg>

### ➤ Write your custom messages

Message1.msg

Message2.msg

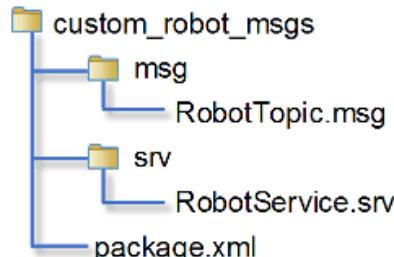
which are stored in **msg** directory.

### ➤ uncomment

add\_message\_files()  
by removing #

### ➤ uncomment

generate\_messages()  
by removing #

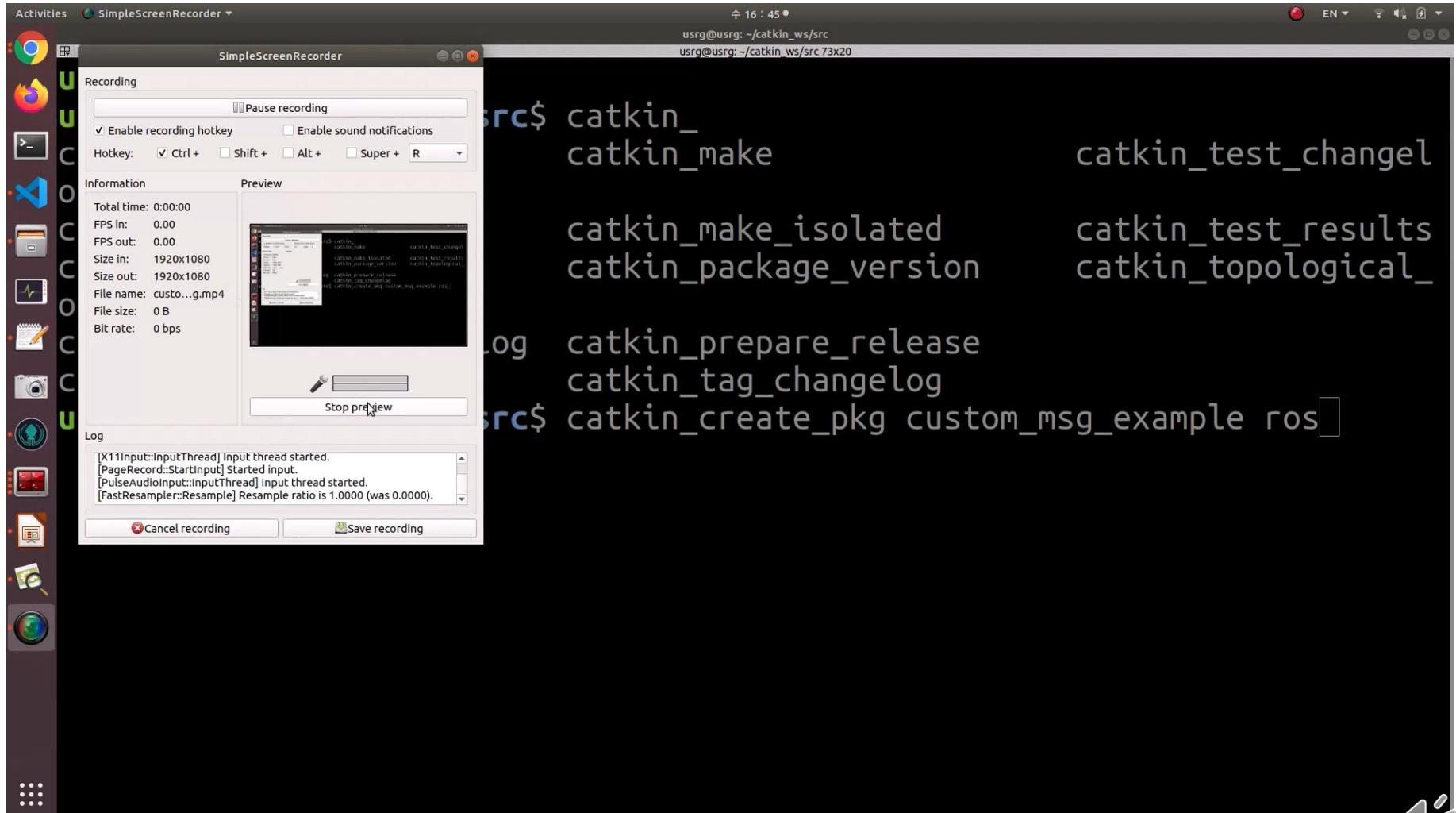


### ➤ Add your custom messages

Message1.msg --> here, RobotTopic.msg  
in **msg** directory.

# ROS Messages (Custom)

## ❖ Building .msg Files (Cont')



---

---

# Q & A