Lab0 – Linux & ROS Intro

View & Edit Lab Assignments - Latex

1. Go to the website : https://www.overleaf.com/
2. Create an account
3. Start a new project
4. Upload the .zip file (download from canvas) to the new project.

Intro to Linux for Robotics

What is Linux

- **Just like Windows, iOS, and Mac OS, Linux is an operating system.**
- An operating system is software that manages all of the hardware resources associated with your desktop or laptop.
- the operating system manages the communication between your software and your hardware
- one of the most popular platforms on the planet, Android, is powered by the Linux operating system

https://www.linux.com/what-is-linux/

Why use Linux?

As we know Windows and MacOS have more fancy GUIs (Graphical User Interface), why Linux is still in use?

- Linux is free and open source.
- Linux is generally far less vulnerable to viruses because system-related files are owned by the “root” superuser.
- **Ubuntu** has been the primary platform for **ROS** from the very beginning, thus ROS is most mature in Ubuntu, which is the reason why we need Ubuntu.
- Shell vs GUIs.

https://www.linux.com/what-is-linux/

What is a “distribution” for Linux

Linux has a number of different versions. These versions are called distributions (or, in the short form, “distros”).

Popular Linux distributions include:

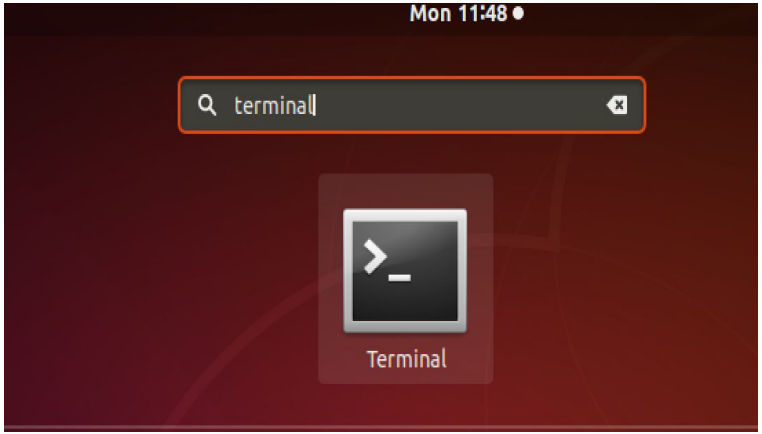
- LINUX MINT
- MANJARO
- DEBIAN
- **UBUNTU**
- ANTERGOS
- SOLUS
- FEDORA
- ELEMENTARY OS
- OPENSUSE

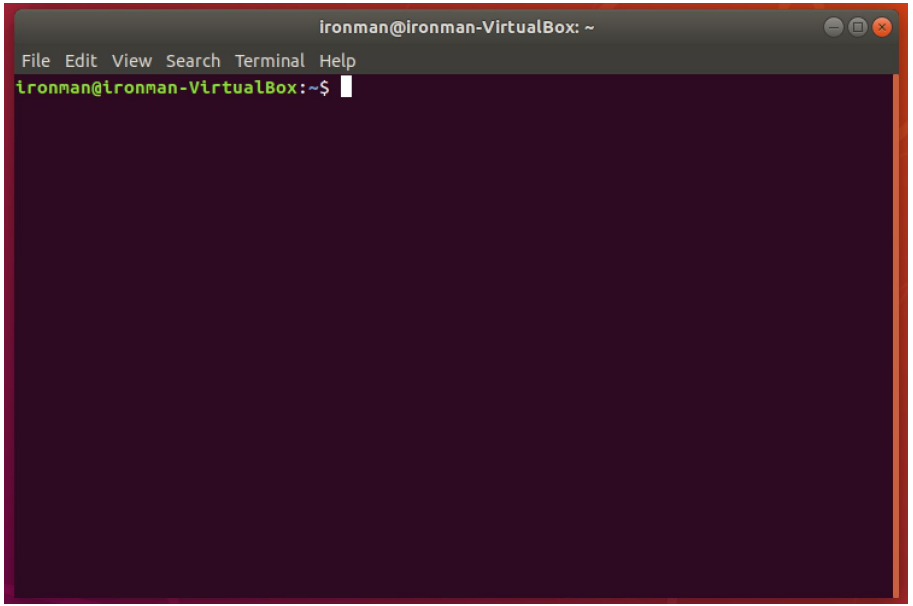
https://www.linux.com/what-is-linux/

Linux and Unix

1. Unix
    a. First developed in 1969 at Bell Labs by Dennis Ritchie and Ken Thompson (developed C and Go)
    b. Many key ideas still used today
        i. “Everything is a file”
        ii. Multiple users, hierarchical file system
        iii. Documentation included (built-in documentation)
    c. macOS is a unix based operating system (built on).
2. Linux
    a. Developed in 1992 by Linus Torvalds, who also developed git!
    b. OS family: Unix-like (derive from)
    c. Default user interface: Unix shell
    d. Linux kernel also derives a lot from Unix kernel.

Opening a Terminal





The Shell

- Shell: an interactive program that allows the user to interact with the operating system and its applications

- Why use a shell vs. the GUI (Graphical User Interface)?

    - Many complicated tasks are easier to do on the command line (navigate through the file system)
    - Useful for working on remote servers (ssh)
    - Programmable and Customizable (Bash scripts)

Some other nomenclatures

- The **terminal** is the GUI window that you see on the screen. It takes commands and shows output.

- The **shell** is the software that interprets and executes the various commands that we type in the terminal.

- **Bash** is a particular shell. It stands for **Bourne Again Shell.** Some other examples of shell are **sh(bourne shell), csh(c shell), tcsh(turbo c shell), dash (Debian Almquist Shell)** etc. (You will see **setup.bash** and **.bashrc** a lot)

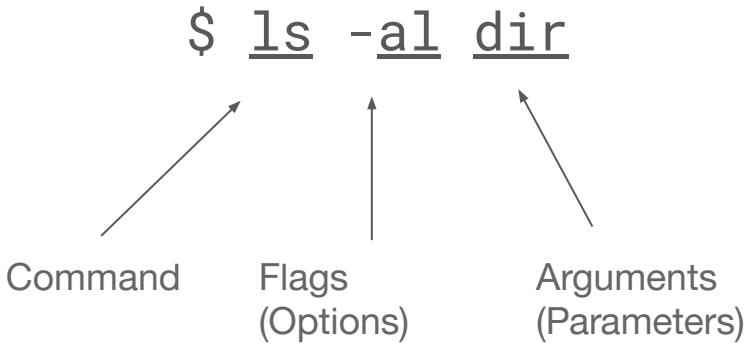
- **CLI(Command Line Interface)** refers to a **user interface** in computers where users type in commands and see the results printed on the screen

Basic Shell Commands

command	description
`pwd`	**P**rint current **w**orking **d**irectory
`cd`	**C**hange working **d**irectory
`ls`	List files in working directory
`man`	Bring up manual for a command
`exit`	Log out of shell
`mkdir`	Make a new directory
`rmdir`	Remove the given directory (must be empty)

Command Line Arguments

- There aren't any consistent definitions when it comes to command line arguments, but one way to use is the following to describe the anatomy of a command



Command Line Arguments

- Much like methods in Java take arguments, so do commands on the command line
- Flags are modifiers which change a programs behavior slightly, and they are usually prepended with a -
- For example, to list all files in **long-list format,** run the following
  - `$ ls -l`
- Flags can be combined, to list all files in long-list format and list hidden files
  - `$ ls -la`
- Commands also take arguments, such as file names
- To view all files , in long-listing format, inside of `dir`
  - `$ ls -l dir`
- To check the usage and options of a command
  - `$[command] --help`

Understanding files and permissions

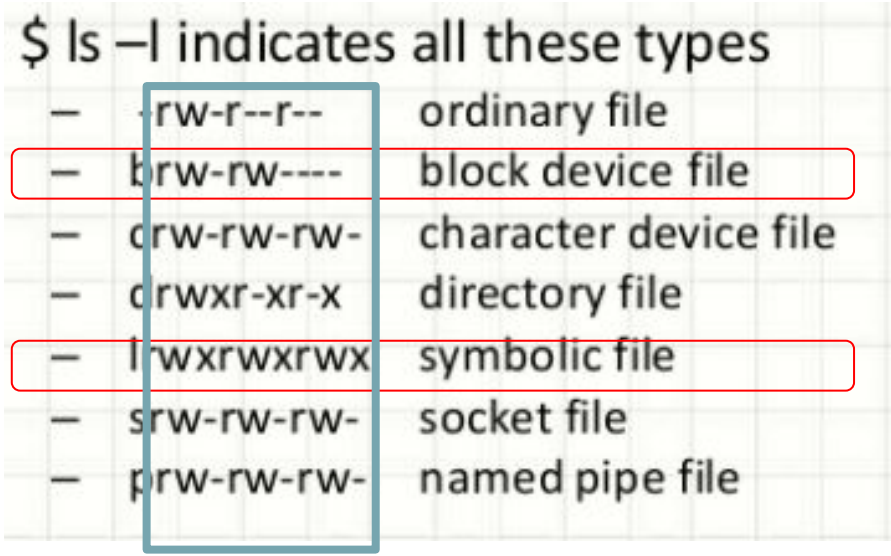
Explore file permissions in this directory

> ls

> ls -l

Change file permissions

> chmod



owner:group:other

More Shell Commands

directory	description
`cp`	Copy a file
`mv`	Move a file (also used to rename files)
`rm`	Remove the given file
`touch`	Create empty file, or change time-modified

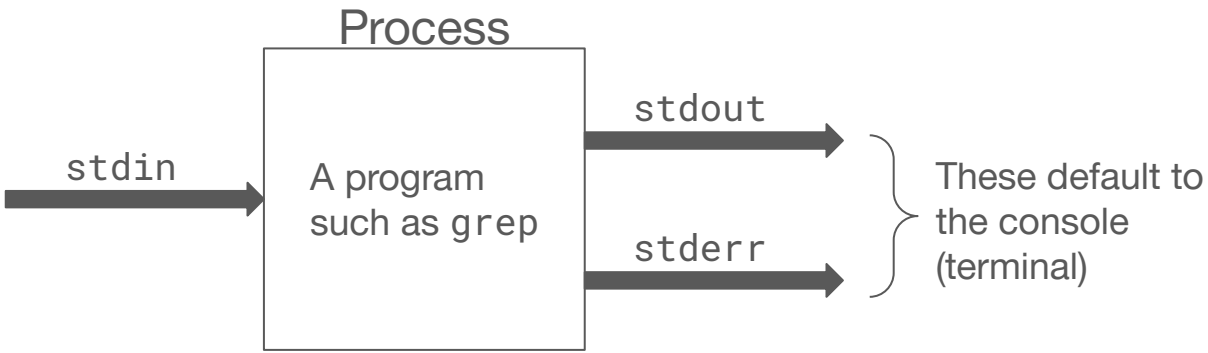
File Examination

Command	description
`cat`	Print contents of a file
`less`	Output file contents, one page
`more`	Output file contents, one page
`head`	Output number of lines of start of file
`tail`	Output number of lines of end of ile
`wc`	Count words, characters, lines in a file

Search and sorting

Command	description
`grep`	Search given file for pattern
`sort`	Sort input or file, line based
`uniq`	Strip duplicate **adjacent lines**
`find`	Search filesystem
`cut`	Remove section from each line of file

Standard streams



Note: **not** every command has the stdin or stdout.
Don't worry about stderr for now.

Stdin vs arguments(parameters)

- A *parameter* is an argument you give on the command line, like so
  - `$ ls dir1`
  - `dir1` is a parameter, it does not come from standard input
- Standard input comes from the user, either from a file or from the console
  - `$ grep “a”`
  - Once you type this command, it accepts input from your keyboard until you close the stream using Ctrl + C

If you cannot tell the difference in practice, then use “man” to look up the command.

Output redirection

`command > filename`

- Execute **`command`** and redirect its standard output to the given **`filename`**
  - If the file *does not* exist, create the given file.
  - If the file *does* exist, it will **overwrite the given file (BE CAREFUL!!)**
  - To append to a file instead of overwrite it, use >> instead of >
- Examples:
  - Output contents of current directory to files.txt: `ls -l > files.txt`
  - Append output of `wc -l CMakeLists.txt` to files.txt: `wc -l CMakeLists.txt >> files.txt`

Input redirection

```
command < filename
```

- Execute **`command`** and read its standard input from the contents of **`filename`** instead of from the console.
  - If a program usually accepts from user input, such as a console Scanner in Java, it will instead read from the file.
- Notice that this affects user input, not parameters.

Pipes

`command1 | command2`

- Execute **`command1`** and send its standard output as standard input to **`command2`**.
- This is essentially shorthand for the following sequence of commands:

```
command1 > filename
command2 < filename
rm filename
```

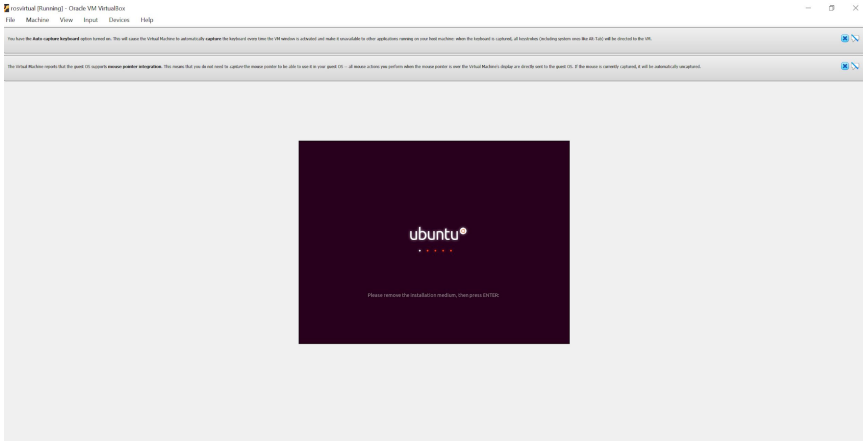
- This is one of the most powerful aspects of unix - being able to chain together simple commands to achieve complex behavior!

Text editor - Vim

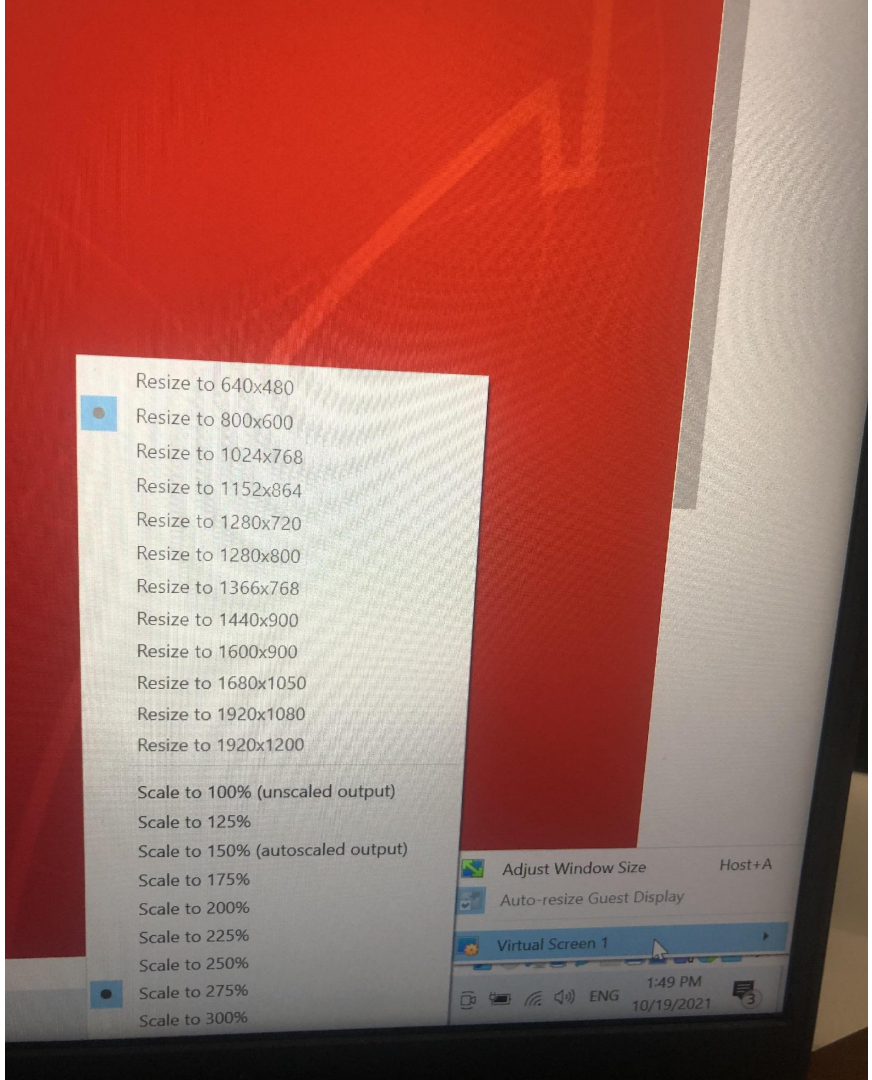
Key stroke	description
`:w`	Write (save) the current file
`:wq`	Write (save) the current file and exit
`:q!`	Quit, ignoring all changes
`i`	Go into insert mode
`Esc`	Go back to normal mode
`hjkl`	Move cursor left, down, up, right
`u`	Undo last change
`x`	Delete character

Other tips

1. How to make Ubuntu full screen

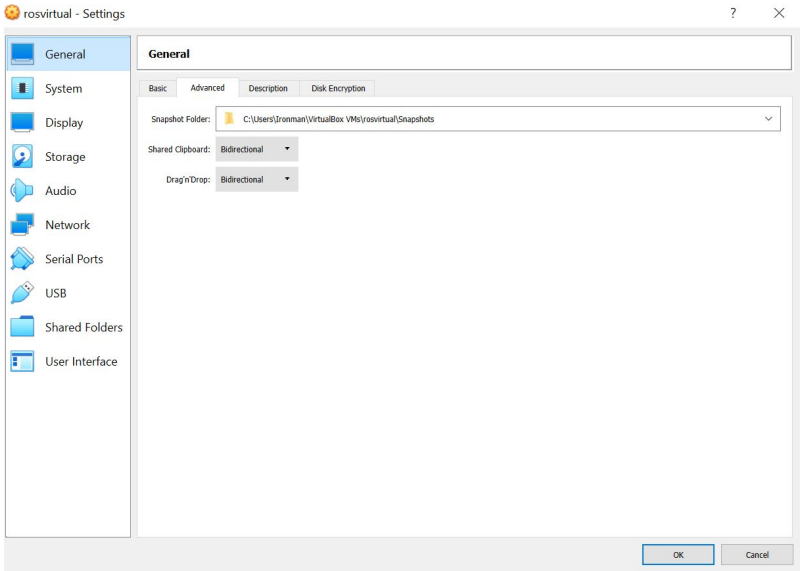


Then click view -> Full-screen Mode



Other tips

2. How to copy words in the windows machine and paste in the virtual machine. A good video is here (Except we install the guest addition in Ubuntu)

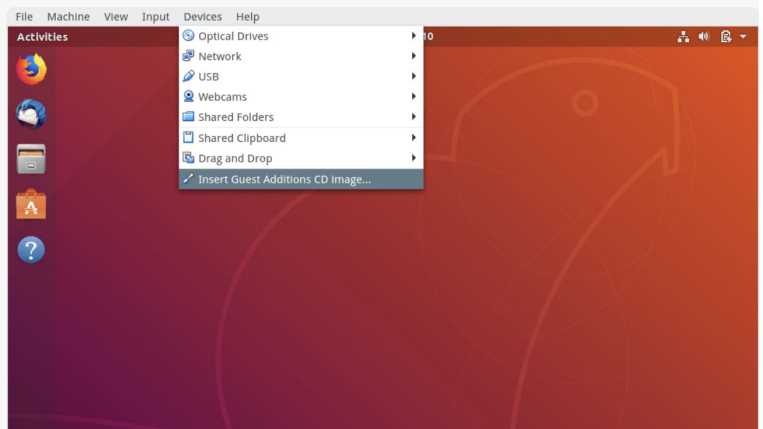


This is not enough.

Other tips

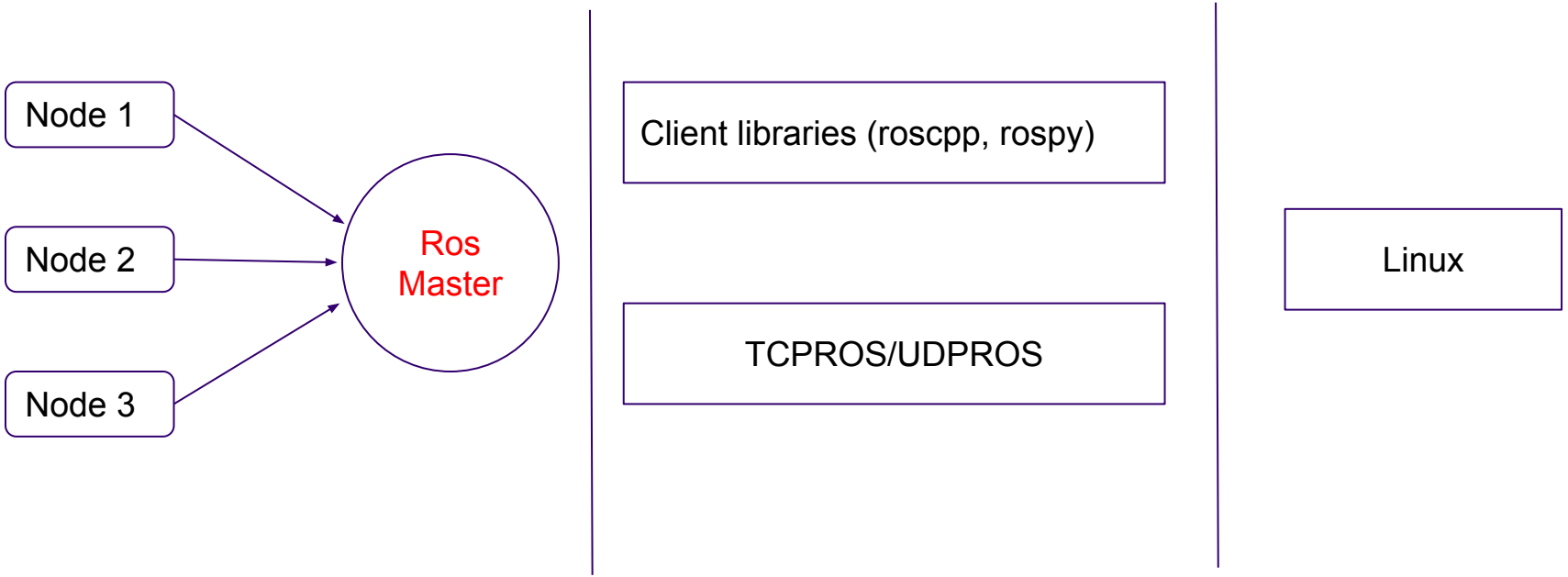
1. In Ubuntu open a terminal and type **sudo apt-get upgrade;**

1. Next, type **sudo apt-get install build-essential**
2. Click “Devices” and click “Insert Guest Additions CD image”

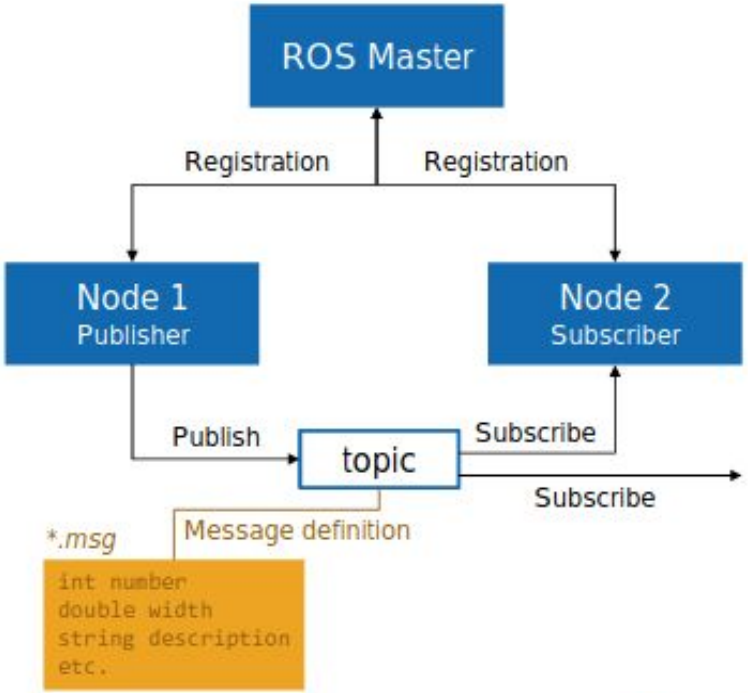


Brief Intro to ROS

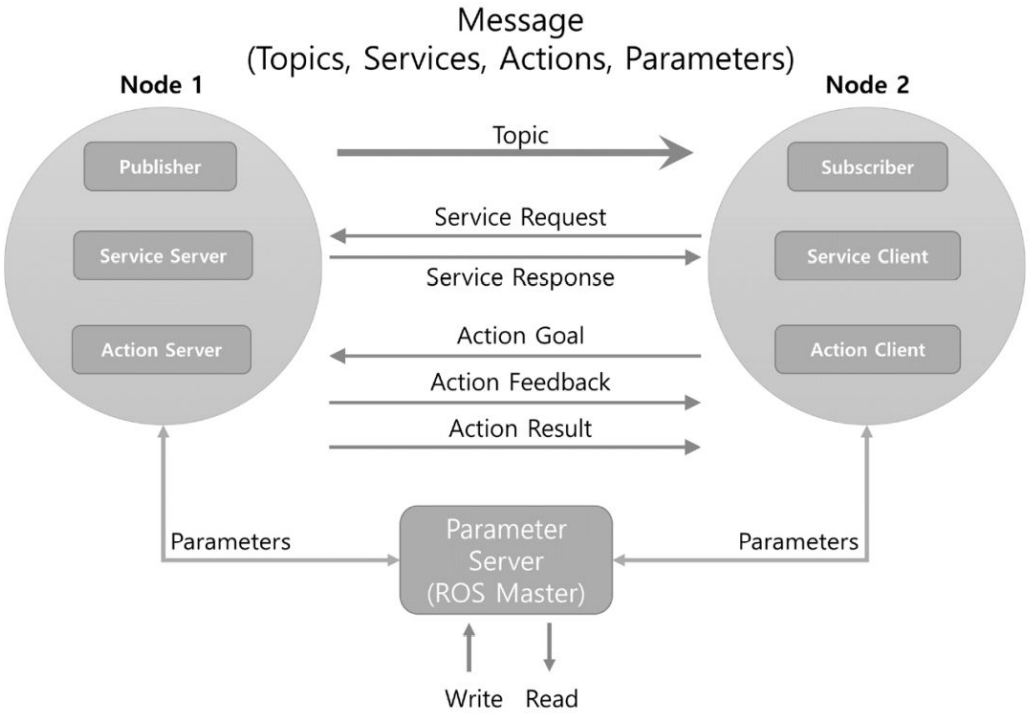
ROS overview



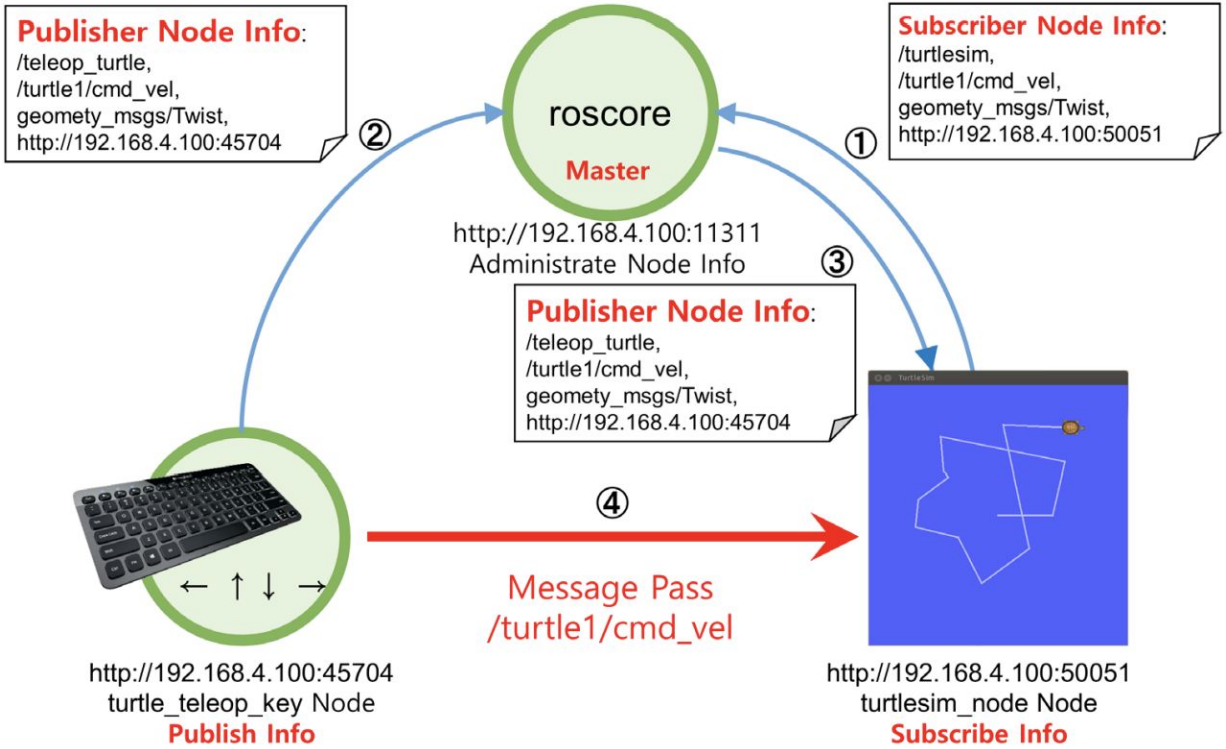
ROS architecture



ROS architecture (more complicated version)



ROS Concepts (an example)



工欲善其事，必先利其器 —— 孔子

a workman must first sharpen his tools if he is to do his work well —— Confucius

Tools：

1. ROS (Robot operating system)
2. Linux
3. Programming language: Python/C++
4. An IDE (Integrated development environment) , for example:

   Visual studio code

1. Latex: for example, overleaf

ROS Installation

ROS Melodic Morenia installation

1. Install the Ubuntu 18.04 (Bionic) release (Prepared in the USB, the link is here)

2. Configure your Ubuntu repositories to allow "restricted," "universe," and "multiverse". (A good link is here)

   - **sudo add-apt-repository universe**
   - **sudo add-apt-repository restricted**
   - **sudo add-apt-repository multiverse**

3. 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/ros-latest.list'**

ROS Melodic Morenia installation

4. Set up your keys

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

About what the apt-key is, refer to the link https://difyel.com/linux/usr/bin/apt-key/

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

- **sudo apt update**
- **sudo apt install ros-melodic-desktop-full**

ROS Melodic Morenia installation

6. 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**

.bashrc

.bashrc is a Bash shell script that Bash runs whenever it is started interactively. It initializes an interactive shell session. You can put any command in that file that you could type at terminal.

For example:

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

ROS Melodic Morenia installation

7. 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.

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

ROS Melodic Morenia installation

8. 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

- **sudo rosdep init**
- **rosdep update**

Configuring Your ROS Environment

1. If you are ever having problems finding or using your ROS packages make sure that you have your environment properly setup. A good way to check is to ensure that environment variables like ROS_ROOT and ROS_PACKAGE_PATH are set

   - **printenv | grep ROS**

Remember what ‘grep’ is? What ‘|’ is?

Configuring Your ROS Environment

2. Let's create and build a catkin workspace:

- **mkdir -p ~/catkin_ws/src**
- **cd ~/catkin_ws/**
- **catkin_make**

Note: catkin_make should be under directory '**~/catkin_ws/**'

You will see directories 'build', 'devel' and 'src' under directory '**~/catkin_ws/**'

Configuring Your ROS Environment

2. Python 3

- **mkdir -p ~/catkin_ws/src**
- **cd ~/catkin_ws/**
- **catkin_make -DPYTHON_EXECUTABLE=/usr/bin/python3**

You will get ‘ImportError: "from catkin_pkg.package import parse_package" failed: No module named 'catkin_pkg”

Note: If you just use ‘catkin_make’, then it is python2 by default. And You only need to do ‘-DPYTHON_EXECUTABLE=/usr/bin/python3’ stuff for the first time. In future, you can use just ‘catkin_make’.

Configuring Your ROS Environment

- **sudo apt install python3-pip**

- **pip3 install catkin_pkg**

Note: catkin_make should be under directory '**~/catkin_ws/**'

You will see directories 'build', 'devel' and 'src' under directory '**~/catkin_ws/**'

- **source devel/setup.bash**

Note: You might need to do this when you do catkin_make

Install python3 binding

- sudo -H pip3 install rosdep rospkg rosinstall_generator rosinstall wstool vcstools catkin_tools catkin_pkg

Configuring Your ROS Environment

3. 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**

It should output

/home/ironman/catkin_ws/src:/opt/ros/melodic/share

Detailed Intro to ROS

ROS file system

1. Install the ros-tutorials:

   - **sudo apt-get install ros-<distro>-ros-tutorials**

2. Try rospack, roscd and rosls according to the link below:

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

ROS package

1. A ROS project is usually wrapped as a ROS package

2. We will learn how to build a ROS package in the future, for now you only need to know:

Each package must have its own folder within the workspace. A catkin package must contain the following files:

1. A package.xml file: meta information.

2. A CMakeLists.txt file: dependencies and configurations for catkin.

ROS package

1. ROS packages are the way software is organized in ROS. They are the smallest thing you can build in ROS.

2. A package is a directory that contains all of the files, programs, libraries, and datasets needed to provide some useful functionality. ROS packages promote software reuse. **Every program that you write in ROS will need to be inside a package.**

3. The goal of a ROS package is to be large enough to be useful but not so large and complicated that nobody wants to reuse it for their own project.

ROS package

ROS catkin packages are organized as follows:

- launch folder: Contains launch files
- src folder: Contains the source code (C++, Python)
- CMakeLists.txt: List of cmake rules for compilation (Must)
- package.xml: Package information and dependencies (Must)

ROS package (Example: Turtlebot3)

1. install the TurtleBot3 simulator:

   - **cd ~/catkin_ws/src/**
   - **git clone https://github.com/ROBOTIS-GIT/turtlebot3_msgs.git**
   - **git clone https://github.com/ROBOTIS-GIT/turtlebot3.git**
   - **cd ~/catkin_ws && catkin_make**

ROS package (Example: Turtlebot3)

2. TurtleBot3 has three models, Burger, Waffle, and Waffle Pi, so you have to set which model you want to use:

- **echo "export TURTLEBOT3_MODEL=burger" >> ~/.bashrc**
- **source ~/.bashrc**
- **cd ~/catkin_ws/src/**
- **git clone https://github.com/ROBOTIS-GIT/turtlebot3_simulations.git**
- **cd ~/catkin_ws && catkin_make**

ROS package (Example: Turtlebot3)

3. If you find an error:

```
Traceback (most recent call last):
  File "/opt/ros/melodic/share/gencpp/cmake/../../../lib/gencpp/gen_cpp.py", line 43, in <module>
    import genmsg.template_tools
  File "/opt/ros/melodic/lib/python2.7/dist-packages/genmsg/template_tools.py", line 39, in <module>
    import em
ModuleNotFoundError: No module named 'em'
```

Then type:

- **pip3 install empy**

ROS package (Example: Turtlebot3)

Then we can use rospack to find a package:

- **rospack find turtlebot3_teleop**

The output will be:

~/catkin_ws/src/turtlebot3/turtlebot3_teleop

Then we can go to see the package structure.

IDE: Visual studio code

Install the visual studio code in this website:

https://code.visualstudio.com/

ROS Graph concepts

- Nodes: A node is an executable that uses ROS to communicate with other nodes.
- Messages: ROS data type used when subscribing or publishing to a topic.
- Topics: Nodes can *publish* messages to a topic as well as *subscribe* to a topic to receive messages.
- Master: Name service for ROS (i.e. helps nodes find each other)
- rosout: ROS equivalent of stdout/stderr
- roscore: Master + rosout + parameter server (parameter server will be introduced later)

ROS node

- A node really isn't much more than an executable file within a ROS package. ROS nodes use a ROS client library to communicate with other nodes. Nodes can publish or subscribe to a Topic. Nodes can also provide or use a Service.
- ROS client libraries allow nodes written in different programming languages to communicate:

  rospy = python client library

  roscpp = c++ client library

Turtlesim

```
ironman@ironman-VirtualBox:~$ roscore
... logging to /home/ironman/.ros/log/d5833352-6eab-11ec-8557-080027da3a06/rosla
unch-ironman-VirtualBox-2571.log
Checking log directory for disk usage. This may take a while.
Press Ctrl-C to interrupt
Done checking log file disk usage. Usage is <1GB.

started roslaunch server http://ironman-VirtualBox:46427/
ros_comm version 1.14.12


SUMMARY
========

PARAMETERS
 * /rosdistro: melodic
 * /rosversion: 1.14.12

NODES

auto-starting new master
process[master]: started with pid [2582]
ROS_MASTER_URI=http://ironman-VirtualBox:11311/
```

Turtlesim

Open a new terminal

```
ironman@ironman-VirtualBox:~$ rosnode list
/rosout
ironman@ironman-VirtualBox:~$ rosnode info /rosout
--------------------------------------------------------------------------------
Node [/rosout]
Publications:
 * /rosout_agg [rosgraph_msgs/Log]

Subscriptions:
 * /rosout [unknown type]

Services:
 * /rosout/get_loggers
 * /rosout/set_logger_level


contacting node http://ironman-VirtualBox:42291/ ...
Pid: 2593
```

Turtlesim- rosrun

rosrun allows you to use the package name to directly run a node within a package (without having to know the package path). in the following format:

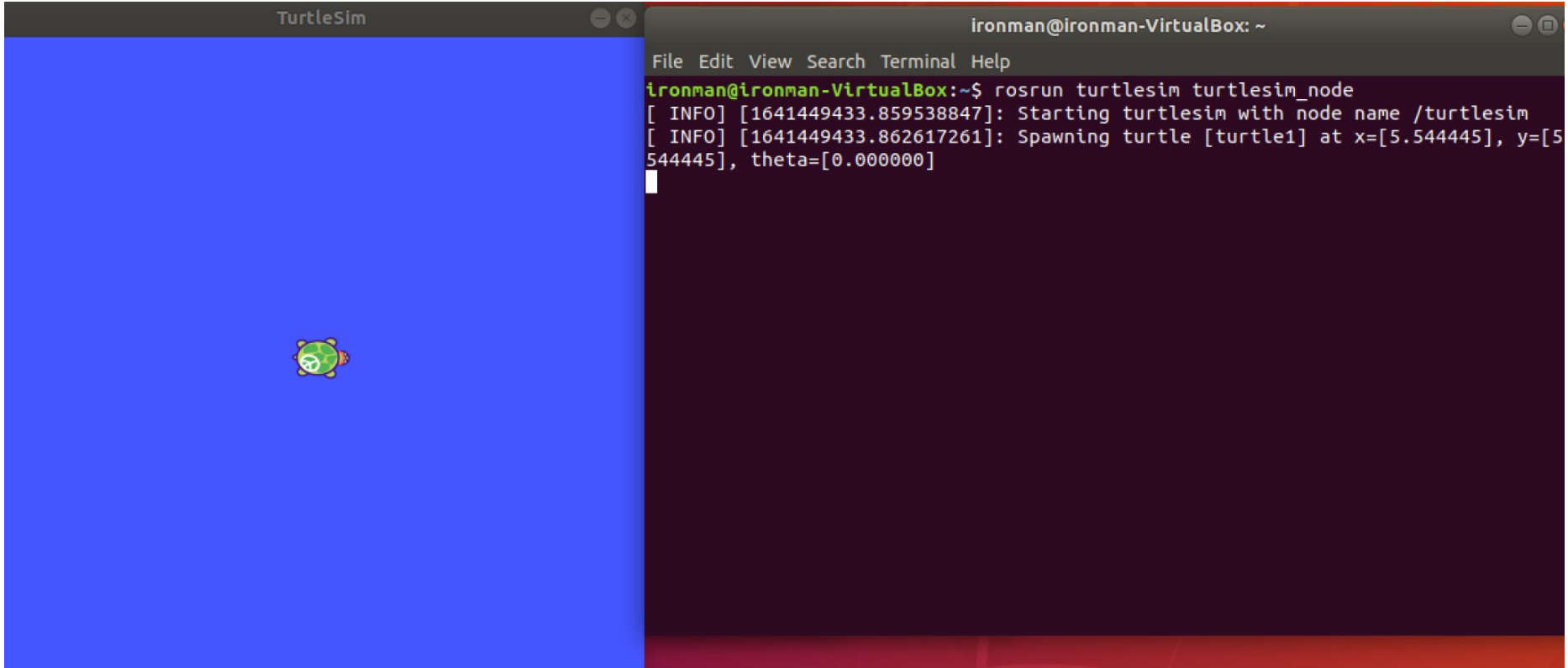
```
$ rosrun [package_name] [node_name]
```

Now let's rosrun a node in Turtlesim package:

```
$ rosrun turtlesim turtlesim_node
```

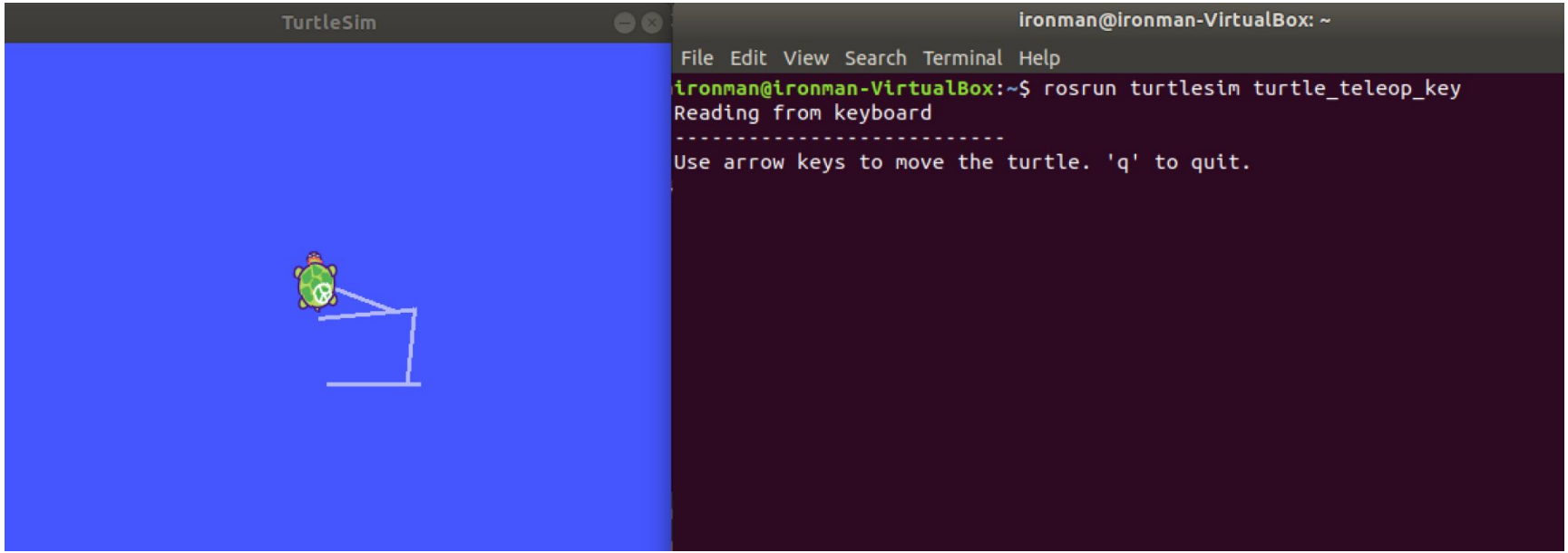
Turtlesim- rosrun

Open a new terminal



Turtlesim- rosrun

Open a new terminal



Note : select the terminal window of the turtle_teleop_key to make sure that the keys that you type are recorded.

Turtlesim- rosrun

You can see the running node now:

```
ironman@ironman-VirtualBox:~$ rosnode list
/rosout
/teleop_turtle
/turtlesim
```

ROS Concepts (call back)