Checkpoint #1 Raspberry Pi and ROS

[ICN5406] Mobile Robot 2021

Due: October 8, 2021

- ## Purpose：

The purpose of checkpoint 1 is to make sure you set the development environment for robot right.

First, you will need to produce the image for your raspberry pi and install ROS in it. Then use rosserial package from http://wiki.ros.org/rosserial_arduino to communicate between Raspberry pi and Arduino.

- ## Tasks：

Demonstrate your environment setup by performing the following two tasks:

1. Use ssh command to remote connect to Raspberry Pi from your PC and use rosversion command show that you already install ROS Melodic on Raspberry Pi. Such as Fig.1 show below. (30%)

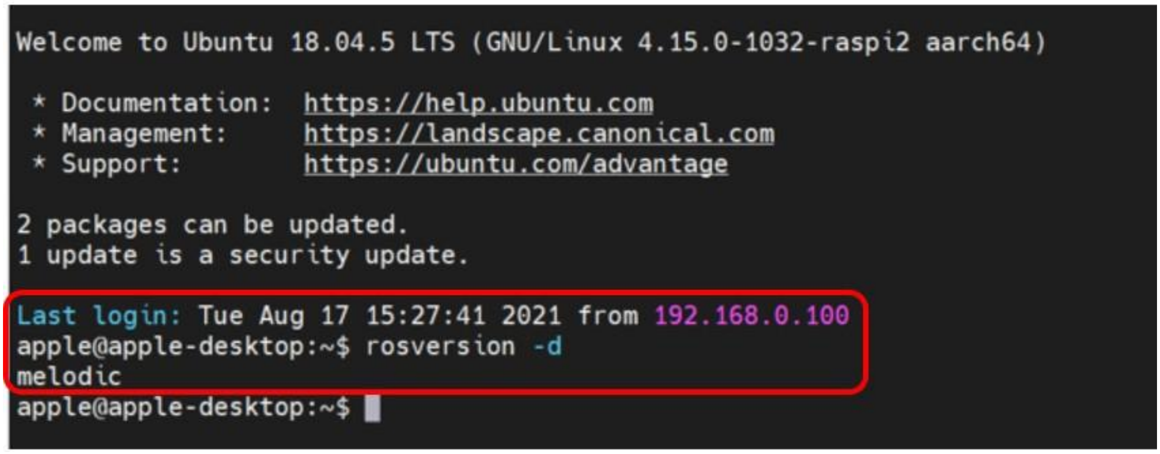
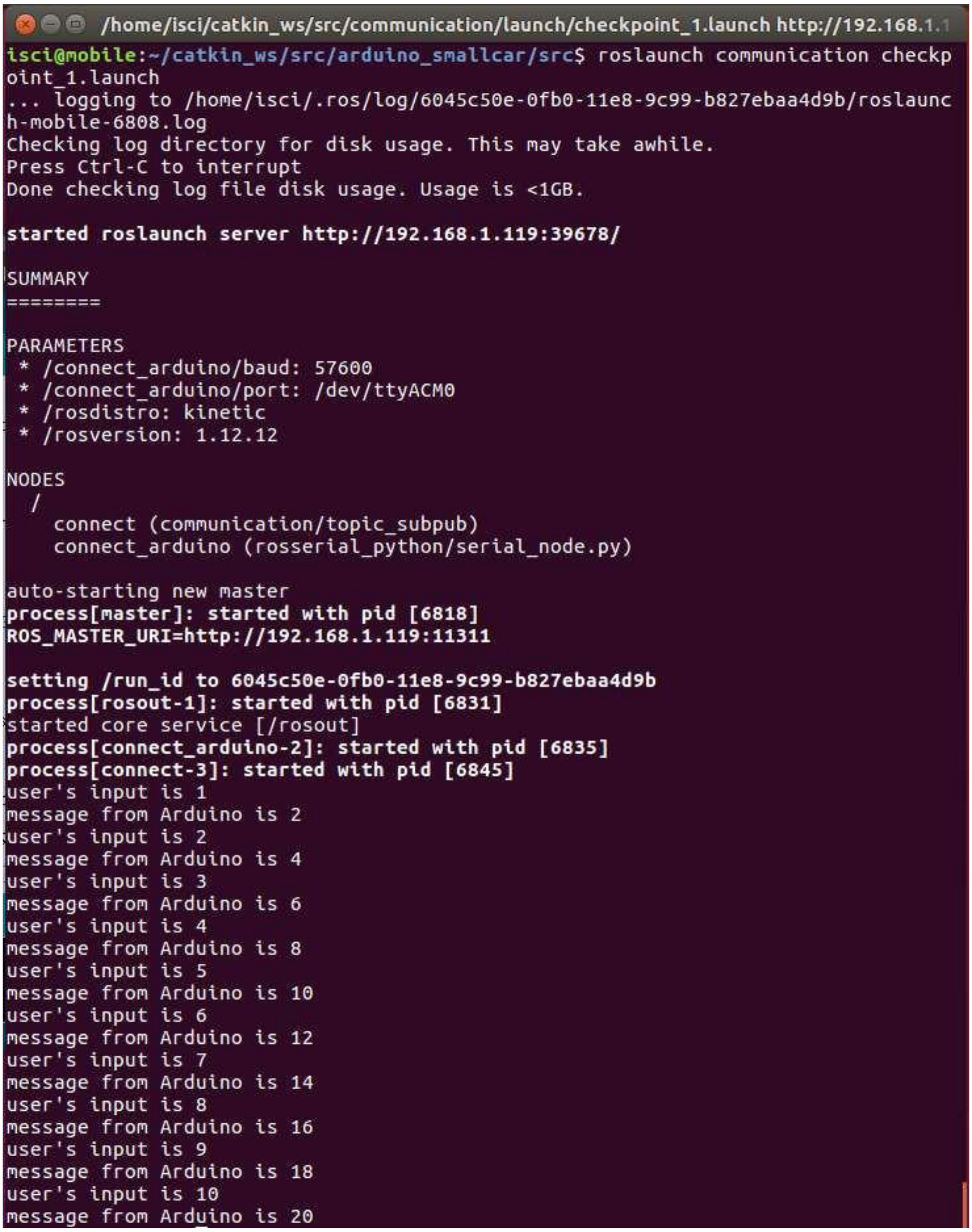


Fig.1 SSH remote connection and ROS version

2. Use command and library from rosserial package to create a program which contains publishers and subscribers. It means that your Raspberry Pi should send and receive messages to and from Arduino. In this program, Arduino should receive a number from Raspberry Pi, multiply it by 2, and then send it back to Raspberry Pi. You should show the result from Terminal. An example result show below like Fig.2. (70%)



```
/home/isci/catkin_ws/src/communication/launch/checkpoint_1.launch http://192.168.1.1
isci@mobile:~/catkin_ws/src/arduino_smallcar/src$ roslaunch communication checkp
oint_1.launch
... logging to /home/isci/.ros/log/6045c50e-0fb0-11e8-9c99-b827ebaa4d9b/roslaunc
h-mobile-6808.log
Checking log directory for disk usage. This may take awhile.
Press Ctrl-C to interrupt
Done checking log file disk usage. Usage is <1GB.

started roslaunch server http://192.168.1.119:39678/

SUMMARY
========

PARAMETERS
 * /connect_arduino/baud: 57600
 * /connect_arduino/port: /dev/ttyACM0
 * /rosdistro: kinetic
 * /rosversion: 1.12.12

NODES
  /
    connect (communication/topic_subpub)
    connect_arduino (rosserial_python/serial_node.py)

auto-starting new master
process[master]: started with pid [6818]
ROS_MASTER_URI=http://192.168.1.119:11311

setting /run_id to 6045c50e-0fb0-11e8-9c99-b827ebaa4d9b
process[rosout-1]: started with pid [6831]
started core service [/rosout]
process[connect_arduino-2]: started with pid [6835]
process[connect-3]: started with pid [6845]
user's input is 1
message from Arduino is 2
user's input is 2
message from Arduino is 4
user's input is 3
message from Arduino is 6
user's input is 4
message from Arduino is 8
user's input is 5
message from Arduino is 10
user's input is 6
message from Arduino is 12
user's input is 7
message from Arduino is 14
user's input is 8
message from Arduino is 16
user's input is 9
message from Arduino is 18
user's input is 10
message from Arduino is 20
```

Fig.2 Communication between RPi and Arduino

- Materials list:

	Material	Number
1	Raspberry Pi 3 Model B	1
2	16G micro SD card (with SD adapter)	1
3	USB Type A male – Micro B male	1
4	Arduino UNO	1
5	USB Type A male – Type B male	1
6	Power Bank	1

Raspberry Pi 3 Model B	SD card and adapter
	
Arduino UNO	**USB A-Micro B**
	
USB A-B	
	

- ## Tutorial

In this tutorial we will go through about the environment setting on Raspberry Pi, the architecture of ROS and how to connect Raspberry Pi and Arduino by using rosserial package.

A. Environment setting on Raspberry Pi
   1. Install Ubuntu mate 18.04
   2. Install ROS Melodic
   3. Setting SSH between Raspberry Pi and PC

B. ROS
   1. ROS file system level structure
   2. ROS computation graph level
   3. Create package file
   4. ROS nodes communication
   5. Publisher and Subscriber
   6. CMakeLists.txt
   7. roslaunch

C. rosserialpackage
   1. Arduino IDE setup
   2. Install the Software
   3. Create a publisher by using rosserial
   4. Create a subscriber by using rosserial

A. Environment Setting on Raspberry Pi

1. Install Ubuntu mate 18.04

(1-1) Download Ubuntu mate 18.04 (64-bit) from https://releases.ubuntu-mate.org/archived/18.04/

/archived/18.04/arm64/

File Name ↓	File Size ↓	Date ↓
../	-	-
ubuntu-mate-18.04.2-beta1-desktop-arm64+raspi3-ext4.img.xz	1.2 GiB	2019-Mar-24 23:17
ubuntu-mate-18.04.2-beta1-desktop-arm64+raspi3-ext4.img.xz.sha256	125 B	2019-Mar-24 23:17
ubuntu-mate-18.04.2-beta1-desktop-arm64+raspi3-ext4.img.xz.sha256.sign	488 B	2019-Mar-24 23:17
ubuntu-mate-18.04.2-beta1-desktop-arm64+raspi3-ext4.img.xz.torrent	93.9 KiB	2019-Mar-29 01:20

Fig.1 Ubuntu MATE 18.04 image file

(1-2) Then install Ubuntu mate image file in SD card. You can use any tool like GNOME Disk (like Fig.2 showed below) or other applications such as ddrescue on Linux or Win32 Disk Imager on Windows can be used.

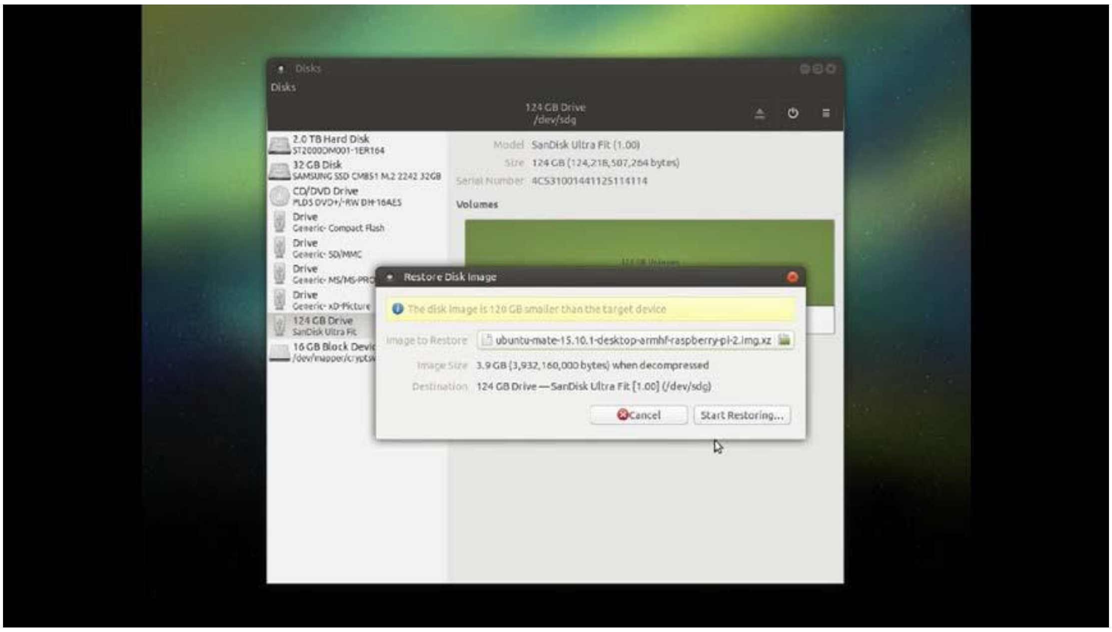


Fig.2 GNOME Disk restore image

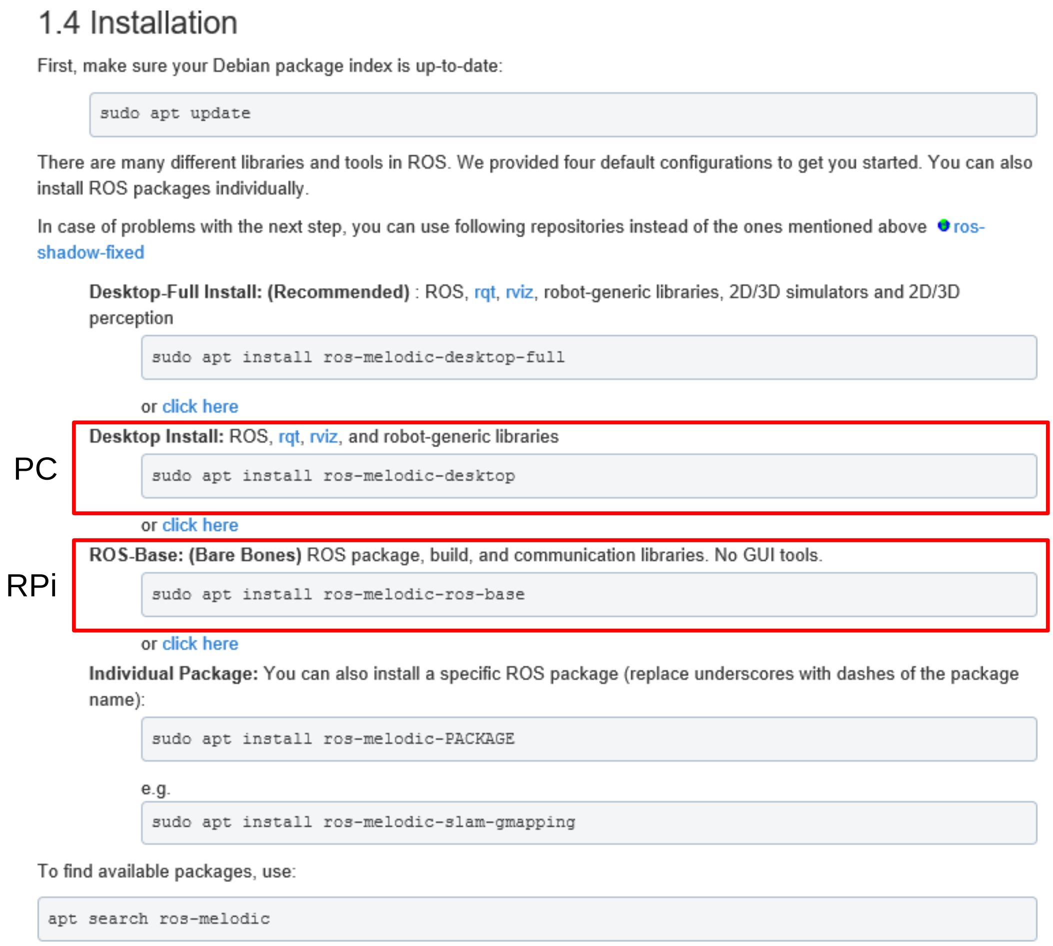
(1-3) Then plug SD card in raspberry pi and connect to screen and finish all the initial setting.

2. Install ROS Melodic

(2-1) You can start to install ROS from *http://wiki.ros.org/melodic/Installation/Ubuntu* "1.2 Setup your sources.list" and keep following the instruction until "1.7 Getting rosinstall".

(2-2) In "1.4 Installation" we recommend you install **ROS-Base** in Raspberry Pi and **Desktop Install** in PC. Show in Fig.5 below.

This instruction takes about 2 hours to install ROS. Elapsed time may vary depending on network environment. Please start your environment setting earlier.



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

PC

Desktop Install: ROS, rqt, rviz, and robot-generic libraries

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

or click here

RPi

ROS-Base: (Bare Bones) ROS package, build, and communication libraries. No GUI tools.

```
sudo apt install ros-melodic-ros-base
```

or click here

Individual Package: You can also install a specific ROS package (replace underscores with dashes of the package name):

```
sudo apt install ros-melodic-PACKAGE
```

e.g.

```
sudo apt install ros-melodic-slam-gmapping
```

To find available packages, use:

```
apt search ros-melodic
```

Fig.3 ROS installation for different processor

(2-3) Then follow the "3. Create a ROS Workspace" from http://wiki.ros.org/ROS/Tutorials/InstallingandConfiguringROSEnvironment to finish all the ROS setting.

3. Setting *ssh* between Raspberry Pi and PC

(3-1) Follow the instruction down below to install *ssh* in both your PC and Raspberry Pi.

```
$ sudo apt-get update
$ sudo apt-get install openssh-server openssh-client
$ sudo service ssh start
$ systemctl enable ssh.socket
$ sudo dpkg-reconfigure openssh-server
$ sudo service ssh restart
```

(3-2) Find Raspberry Pi's address, netmask and gateway by using

```
$ ifconfig
$ route -n
```

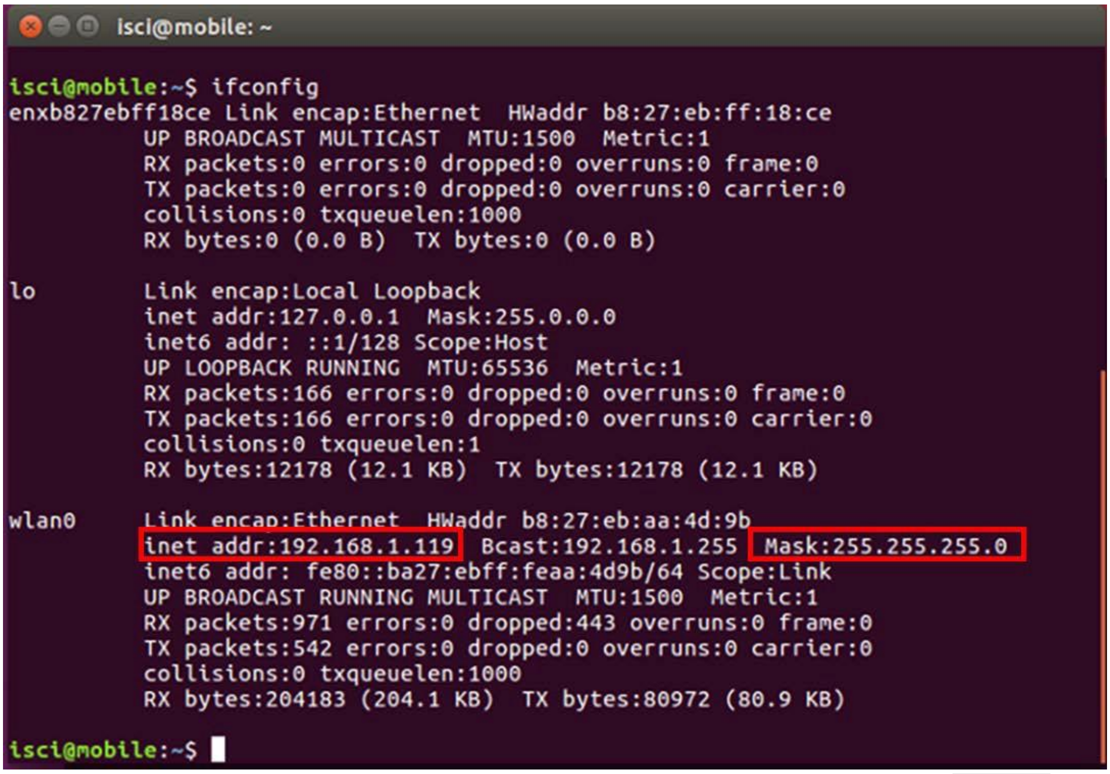


Fig.4 Raspberry Pi's network information by using *ifconfig*

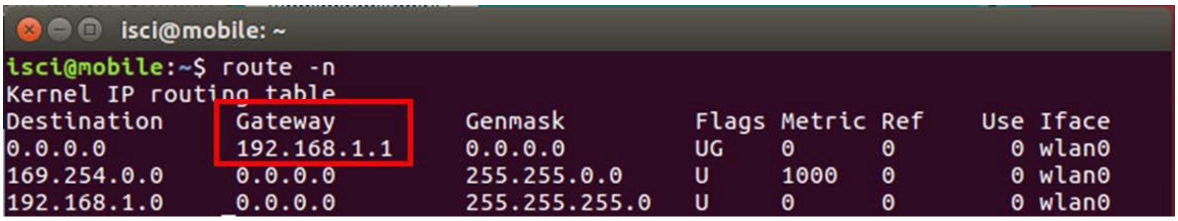


Fig.5 Raspberry Pi's network information by using *route –n*

(3-3) Setting static IP in Graphical User Interface.

You will need to set up address <rpi's inet addr>, netmask <rpi's Mask> and gateway <rpi's Gateway> in GUI.

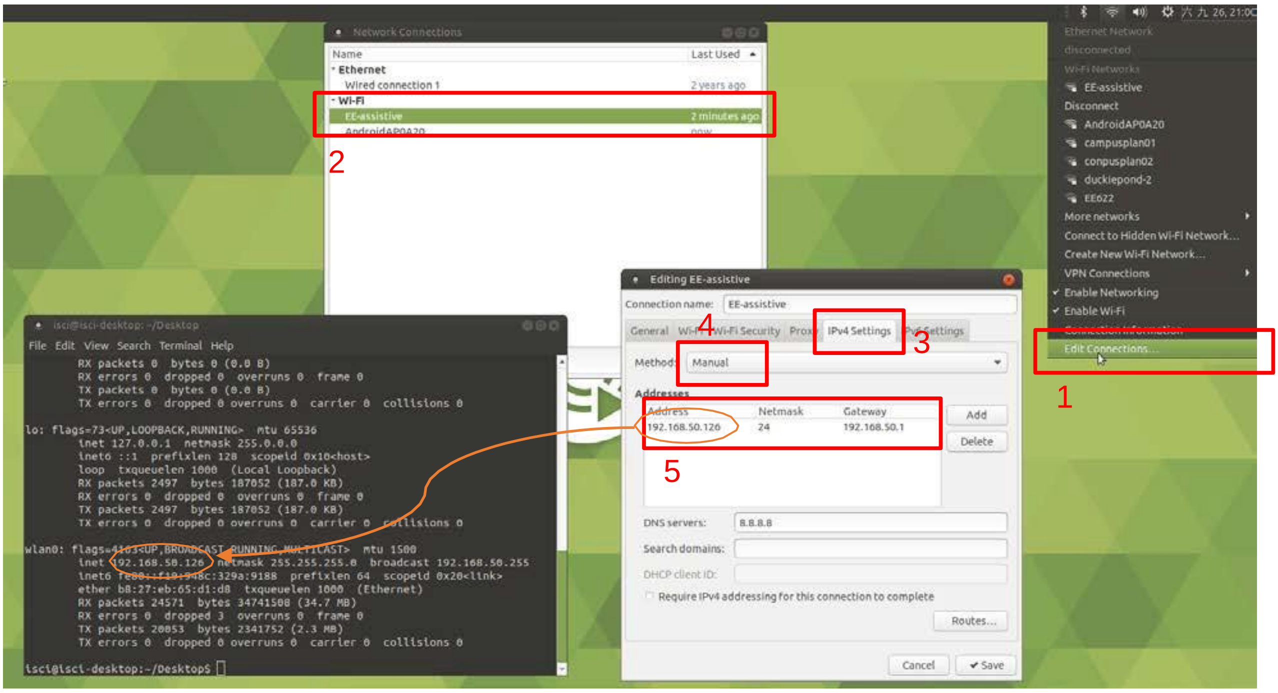


Fig.6 Steps for setting static IP

(3-4) Using *ssh* to remote connect with PC.

```
$ ssh <user_name>@<ip_addr>
```

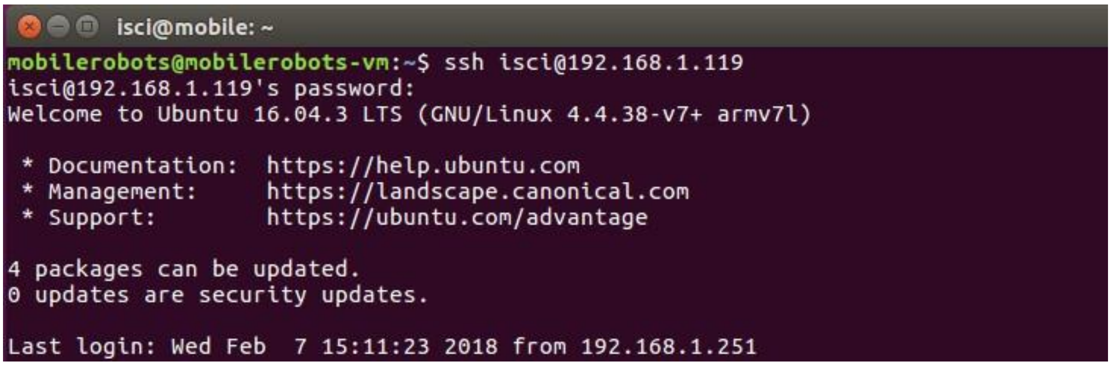


Fig.7 *ssh* command for remote connection

B. RC S

1. ROS file system level structure

Reference: http://wiki.ros.org/ROS/Concepts

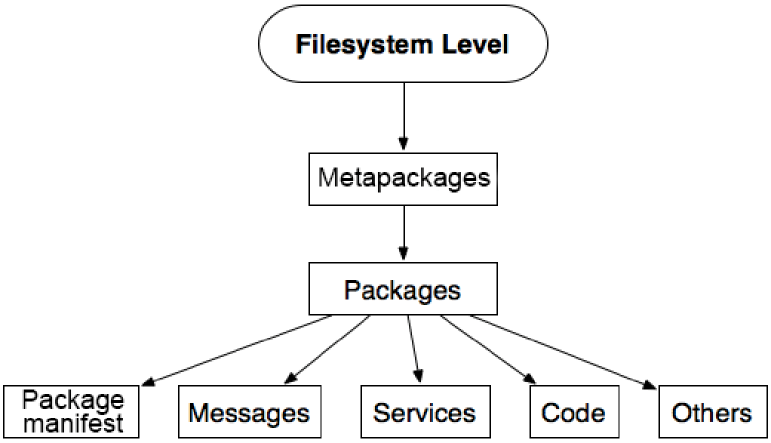


Fig.8 ROS file system level structure

(1-1) Packages:

Packages are the main unit for organizing software in ROS. A package may contain ROS runtime processes (nodes), a ROS-dependent library, datasets, configuration files, or anything else that is usefully organized together. Packages are the most atomic build item and release item in ROS. Meaning that the most granular thing you can build, and release is a package.

2. ROS computation graph level

Reference: http://wiki.ros.org/ROS/Concepts

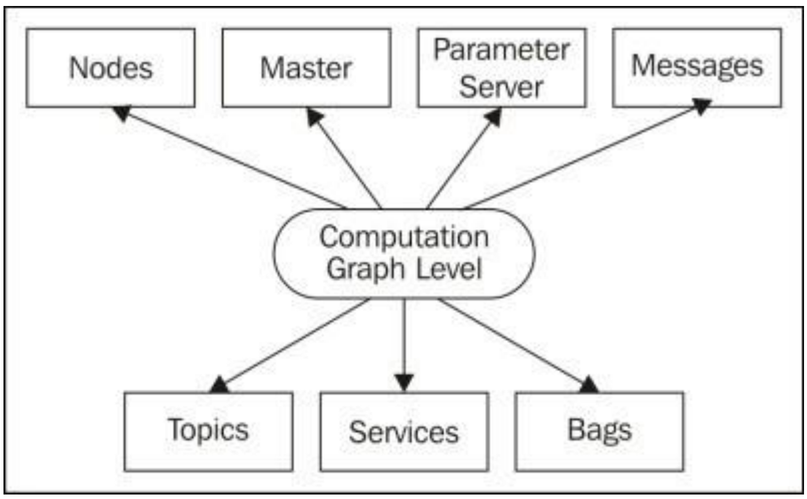


Fig.9 ROS computation graph level

(2-1) Nodes:

Nodes are processes that perform computation. ROS is designed to be modular at a fine-grained scale; a robot control system usually comprises many nodes. For example, one node controls a laser range-finder, one node controls the wheel motors, one node performs localization, one node performs path planning, one Node provides a graphical view of the system, and so on. A ROS node is written with the use of a ROS client library, such as *roscpp* or *rospy*.

(2-2) Master:

The ROS Master provides name registration and lookup to the rest of the Computation Graph. Without the Master, nodes would not be able to find each other, exchange messages, or invoke services.

(2-3) Topics:

Messages are routed via a transport system with publish / subscribe semantics. A node sends out a message by publishing it to a given topic. The topic is a name that is used to identify the content of the message. A node that is interested in a certain kind of data will subscribe to the appropriate topic. There may be multiple concurrent publishers and subscribers for a single topic, and a single node may publish and/or subscribe to multiple topics. In general, publishers and subscribers are not aware of each other's existence. The idea is to decouple the production of information from its consumption. Logically, one can think of a topic as a strongly typed message bus. Each bus has a name, and anyone can connect to the bus to send or receive messages as long as they are the right type.

(2-4) Messages:

Nodes communicate with each other by passing messages. A message is simply a data structure, comprising typed fields. Standard primitive types (integer, floating point, boolean, etc.) are supported, as are arrays of primitive types. Messages can include arbitrarily nested structures and arrays (much like C structs).

(2-5) Parameter Server:

The Parameter Server allows data to be stored by key in a central location. It is currently part of the Master.

3. Create package file

Create a ROS package by using *catkin_create_pkg* command.

```
$ cd catkin_ws/src
$ catkin_create_pkg [package_name] [dependency1] [dependency2]
```

Common dependency such as *roscpp, std_msgs, actionlib, actionlib_msgs*…etc.

4. ROS nodes communication

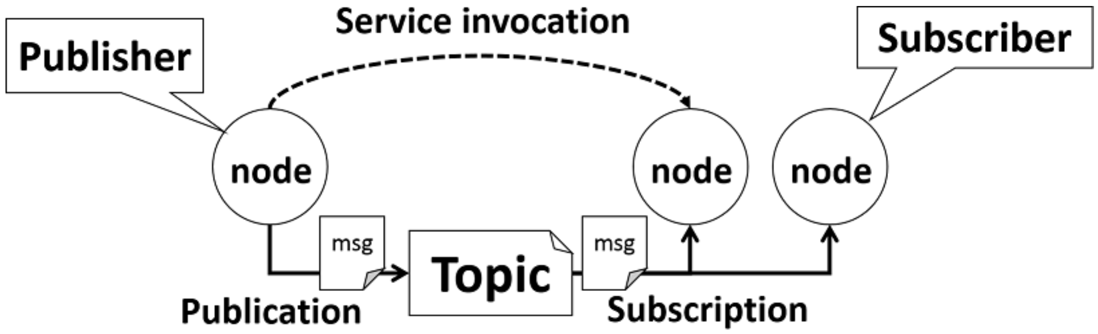


Fig.10 ROS node Communication

5. Publisher and Subscriber

Reference: *Mastering ROS for Robotics Programming, p.29-31*

(5-1) This example code will publish an integer value on a topic called */numbers*.

Demo_topic_publisher.cpp

```
#include “ros/ros.h”
#include “std_msgs/Int32.h”
#include <iostream>
int main(int argc, char **argv)
{
    ros::init(argc, argv, “demo_topic_publisher”);
    ros::NodeHandle node_obj;
    ros::Publisher number_publisher =
    node_obj.advertise<std_msgs::Int32>(“/numbers”, 10);
    ros::Rate loop_rate(10);
    int number_count = 0;
    while (ros::ok())
    {
        std_msgs::Int32 msg;
        msg.data = number_count;
        ROS_INFO(“%d”, msg.data);
        number_publisher.publish(msg);
        ros::spinOnce();
        loop_rate.sleep();
        ++number_count;
    }
    return 0;
```

```
}
```

Here is some detailed explanation of the example code:

```
ros::init(argc, argv, “demo_topic_publisher”);
```

This code will initialize a ROS node with a name. It should be noted that the ROS node should be unique. This line is mandatory for all ROS C++ nodes.

```
ros::NodeHandle node_obj;
```

This will create a *Nodehandle* object, which is used to communicate with the ROS system.

```
ros::Publisher number_publisher =
node_obj.advertise<std_msgs::Int32>(“/numbers”, 10);
```

This will create a topic publisher and name the topic */number* with a message type *std_msgs::Int32*.

The second argument is the buffer size. It indicates that how many messages need to be put in a buffer before sending.

```
ros::Rate loop_rate(10);
```

This is used to set the frequency of sending data.

```
ros::ok()
```

This function returns zero when there is an interrupt like *Ctrl+C*.

```
number_publisher.publish(msg);
```

This will publish the message to the topic */numbers*.

(5-2) This example code is the definition of the subscriber node:

Demo_topic_subscriber.cpp

```
#include “ros/ros.h”
#include “std_msgs/Int32.h”
#include <iostream>
void number_callback(const std_msgs::Int32::Constptr& msg)
{
    ROS_INFO(“Received [%d]”, msg->data);
}
int main(int argc, char **argv)
{
    ros::init(argc, argv, “demo_topic_publisher”);
    ros::NodeHandle node_obj;
```

```
    ros::Subscriber number_subscriber =
    node_obj.subscribe("/numbers", 10, number_callback);
    ros::spin();
    return 0;
}
```

Here is some detailed explanation of the example code:

```
void number_callback(const std_msgs::Int32::Constptr& msg)
{
    ROS_INFO("Received [%d]", msg->data);
}
```

This is a callback function that will execute whenever a data comes to the *numbers* topic. Whenever a data reaches this topic, the function will call and extract the value and print it on the console.

```
ros::Subscriber number_subscriber =
node_obj.subscribe("/numbers", 10, number_callback);
```

This is the subscriber and here, we are giving the topic name needed to subscribe, buffer size, and the callback function. We are subscribing */number* topic and we have already seen the callback function in the preceding section.

6. CMakeLists.txt:

Reference: http://wiki.ros.org/catkin/CMakeLists.txt

(6-1) The file **CMakeLists.txt** is the input to the CMake build system for building software packages. Any CMake-compliant package contains one or more CMakeLists.txt file that describe how to build the code and where to install it to. The CMakeLists.txt file used for *catkin_make* project.

(6-2) Building the nodes for example. We have to edit the *CMakeLists.txt* file in the package to compile and build the source code. The following example code is responsible for building those two nodes above.

```
catkin_package(
    CATKIN_DEPENDS roscpp rospy std_msgs
)
include_directories( includ
    e
    ${catkin_INCLUDE_DIRS}
)
# This will create executables of the nodes
add_executable(demo_topic_publisher src/demo_topic_publisher.cpp)
add_executable(demo_topic_subscriber     src/demo_topic_subscriber.cpp)
```

```
# This will generate message header file before building the target
add_dependencies(demo_topic_publisher
mastering_ros_demo_pkg_generate_message_cpp)
add_dependencies(demo_topic_subscriber
mastering_ros_demo_pkg_generate_message_cpp)

# This will link executables to the appropriate libraries
target_link_libraries(demo_topic_publisher ${catkin_LIBRARIES})
target_link_libraries(demo_topic_subscriber ${catkin_LIBRARIES})
```

Build mastering_ros_demo_package as follow:

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

7. roslaunch:

Reference: *Mastering ROS for Robotics Programming, page 48*

(7-1) The *launch* files in ROS are a very useful feature for launching more than one node. It is difficult if we run each node in a terminal one by one. Instead of that, we can write all nodes inside a *XML* based file called *launch* files and using a command called *roslaunch*, we can parse this file and launch the nodes.

(7-2) Create a *launch* folder to keep the launch files

```
$ cd ~/catkin_ws/src/<ros_package>
$ mkdir launch
```

(7-3) The example launch file will launch two ROS nodes that are publishing and subscribing an integer value.

Demo_topic.launch

```
<launch>
    <node name=”publisher_node” pkg=”mastering_ros_demo_pkg”
type=”demo_topic_publisher” output=”screen” />

    <node name=”subscriber_node” pkg=”mastering_ros_demo_pkg”
type=”demo_topic_subscriber” output=”screen” />
</launch>
```

(7-4) After creating the launch file, you can launch it by using the following command:

```
$ roslaunch mastering_ros_demo_pkg demo_topic.launch
```

C. *rosserial* package

1. Arduino IDE setup

(1-1) Download Arduino IDE on your PC and Raspberry Pi by typing instructions below in Terminal. The first two instructions will take some time.

```
$ sudo apt-get update

$ sudo apt-get install arduino
```

(1-2) Set Serial Port Permission

Reference: https://www.arduino.cc/en/Guide/Linux

Connect your Arduino UNO board in the device you are setting in, if you're setting Arduino IDE in Raspberry pi then connect Arduino UNO with Raspberry Pi, so does your PC.

Open Terminal and type:

```
$ ls -l /dev/ttyACM*
```

You will get something like:

crw-rw---- 1 root dialout 188, 0 5 apr 23.01 ttyACM0

The "0" at the end of ACM might be a different number, or multiple entries might be returned. The data we need is "dialout" (is the group owner of the file).

Then add our user to the group by typing follow instruction in Terminal:

```
$ sudo usermod -a -G dialout <username>
```

Where <username> is your linux user name. You will need to log out and log in again for this change to take effect.

(1-3) You can test whether your Arduino IDE is work or not by typing *arduino* in Terminal. And after you can open Arduino sketch successfully, you can find a new folder called "sketchbook" in /home.

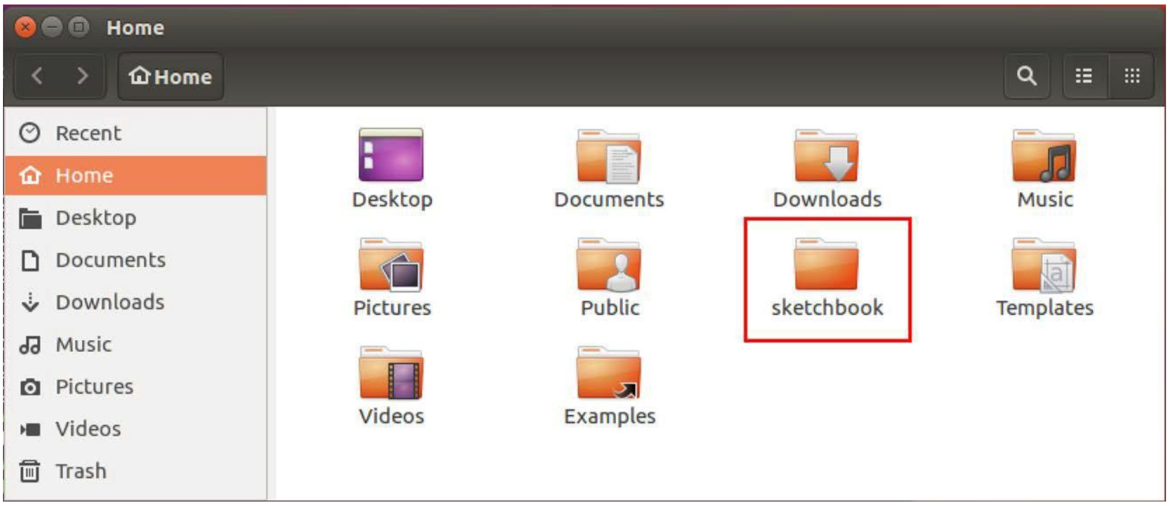


Fig.11 Find Arduino sketchbook folder

2. Installing the Software

Reference: http://wiki.ros.org/rosserial_arduino/Tutorials

(3-1) Installing on the ROS workstation

```
There are 2 options of how to install related libraries:
$ sudo apt-get install ros-melodic-rosserial-arduino
$ sudo apt-get install ros-melodic-rosserial
or
$ cd <ws>/src
$ git clone https://github.com/ros-drivers/rosserial.git
$ cd <ws>
$ catkin_make
$ catkin_make install
```

(3-2) Install *ros_lib* into the Arduino Environment

```
$ cd sketchbook/libraries
$ rm -rf ros_lib
$ rosrun rosserial_arduino make_libraries.py .
```

Notice that there is a dot (.) behind the python file. After restarting your IDE, you should see *ros_lib* listed under File>examples or File>sketchbook.

3. Create a publisher by using ***rosserial***

Reference : http://wiki.ros.org/rosserial_arduino/Tutorials/Hello%20World

(4-1) This example code is in the File>example>ros_lib>HelloWorld. This code will create a node publishes “Hello World” from Arduino.

Hello World.ino

```
#include <ros.h>
#include <std_msgs/String.h>

ros::NodeHandle nh;

std_msgs::String str_msg;
ros::Publisher chatter(“chatter”, &str_msg);

char hello[13] = “hello world”;

void setup()
{
    nh.initNode();
    nh.advertise(chatter);
}

```

```
void loop()
{
    str_msg.data = hello;
    chatter.publish( &str_msg)
    ; nh.spinOnce();
    delay(1000);
}
```

(4-2) To upload the code to your Arduino, use the upload function within the Arduino IDE. This is no different from uploading any othersketch.

(4-3) Running the code

Open a new Terminal and type:

```
$ roscore
```

Other new Terminal and type:

```
$ rosrun rosserial_python serial_node.py /dev/ttyACM0
```

Another new Terminal and type:

```
$ rostopic echo /chatter
```

```
isci@mobile:~$ rostopic echo chatter
data: "hello world!"
---
data: "hello world!"
---
data: "hello world!"
---
data: "hello world!"
---
data: "hello world!"
---
data: "hello world!"
---
data: "hello world!"
---
data: "hello world!"
---
data: "hello world!"
---
data: "hello world!"
---
```

4. Create a subscriber by using ***rosserial***

Reference: http://wiki.ros.org/rosserial_arduino/Tutorials/Blink

(5-1) This example code is in the File>example>ros_lib>Blink. This code will create a subscriber and the LED on the Arduino will toggle every time receive a empty message from Raspberry Pi.

Blink.ino

```
#include <ros.h>
```

```
#include <std_msgs/Empty.h>

ros::NodeHandle nh;

void messageCb( const std_msgs::Empty&
    toggle_msg){ digitalWrite(13, HIGH-digitalRead(13));  //
    blink the led
}

ros::Subscriber<std_msgs::Empty> sub("toggle_led", &messageCb );

void setup()
{
    pinMode(13, OUTPUT);
    nh.initNode();
    nh.subscribe(sub);
}

void loop()
{
    nh.spinOnce();
    delay(1);
```

(5-2) To upload the code to your Arduino, use the upload function within the Arduino IDE.

(5-3) Running the code

Open a new Terminal and type:

```
$ roscore
```

Other new Terminal and type:

```
$ rosrun rosserial_python serial_node.py /dev/ttyACM0
```

Another new Terminal and type:

```
$ rostopic pub toggle_led std_msgs/Empty --once
```

- Hardware architecture