

Introduction to The Robot Operating System (ROS)

Presented by Omar Gamal

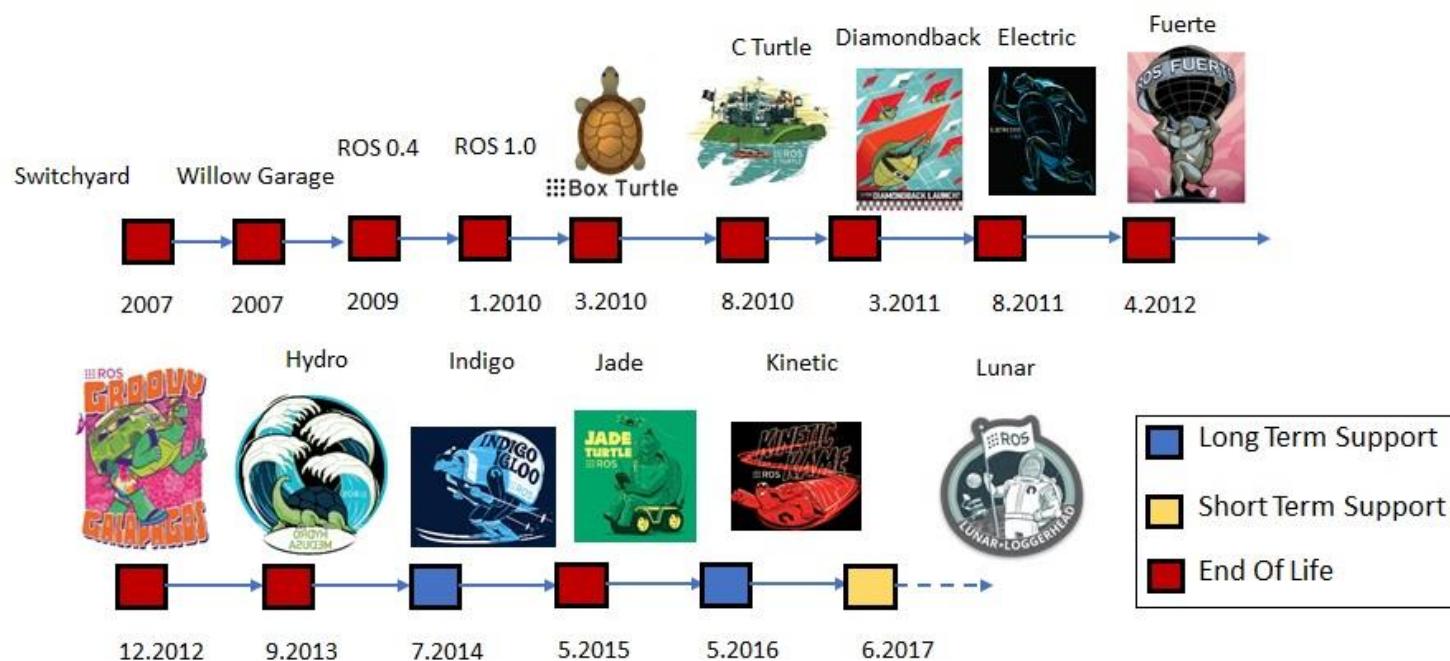
Agenda

- **Introduction**
- **ROS Concepts**
- **Organizing and Building ROS Code**
- **ROS Across Multiple Machines**
- **Lab Assignment**
- **Installing and Configuring ROS Indigo Environment in VirtualBox**
- **Create ROS Catkin Workspace and Package**
- **Configuring EV3 Control Packages in ROS**
- **ROS ⇔ EV3 Communication Setup**
- **ROS ⇔ EV3 Communication over Network**
- **Lego Mindstorms Ev3 Sensors**
- **References**
- **Questions**

Introduction

The Robot Operating System (ROS)

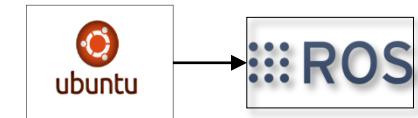
ROS is a flexible framework for writing robot software. It is a collection of tools, libraries, and conventions that aim to simplify the task of creating complex and robust robot behaviour across a wide variety of robotic platforms.



ROS Environment Installation

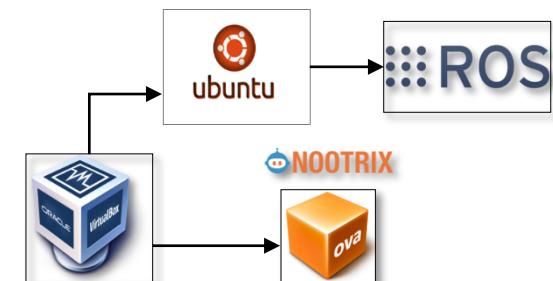
Operating System:

1. Installing ROS desktop-full configuration



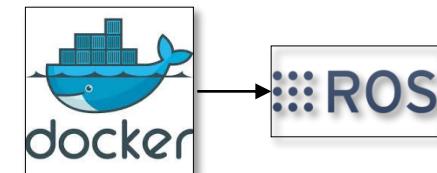
Virtual box:

1. Ubuntu desktop installation + Installing ROS desktop-full configuration.
2. NOOTRIX ".ova" archive with a preinstalled ROS desktop-full configuration.



Container:

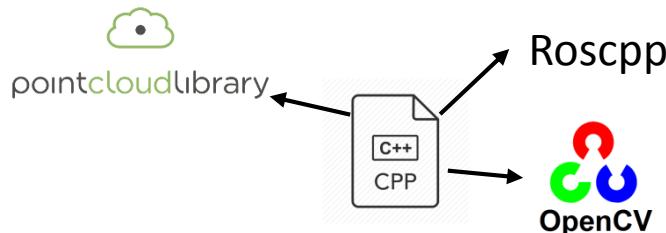
1. Installing ROS container image on Docker.



ROS Concepts

ROS Node

A **Node** is an executable that uses ROS to communicate with other nodes.

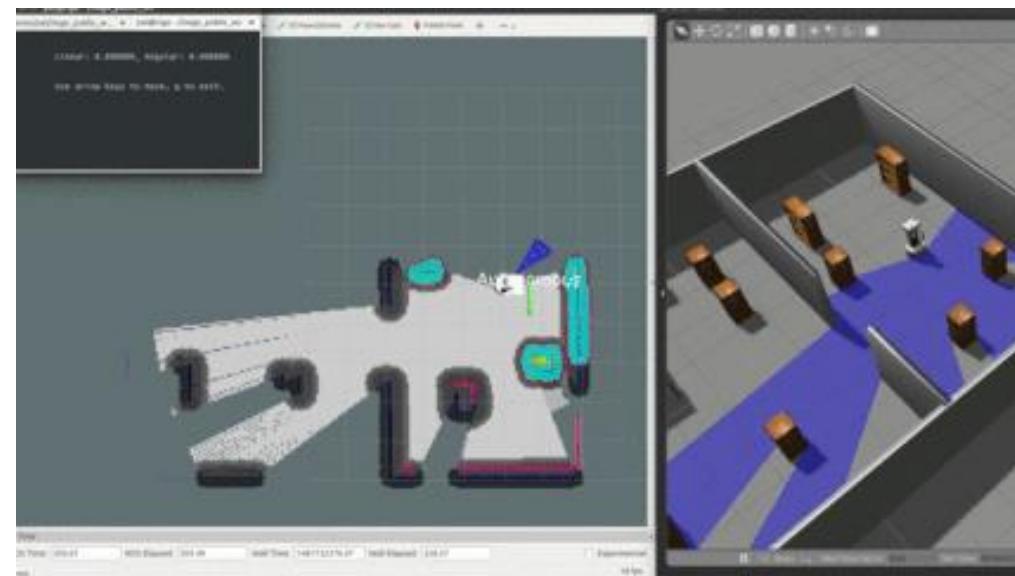


```
#include “....”

int main(int argc, char **argv)
{
    ros::init(argc, argv, “Mapping_Node”);

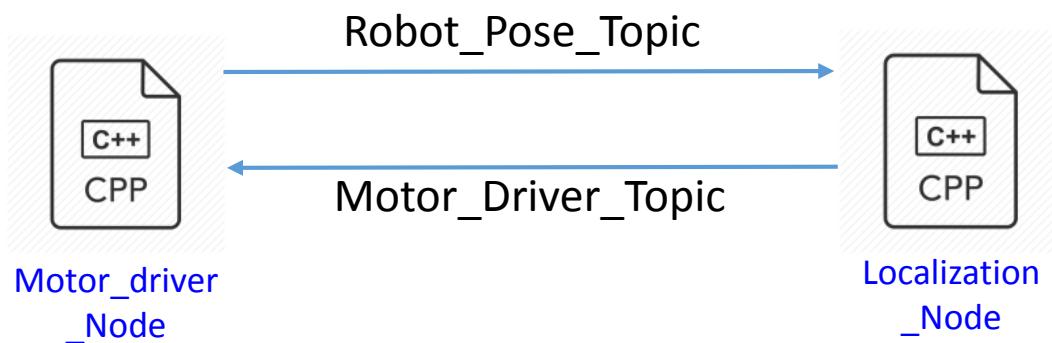
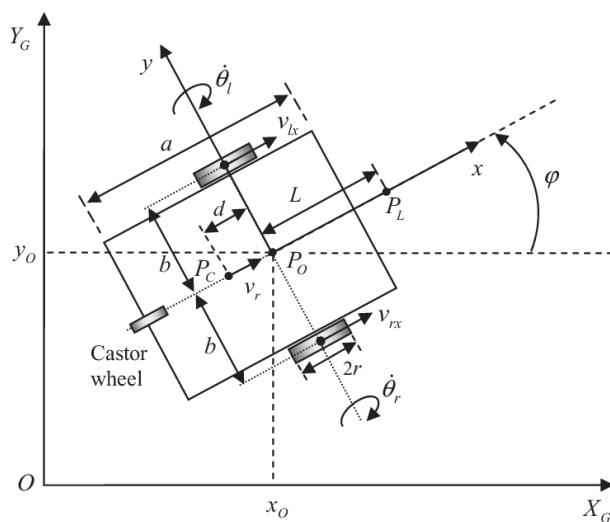
    /* ..... Image Processing Code ..... */

}
```



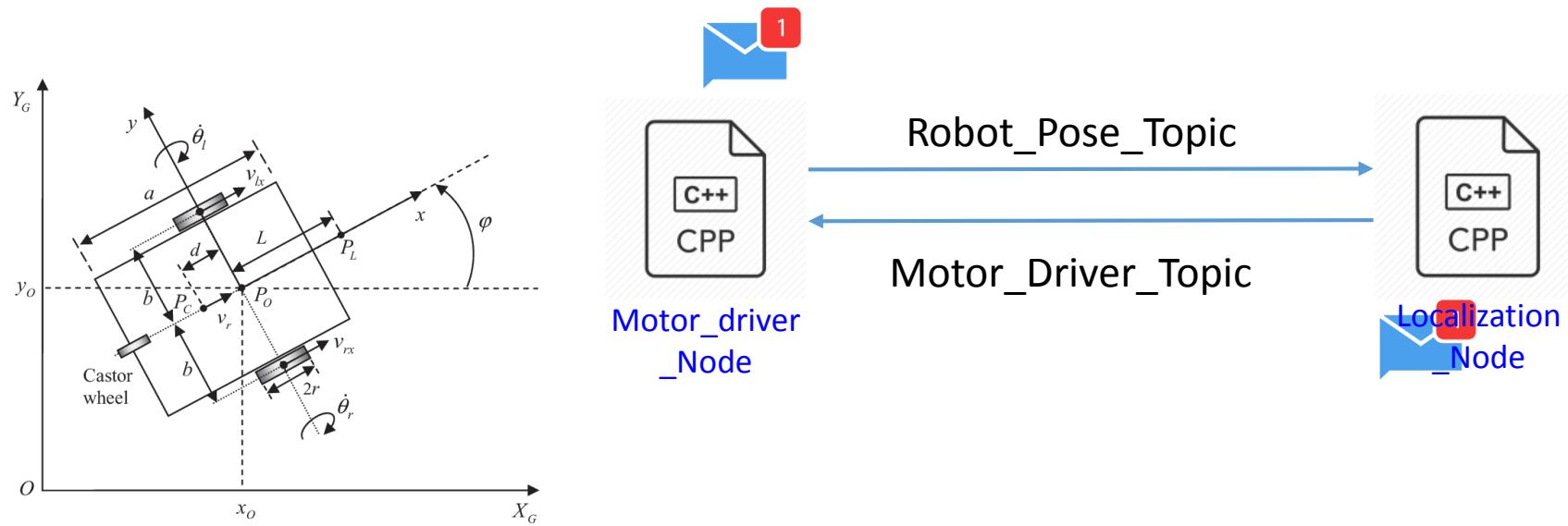
Topics

Topics are named buses over which nodes exchange messages. ROS currently supports TCP/IP-based and UDP-based message transport.



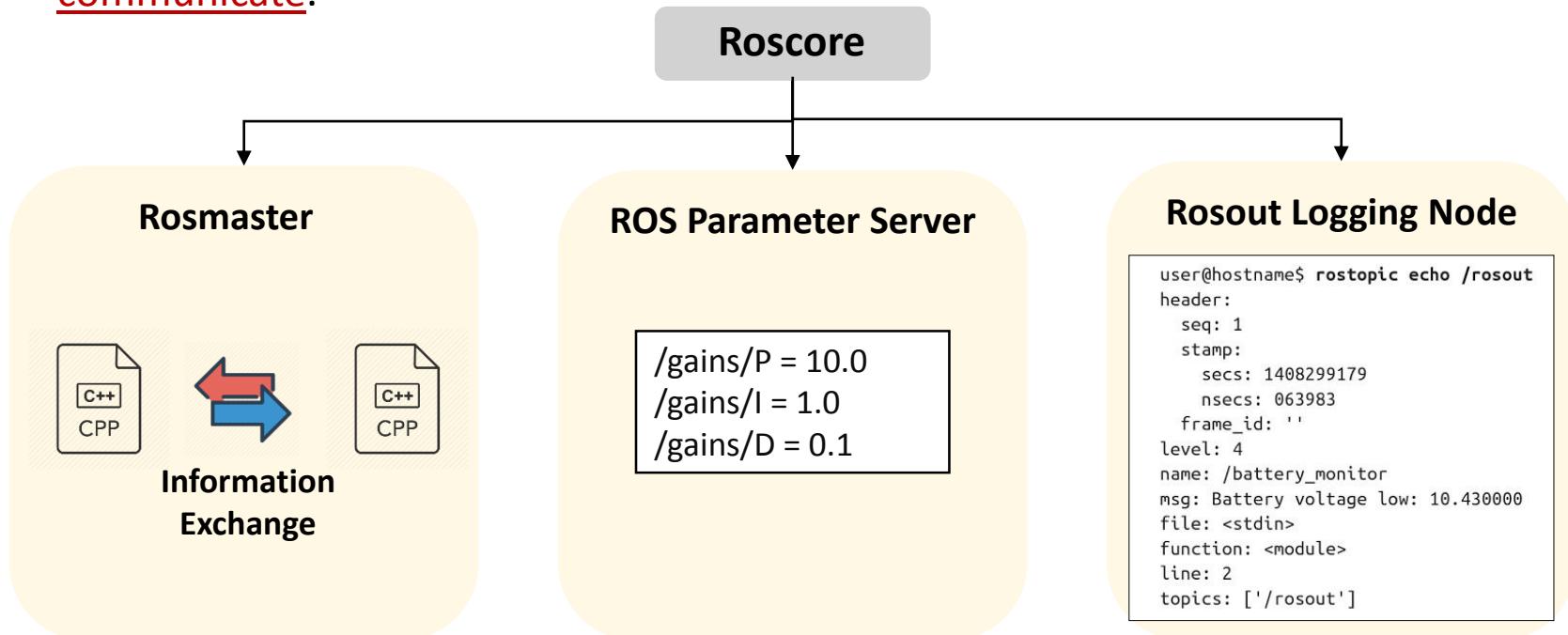
ROS Message

A **message** is a simple data structure, comprising typed fields. ROS provides a set of standard message formats that cover most of the common use cases in robotics such as Geometric concepts, Sensors, Navigation data, and Diagnostic.

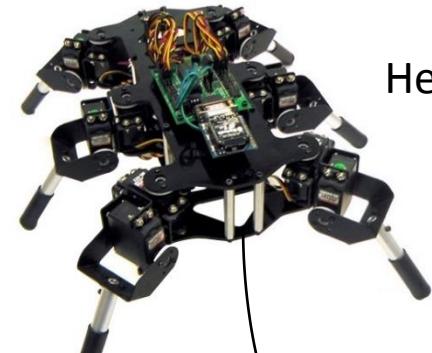
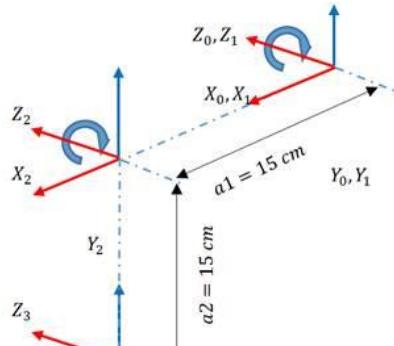


ROS Core

Roscore: is a collection of nodes and programs that are pre-requisites of a ROS-based system. You must have a roscore running in order for ROS nodes to communicate.



Example: Publisher & Subscriber



Hexapod

Nodes

Message

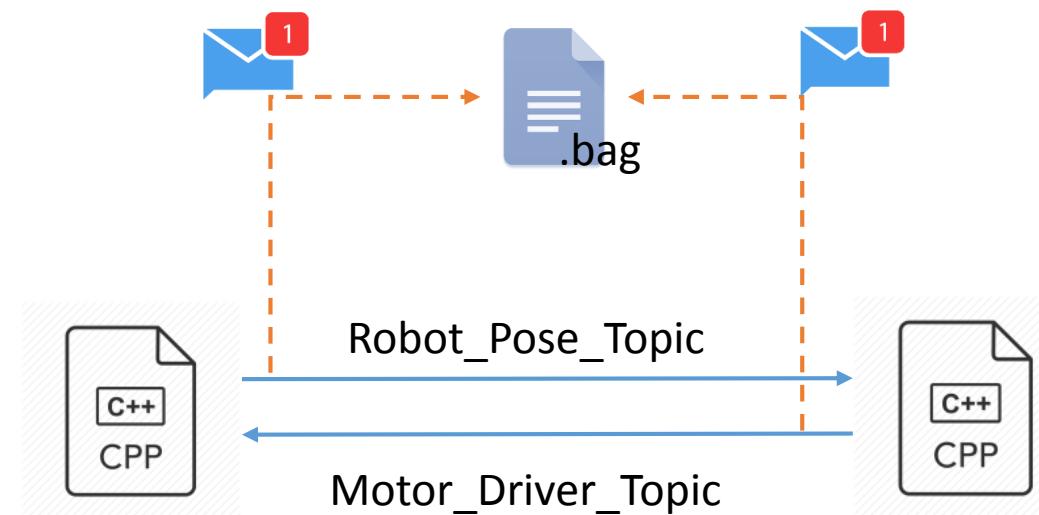
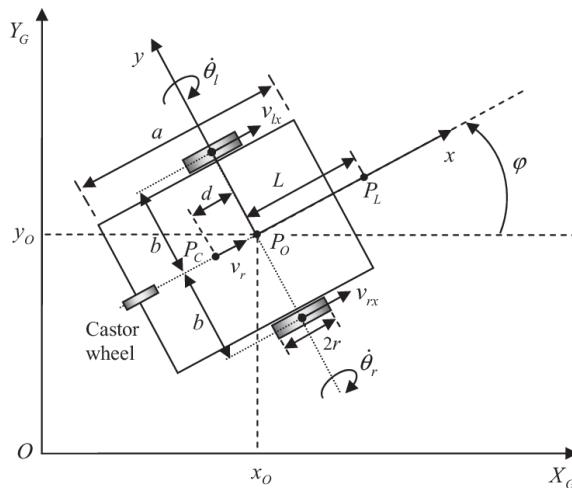
Topics

Master

```
#include "....."  
#include "....."  
  
void subscriberCallback(const geometry_msgs::PoseStamped::ConstPtr& msg)  
{  
  
    ROS_INFO("\n \n Tool_pose: (%.2f, %.2f, %.2f) \n orientation:(%.2f, %.2f, %.2f, %.2f) \n", msg ->pose.position.x, msg ->pose.position.y, msg ->pose.position.z, msg ->pose.orientation.x, msg ->pose.orientation.y, msg ->pose.orientation.z, msg ->pose.orientation.w);  
  
}  
  
int main(int argc, char **argv)  
{  
    ros::init(argc, argv, "subscriber_node");  
    ros::Subscriber sub = n.subscribe("sensor_data", 1000, subscriberCallback);  
    ros::spin();  
  
    return 0;  
}
```

ROS Bag

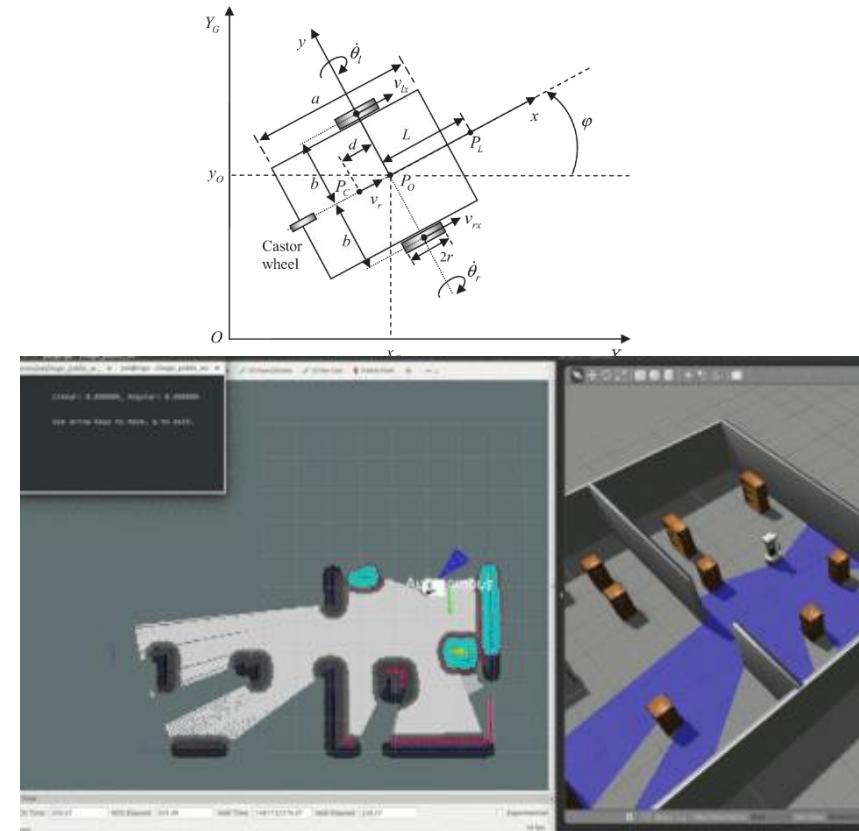
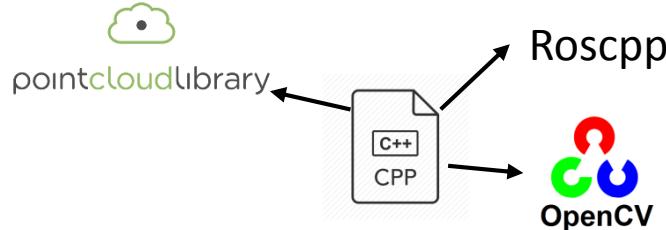
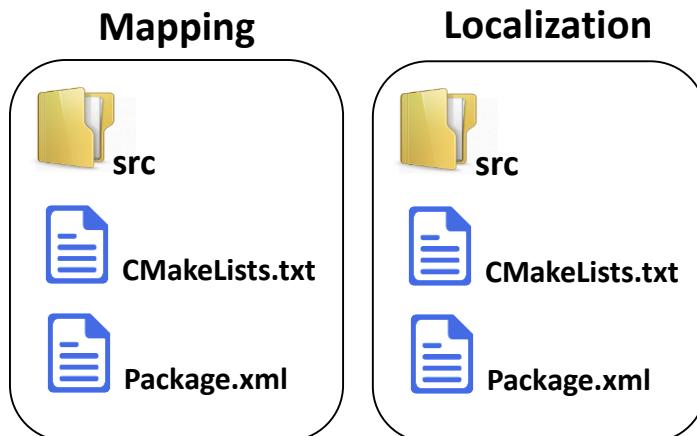
A **bag** is a file format in ROS for storing ROS message data. Bags are typically created by a tool like [rosbag](#), which subscribe to one or more ROS topics, and store the serialized message data in a file as it is received. These bag files can also be played back in ROS to the same topics they were recorded from, or even remapped to new topics.



Organizing and Building ROS Code

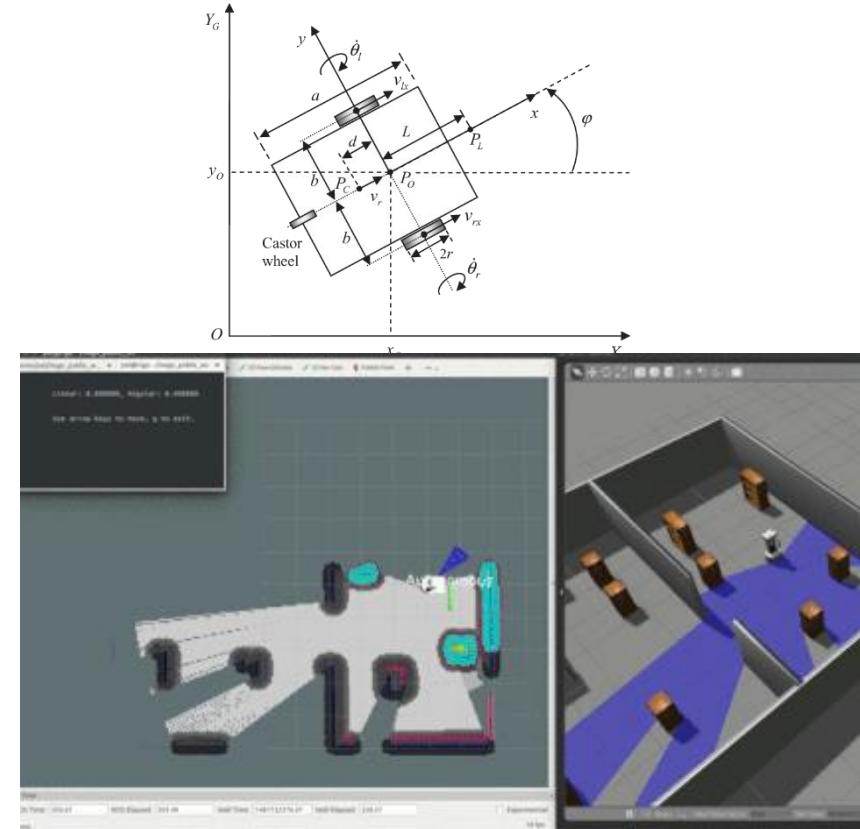
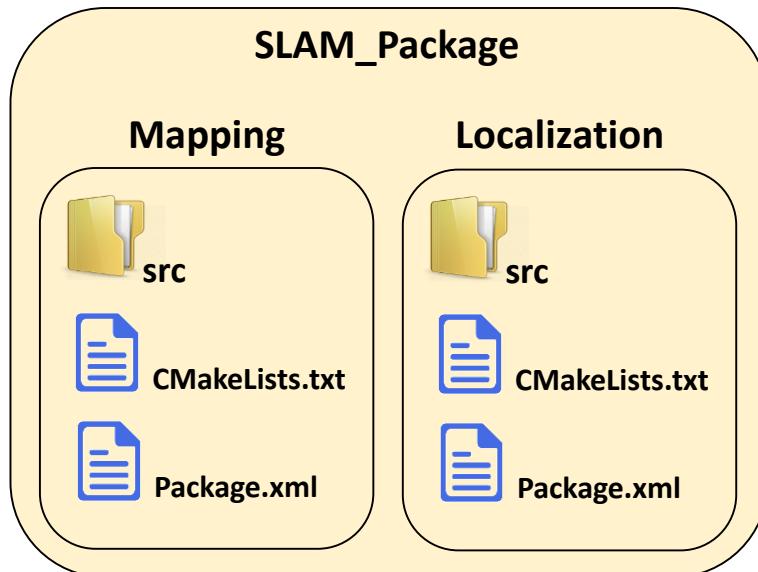
ROS Package

Packages: are the main unit for organizing software in ROS.



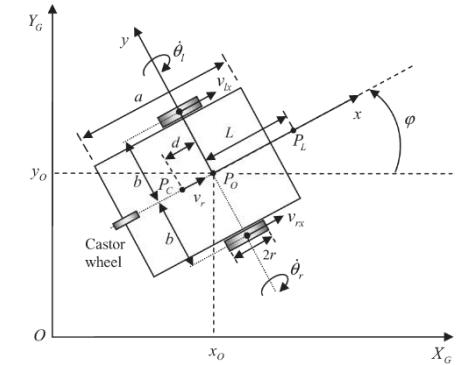
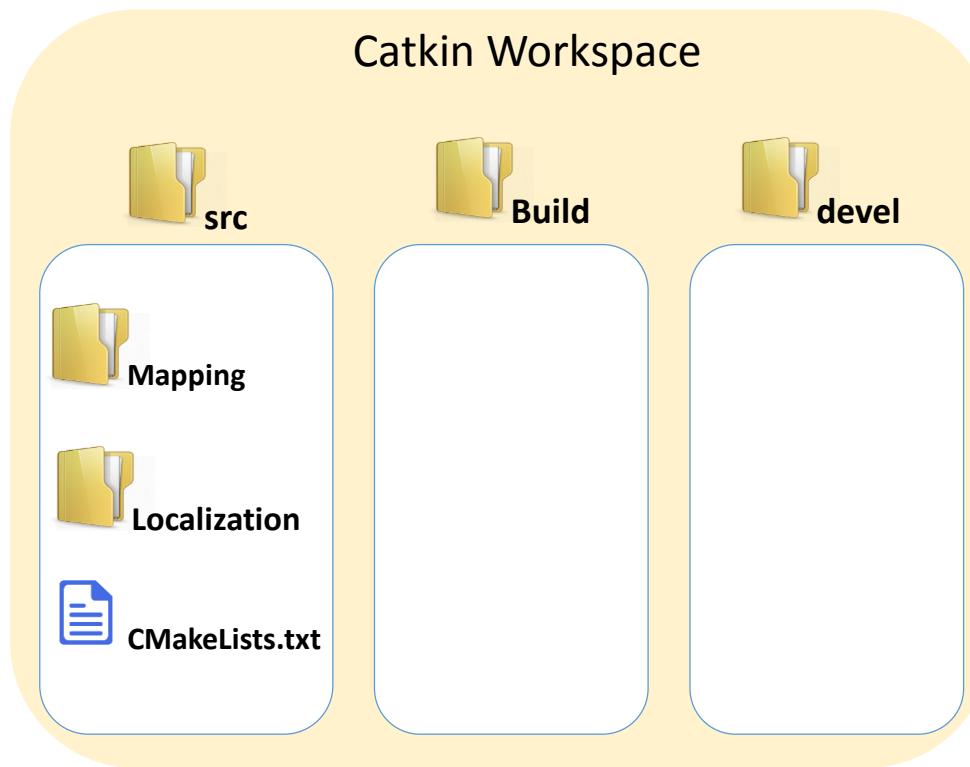
ROS Metapackages

Metapackages: are specialized Packages which only serve to represent a group of related other packages.



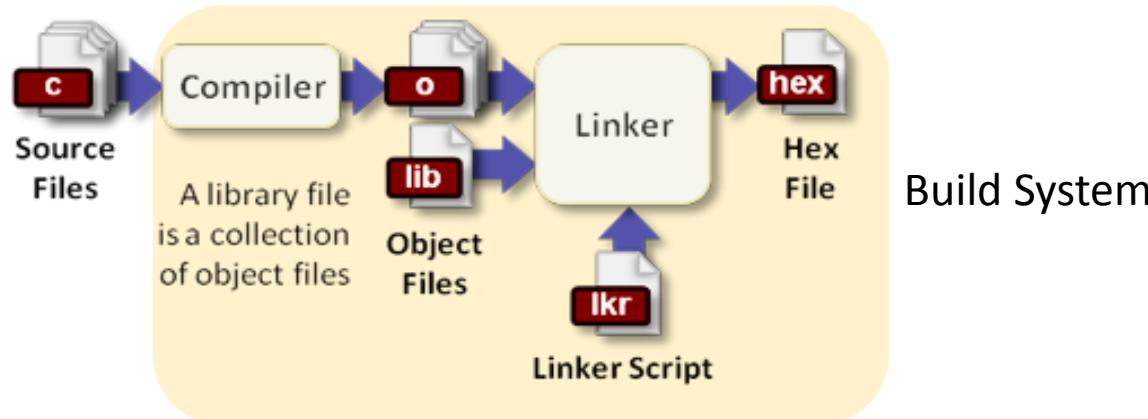
Catkin Workspace

A [catkin workspace](#) is a folder where you modify, build, and install catkin packages.



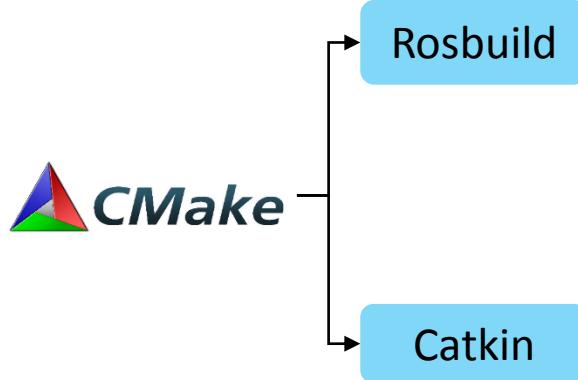
What is a Build System?

A **build system** is responsible for generating 'targets' from raw source code that can be used by an end user.



ROS Custom Build System

“ROS utilizes a custom build system, that extends CMake to manage dependencies between packages.”

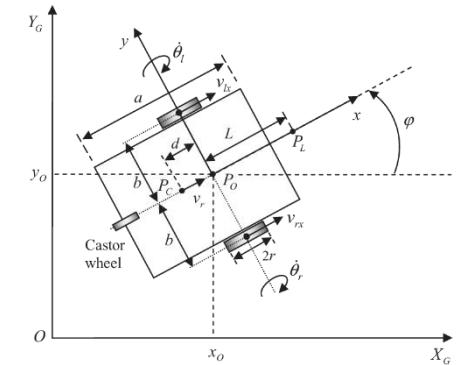
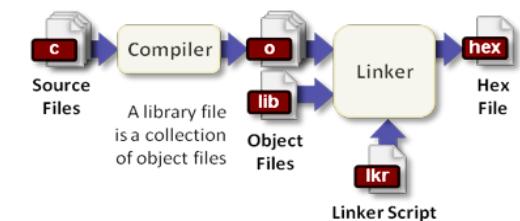
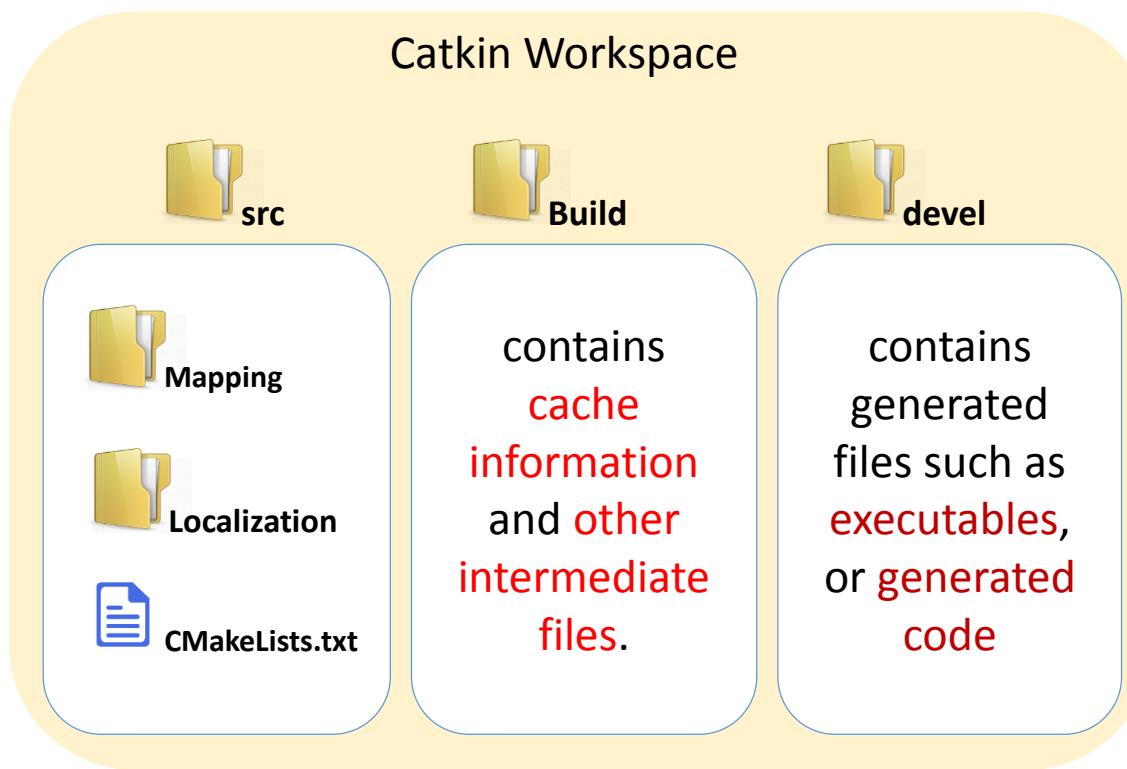


Drawbacks

- Rapid growth of the ROS.
- The difficulty of installing ROS on other operating systems.
- Low build speed.

Catkin Workspace

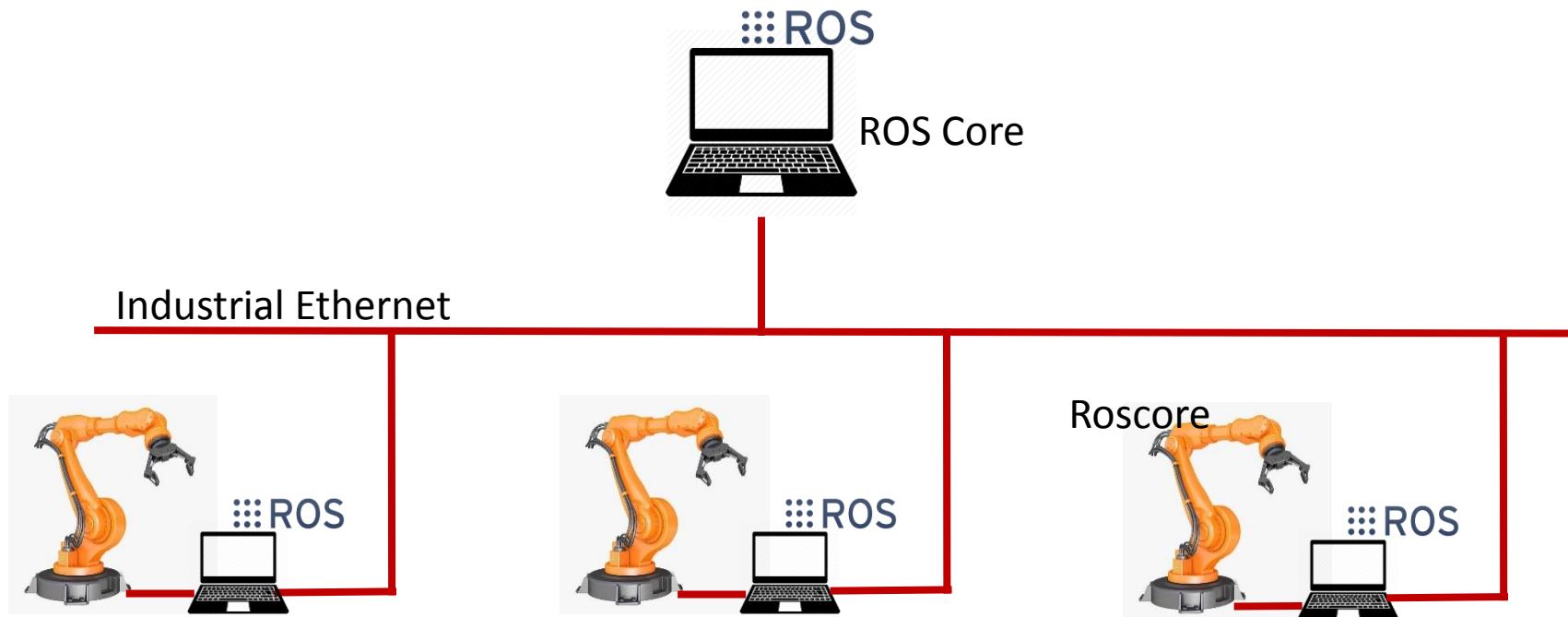
A **catkin workspace** is a folder where you modify, build, and install catkin packages.



ROS Across Multiple Machines

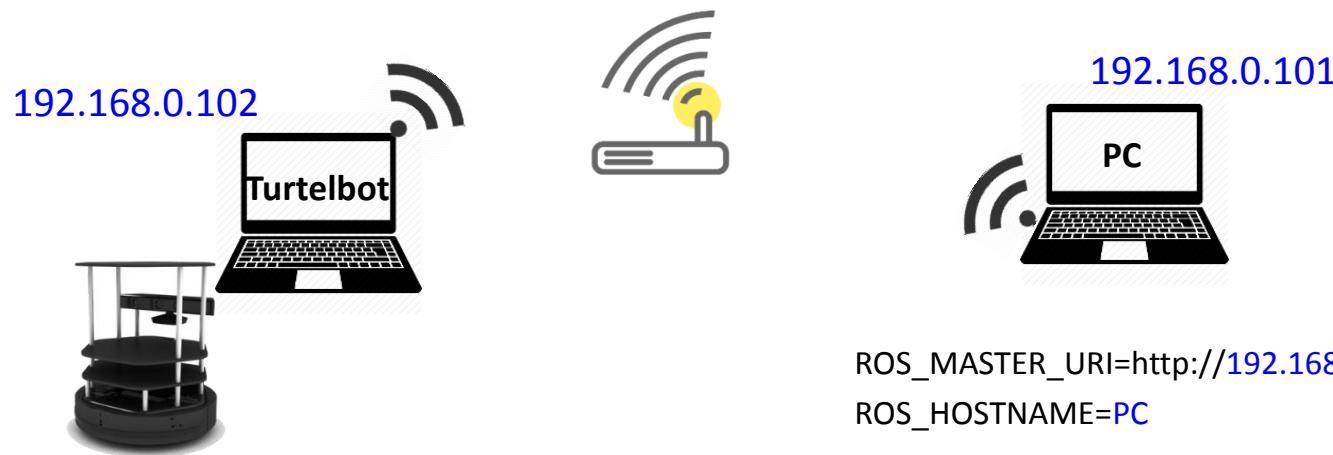
ROS Network Configuration Requirements

ROS is a distributed computing environment. As a result, ROS has certain requirements of the network configuration:



ROS Network Setup

- **ROS_MASTER_URI**: is a required setting that tells nodes where they can locate the master.
- **ROS_IP** and **ROS_HOSTNAME** are optional environment variable that sets the declared network address of a ROS Node or tool.



ROS_MASTER_URI=<http://192.168.0.101:11311>

ROS_HOSTNAME=Turtelbot

ROS_MASTER_URI=<http://192.168.0.101:11311>
ROS_HOSTNAME=PC

Lab Assignment #2: Extending Publisher and Subscribe Node to Send and Receive Sensor Data



Task Pre-requirements

1. Installing and Configuring ROS Indigo Environment in a VirtualBox. “[Tutorial#1](#)”
2. Create a ROS Catkin Workspace. “[Tutorial#2](#)”
3. Download course packages in the source space directory. “[Tutorial#2](#)”

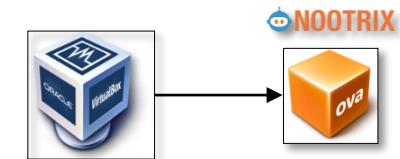
Tutorial #1: Installing and Configuring ROS Indigo Environment in VirtualBox

Installing ROS Indigo in VirtualBox

1. The first thing you have to do, is to obtain VirtualBox. Visit the VirtualBox website's download page. Install it the same way as any normal Windows program.

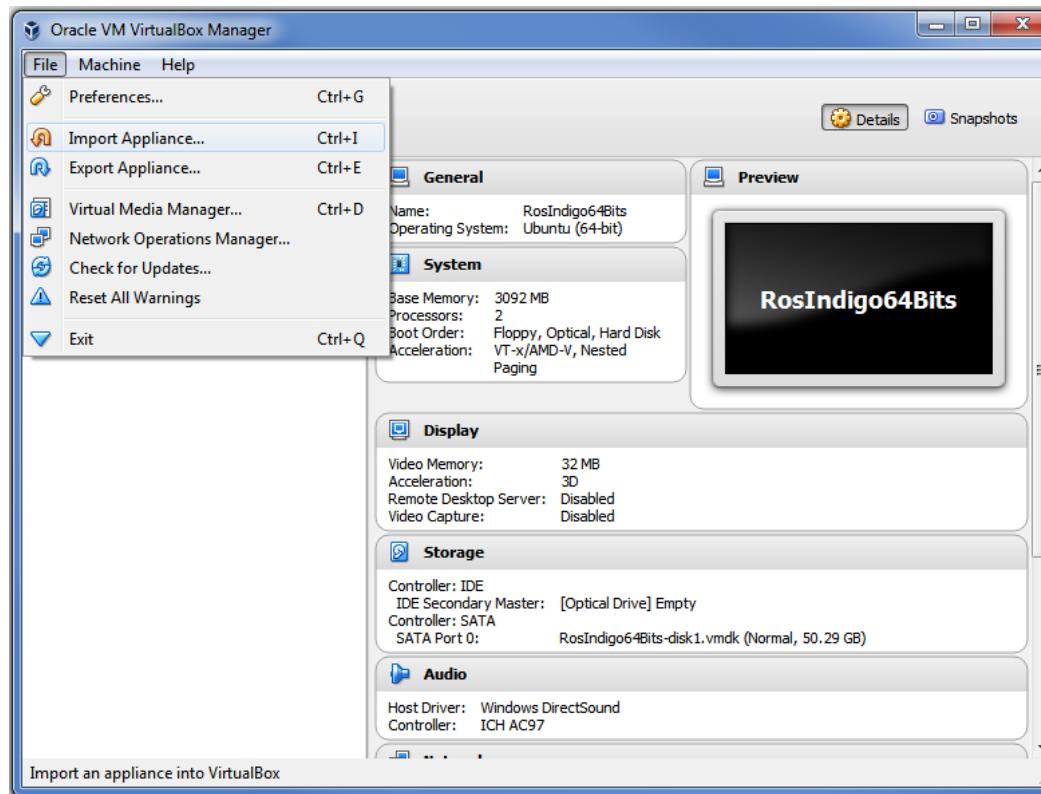
<https://www.virtualbox.org/wiki/Downloads>

2. Download ROS indigo OVA file archive from >>>>



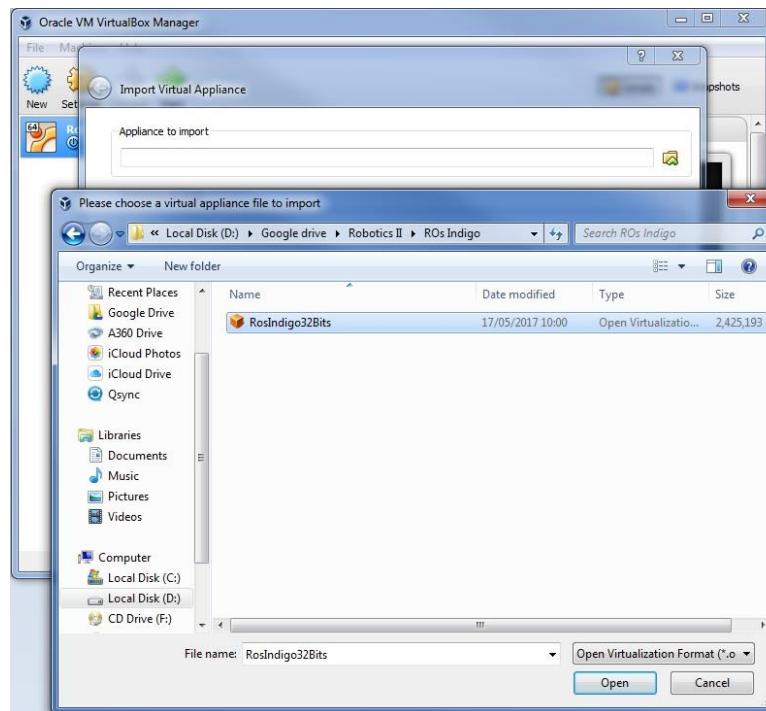
Installing ROS Indigo in VirtualBox

3. After you launch VirtualBox from the Windows Start menu, click on File menu and choose Import Appliance.



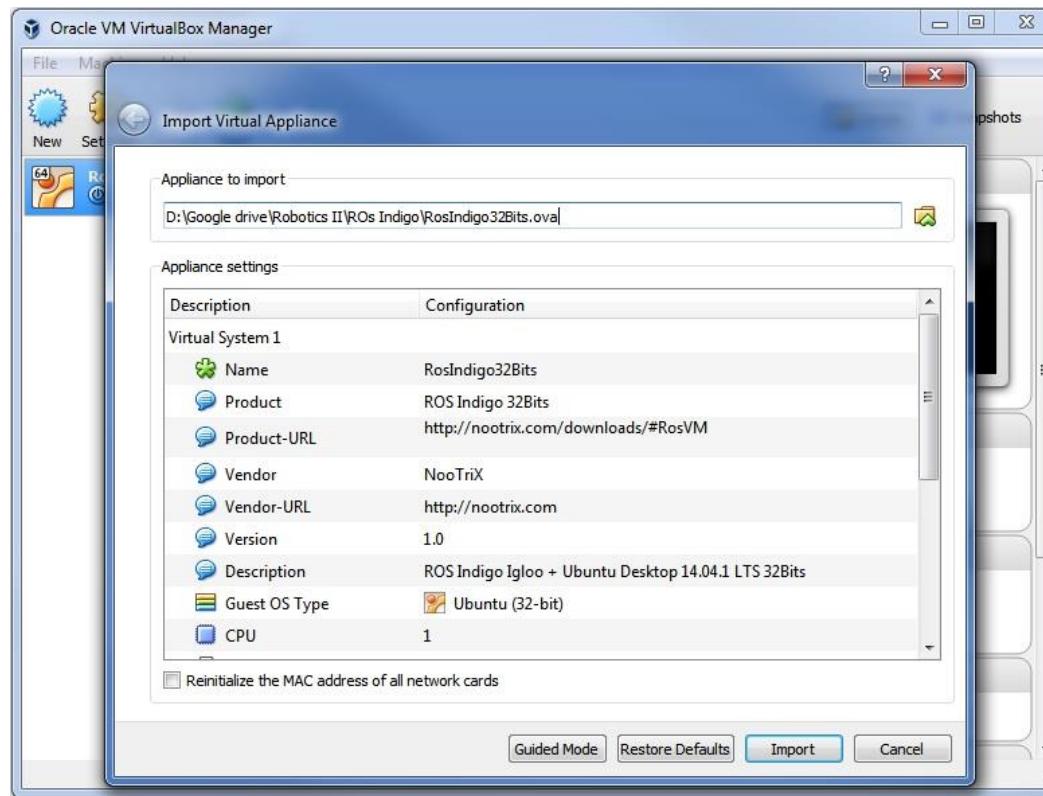
Installing ROS Indigo in VirtualBox

4. In the Appliance Import section, choose the virtual appliance file to import Which in our case the Rosindigo32Bits.ova File. You can choose between 32Bits or 64Bits according to your windows version.



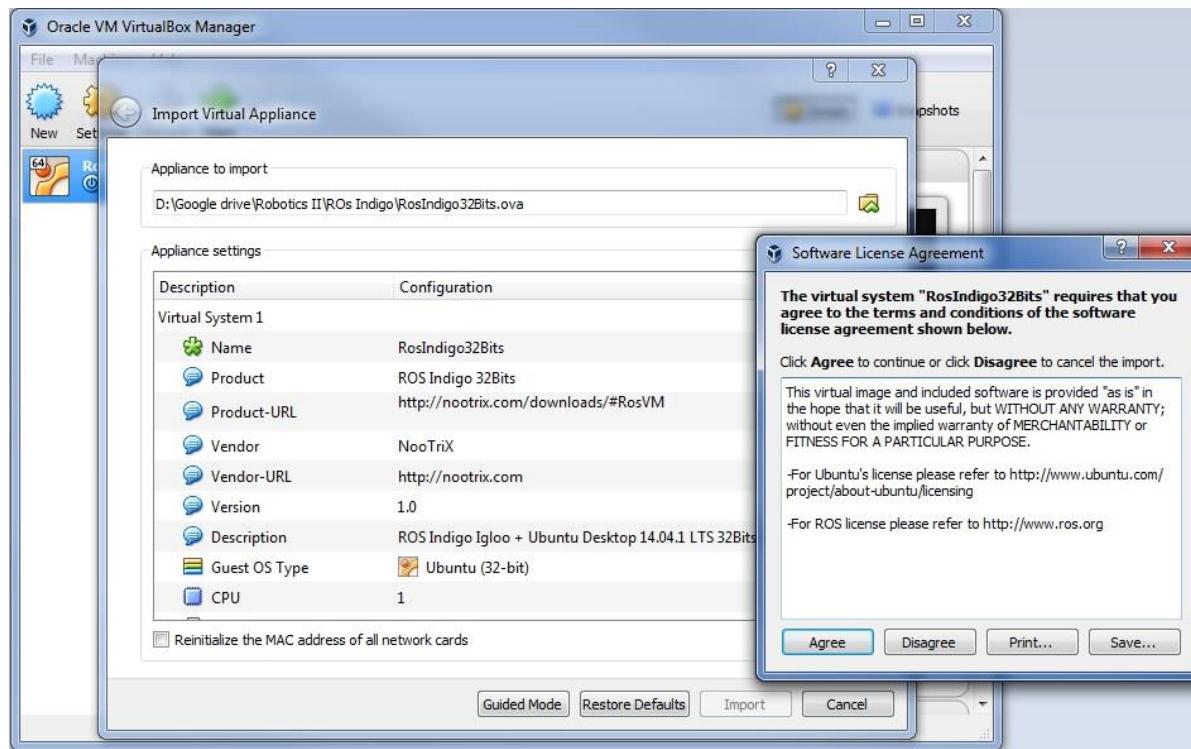
Installing ROS Indigo in VirtualBox

5. Now we are ready to choose Import.



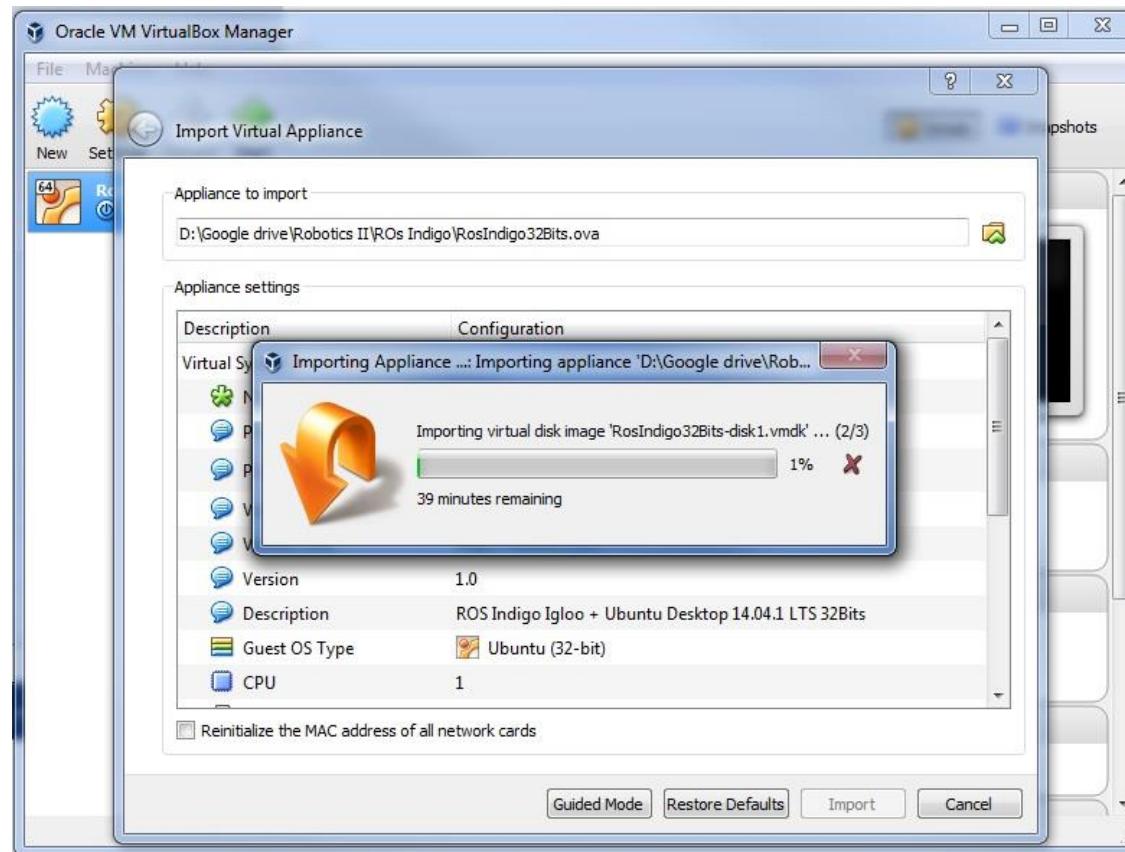
Installing ROS Indigo in VirtualBox

6. A new dialog will appear and asks if we agree to the software terms and conditions, you can press agree to continue.



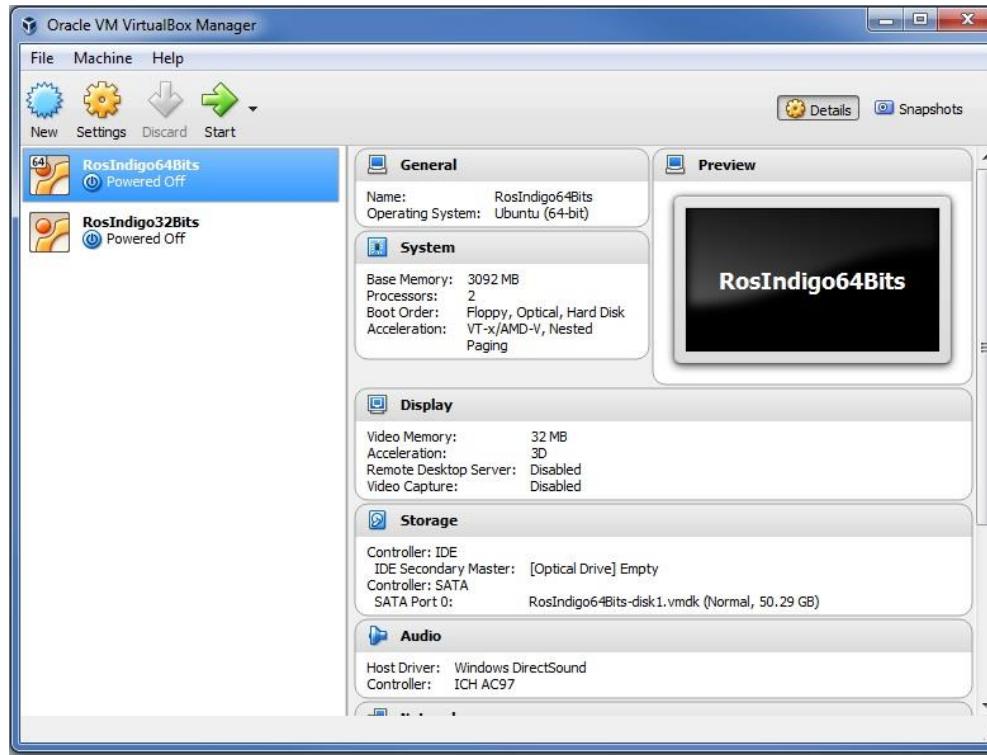
Installing ROS Indigo in VirtualBox

7. Now VirtualBox will start importaing our .ova file.



Installing ROS Indigo in VirtualBox

- Once it is finished, you will find ROS Indigo in the list of the installed virtual machines. Now you can select it, and press start.



Tutorial #2: Create a ROS Catkin Workspace and Package

Create and Build a Catkin Workspace

1. Create a catkin_ws directory

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

2. Using catkin_make command will create a CMakeLists.txt link in the 'src' folder.
Additionally, a 'build' and 'devel' folder are created.

```
$ catkin_make
```

3. Source your new setup.*sh file:

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

4. Ensure ROS_PACKAGE_PATH environment variable includes the directory we are in. “/home/youruser/catkin_ws/src:/opt/ros/kinetic/share”

```
$ echo $ROS_PACKAGE_PATH
```

Course Packages

1. Extract the downloaded course packages in the source directory folder.
2. Now you need to build the downloaded packages in the catkin workspace:

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

“Tutorial#3” Configuring EV3 Control Packages in ROS

Configuring EV3 Control Packages

1. First change to the catkin workspace

```
$ cd ~/catkin_ws
```

2. Now we need to install some prerequisite packages for the EV3 control packages to work. Start executing the following commands one by one.

```
$ sudo apt-get install ros-indigo-ros-control ros-indigo-ros-controllers  
$ sudo apt-get install ros-indigo-joint-state-controller  
$ sudo apt-get install ros-indigo-effort-controllers  
$ sudo apt-get install ros-indigo-position-controllers
```

3. Now we need to source the generated source files.

```
$ . ~/catkin_ws/devel/setup.bash
```

Configuring EV3 Control Packages

4. First change to the source space directory of the catkin workspace

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

5. Now we need to download and extract the EV3 Control packages in the source space directory.
6. Now we need to build the packages in the catkin workspace:

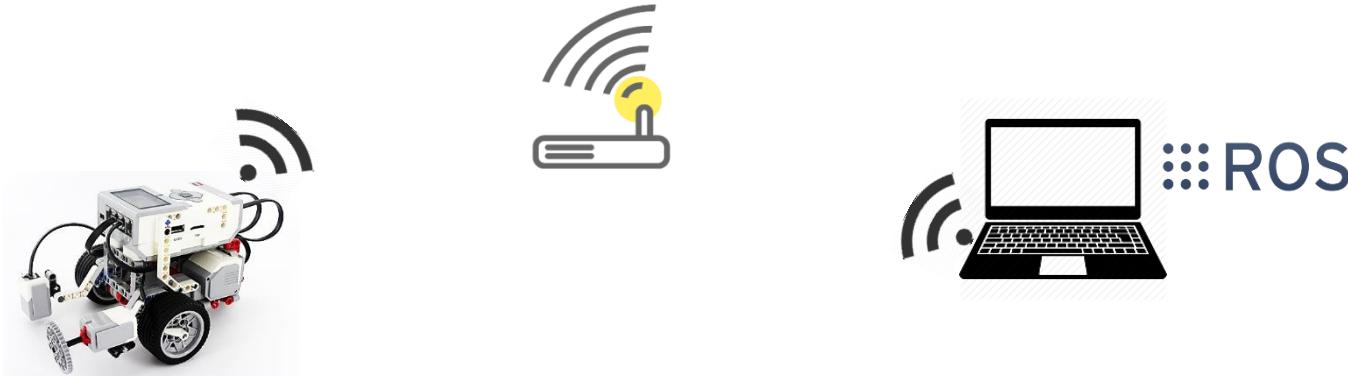
```
$ cd ~/catkin_ws
$ catkin_make
```

Tutorial#4

ROS ⇔ EV3 Communication Setup



ROS ↔ EV3 Communication over Network



In order to configure the communication between ROS and the EV3 we need to adjust the ROS environment variables on both sides (i.e. ROS and the EV3),

- ROS_MASTER_URI
- ROS_IP
- and ROS_HOSTNAME

First: Network Setup in ROS Side



1. First we need to know the IP address of our PC. In a new terminal window write,

```
$ ifconfig
```

2. You should have some output like the following. Note your IP address as we will need it in the following steps. In my case here **192.168.0.10.**

```
viki@c3po:~/catkin_ws$ ifconfig
eth0      Link encap:Ethernet HWaddr 08:00:27:af:de:fe
          inet addr:10.0.2.15 Bcast:10.0.2.255 Mask:255.255.255.0
              inet6 addr: fe80::a00:27ff:feaf:defe/64 Scope:Link
                  UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1
                  RX packets:9253 errors:0 dropped:0 overruns:0 frame:0
                  TX packets:1860 errors:0 dropped:0 overruns:0 carrier:0
                  collisions:0 txqueuelen:1000
                  RX bytes:12542111 (12.5 MB) TX bytes:152253 (152.2 KB)

lo       Link encap:Local Loopback
          inet addr:127.0.0.1 Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
              UP LOOPBACK RUNNING MTU:65536 Metric:1
              RX packets:228 errors:0 dropped:0 overruns:0 frame:0
              TX packets:228 errors:0 dropped:0 overruns:0 carrier:0
              collisions:0 txqueuelen:0
              RX bytes:23854 (23.8 KB) TX bytes:23854 (23.8 KB)

viki@c3po:~/catkin_ws$
```

First: Network Setup in ROS Side

3. Now we need to edit the `~/.bashrc` file, and add the following lines to adjust Ros enviroment variables (i.e. `ROS_MASTER_URI`, `ROS_IP`, and `ROS_HOSTNAME`) for the network setup.
4. First change to the catkin workspace,



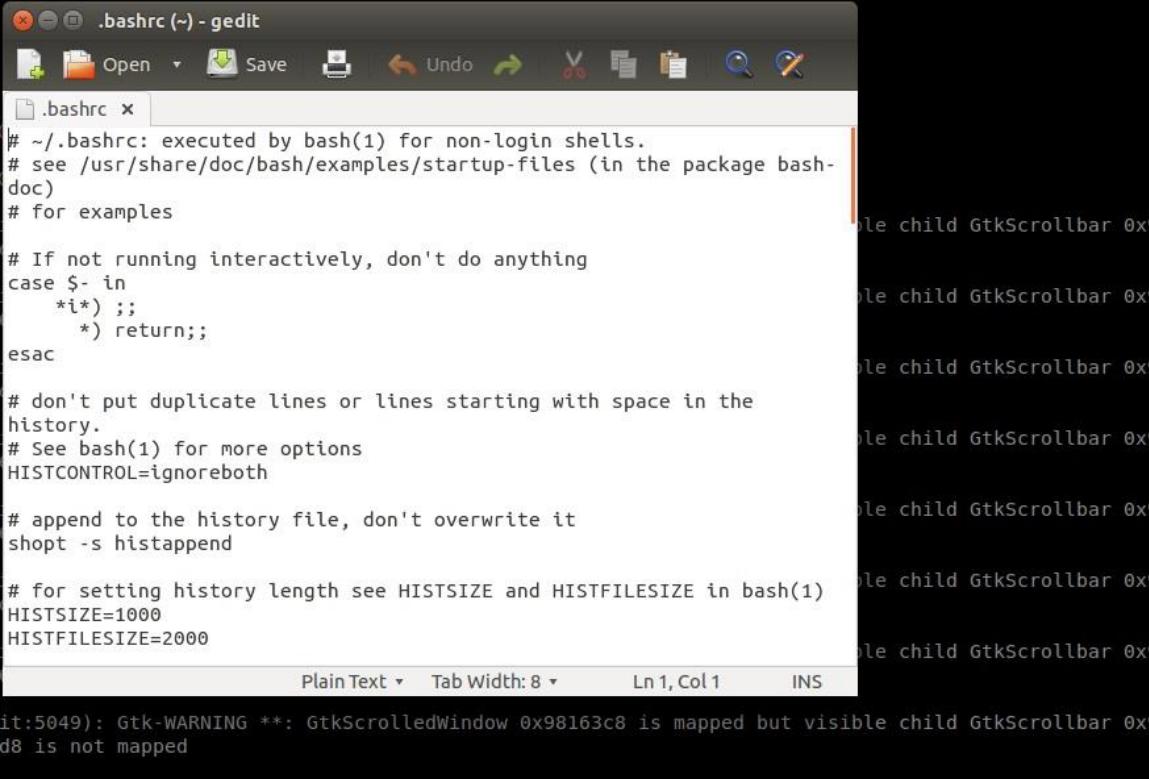
```
$ cd ~/catkin_ws
```

5. Now use the gedit command to edit the `~/.bashrc` file

```
$ gedit ~/.bashrc
```

First: Network Setup in ROS Side

6. A text file should now open as it shown.



The screenshot shows a terminal window with a dark background. On the left, there is a vertical scroll bar. On the right, there is a horizontal scroll bar. The terminal window displays the contents of the .bashrc file in gedit. The file contains several lines of shell script code, including comments and variable assignments. At the bottom of the terminal window, there is a status bar with the text "Plain Text" and "Tab Width: 8". The status bar also shows the current line and column position as "Ln 1, Col 1".

```
# ~/.bashrc: executed by bash(1) for non-login shells.
# see /usr/share/doc/bash/examples/startup-files (in the package bash-doc)
# for examples

# If not running interactively, don't do anything
case $- in
    *i*) ;;
    *) return;;
esac

# don't put duplicate lines or lines starting with space in the
history.
# See bash(1) for more options
HISTCONTROL=ignoreboth

# append to the history file, don't overwrite it
shopt -s histappend

# for setting history length see HISTSIZE and HISTFILESIZE in bash(1)
HISTSIZE=1000
HISTFILESIZE=2000
```

First: Network Setup in ROS Side

7. Since we are going to run our Ros master on our PC. Then we need to inform all robots in our ROS Network, that we will run our Ros master on our PC which has the IP adress 192.168.0.10 –we got this IP adress before from step2 -. In order to do that Programmaticlly we need to add the following line at the end of the bash file.

```
> echo export ROS_MASTER_URI=http://192.168.0.10:11311 >> ~/.bashrc
```

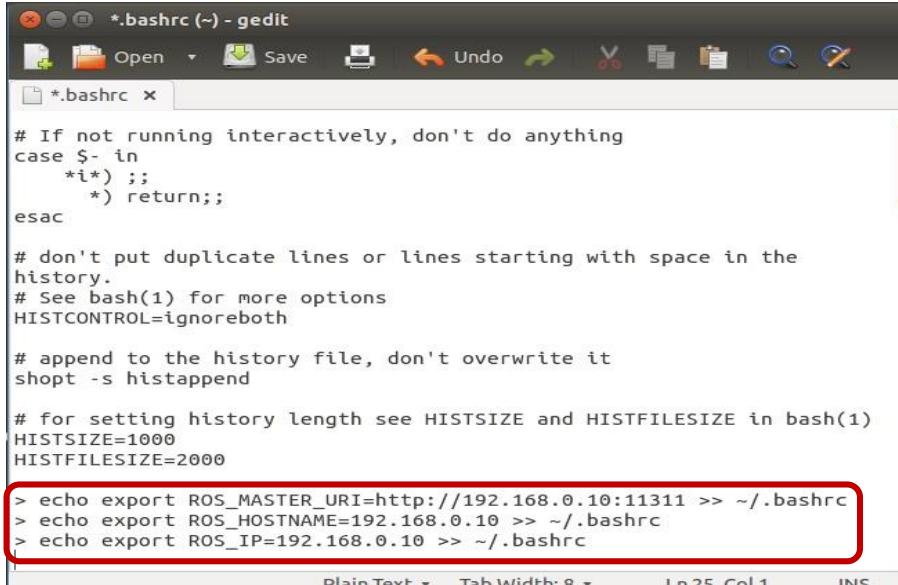
8. Now we need to identify the ROS_IP and ROS_HOSTNAME environment variables, in order to force ROS to use our PC IP address (i.e. 192.168.0.10) for all nodes generated on our PC.

First: Network Setup in ROS Side

- In order to do that Programmatically we need to add the following two lines at the end of the bash file as well.

```
> echo export ROS_HOSTNAME=192.168.0.10 >> ~/.bashrc  
> echo export ROS_IP=192.168.0.10 >> ~/.bashrc
```

- Now you should have something like this. If so press save and close the `~/.bashrc` file and all the opened terminals so all the changes can take place.



```
# If not running interactively, don't do anything  
case $- in  
    *i*) ;;  
    *) return;;  
esac  
  
# don't put duplicate lines or lines starting with space in the  
history.  
# See bash(1) for more options  
HISTCONTROL=ignoreboth  
  
# append to the history file, don't overwrite it  
shopt -s histappend  
  
# for setting history length see HISTSIZE and HISTFILESIZE in bash(1)  
HISTSIZE=1000  
HISTFILESIZE=2000  
  
> echo export ROS_MASTER_URI=http://192.168.0.10:11311 >> ~/.bashrc  
> echo export ROS_HOSTNAME=192.168.0.10 >> ~/.bashrc  
> echo export ROS_IP=192.168.0.10 >> ~/.bashrc
```

Second: Network Setup in EV3 Side

1. Insert the SD Card on the EV3.
2. Connect a keyboard and the EDIMAX Wifi nano USB adapter to the EV3 using a USB Hub.



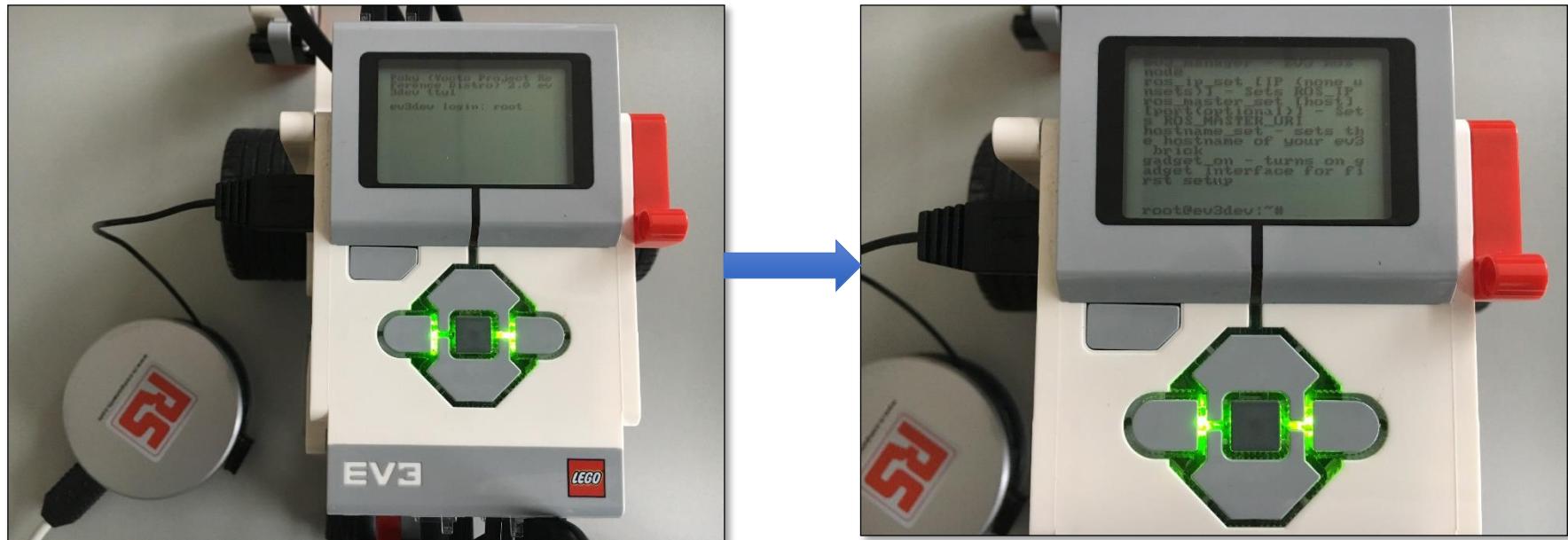
Second: Network Setup in EV3 Side

3. Some seconds after starting you should see the H4R boot logo. After some time the yellow blinking LEDs should end and lighting up in a constant green.



Second: Network Setup in EV3 Side

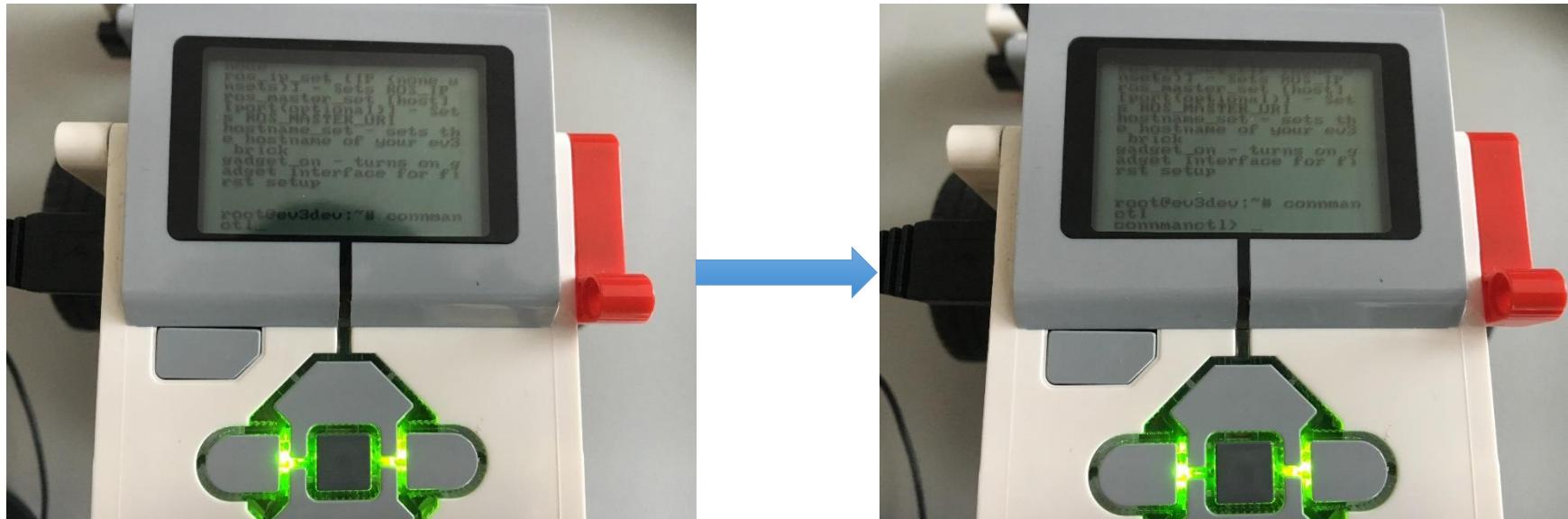
4. Now use the keyboard to enter the login password: **root** and press enter



Second: Network Setup in EV3 Side

- Now we need to configure our wifi network. To do that, write connmanctl which should give you a connman prompt to configure our wifi network.

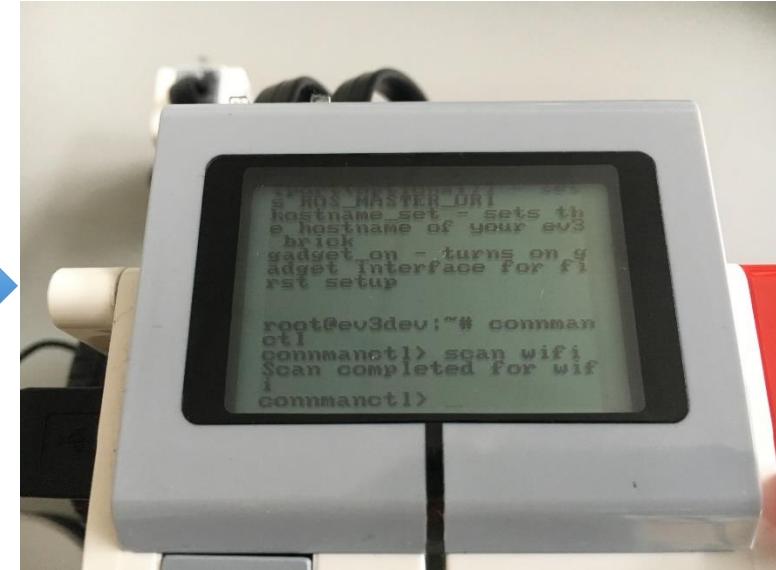
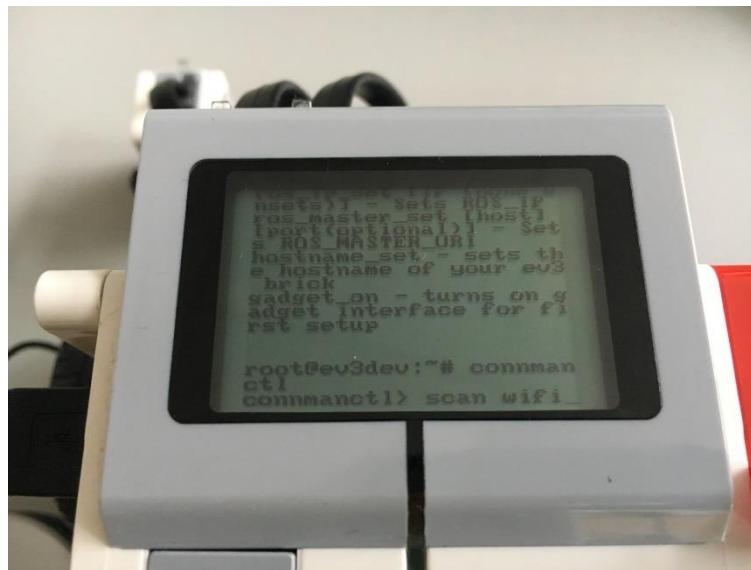
```
robot@ev3dev:~# connmanctl
```



Second: Network Setup in EV3 Side

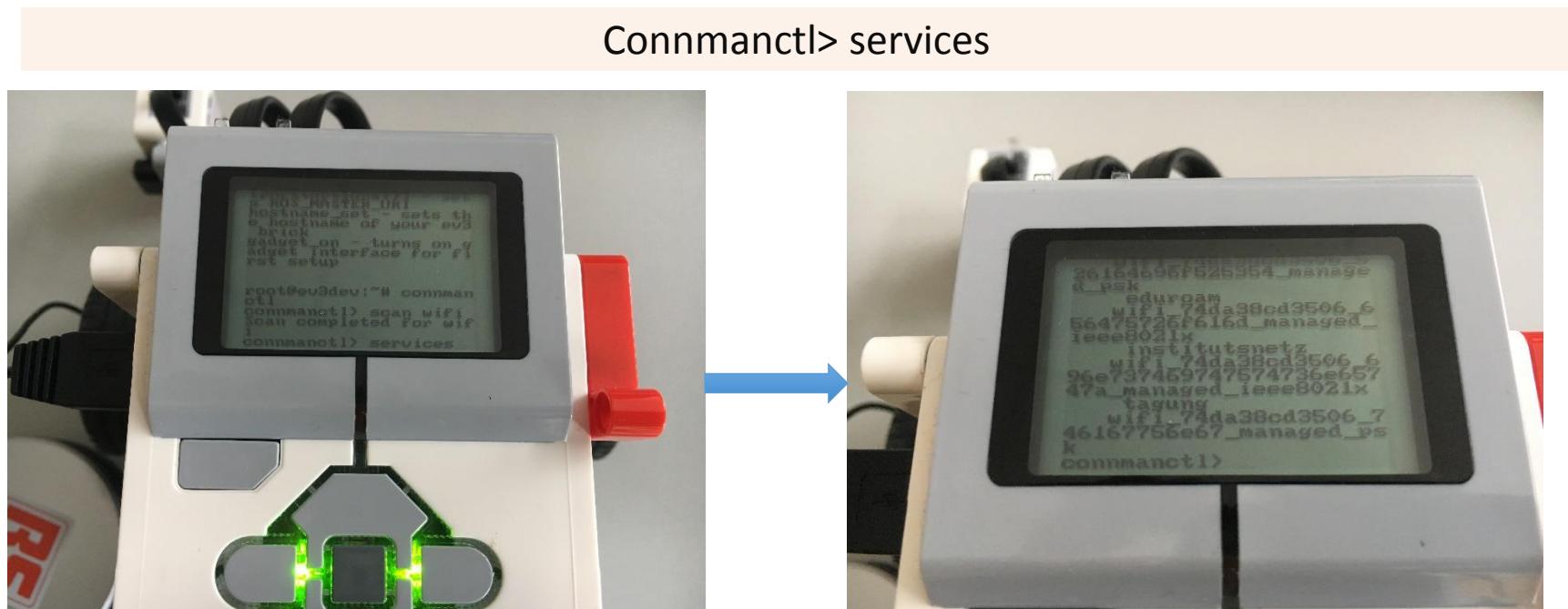
- Now we need to Scan for access points (i.e. available wifi networks):

Connmanctl> scan wifi



Second: Network Setup in EV3 Side

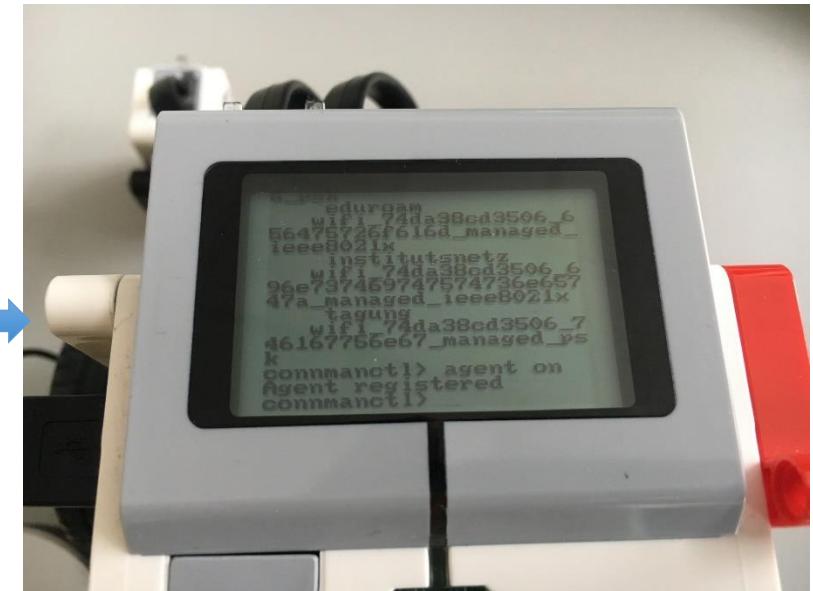
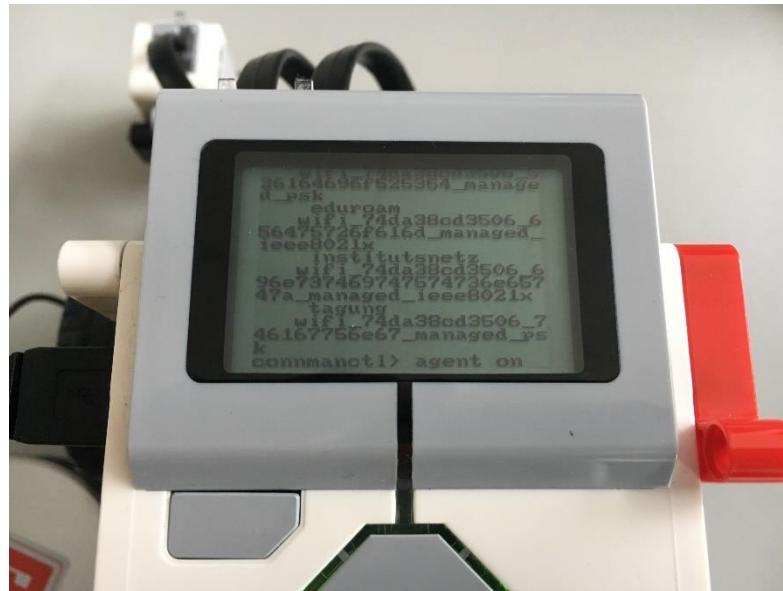
7. Then we need to display all discovered wifi networks. Take note of the network name, for example our wifi network is tagung and has a name "[wifi_74da38cd3506_746167756e67_managed_psk](#)"



Second: Network Setup in EV3 Side

8. Then we need to register the agent to handle user requests. So we write agent on and press enter.

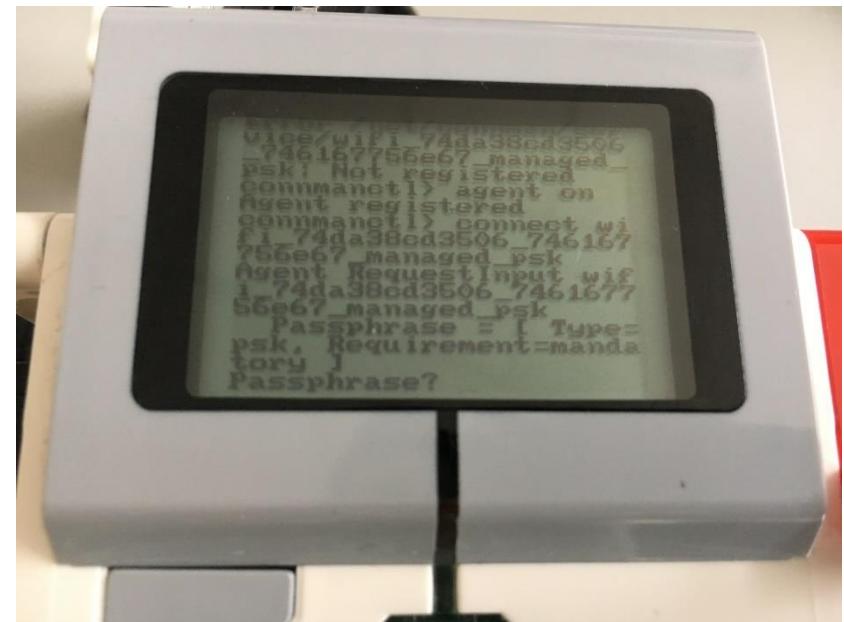
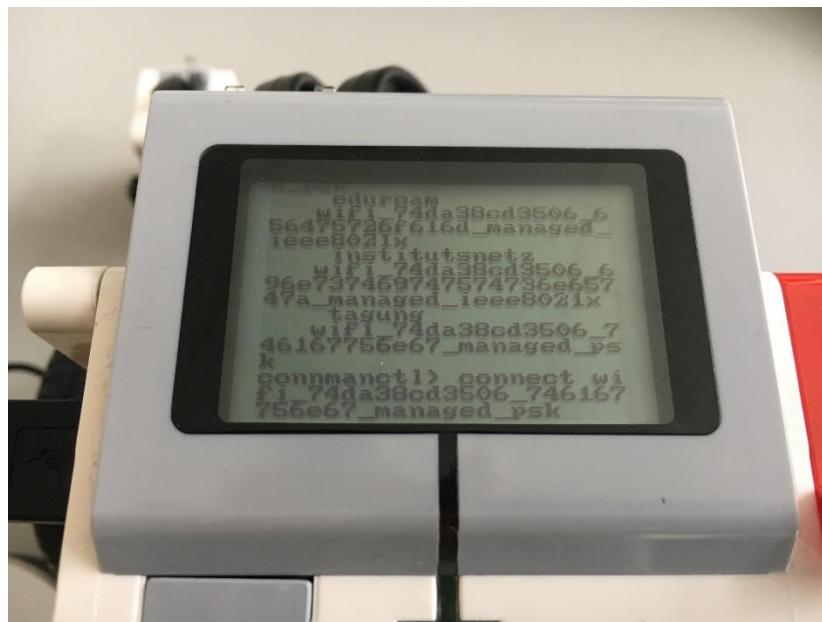
Connmanctl> agent on



Second: Network Setup in EV3 Side

9. Connect to the tagung network using connect command and network name.
Then enter the wifi password after “passphrase?”.

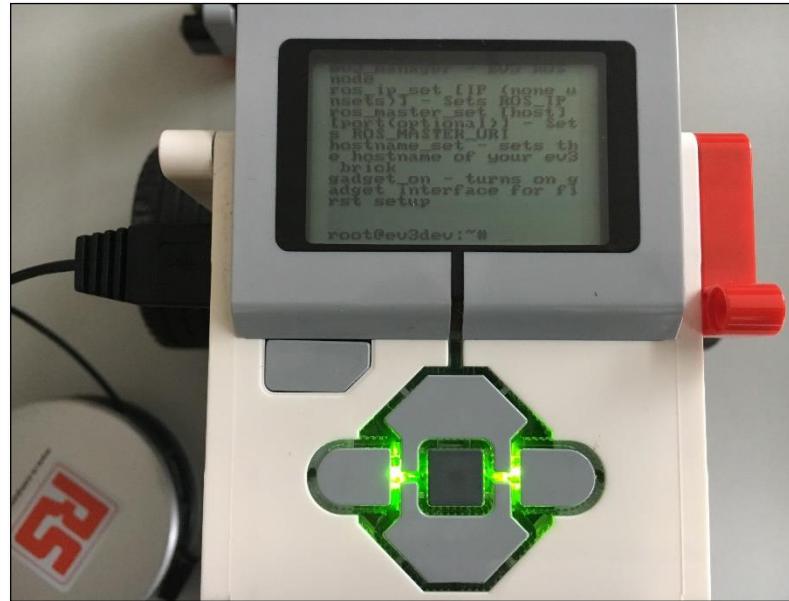
```
Connmanctl> connect wifi_74da38cd3506_746167756e67_managed_psk
```



Second: Network Setup in EV3 Side

10. After we connect to the wifi network, we can now exit the Connmanctl by writing exit in order to go back to the main screen again.

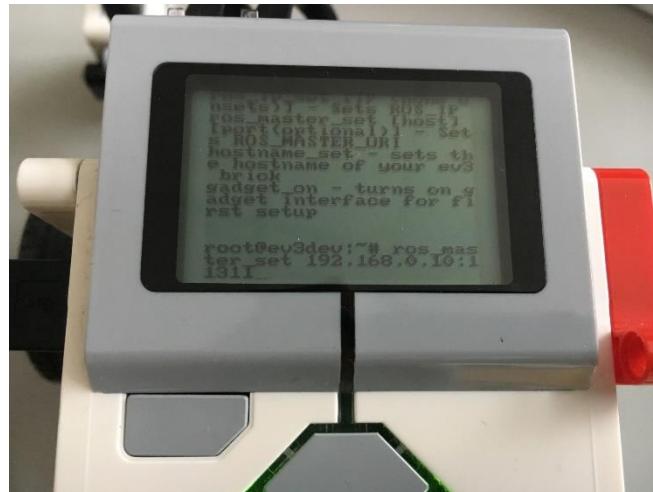
Connmanctl> exit



Second: Network Setup in EV3 Side

11. Now we need to adjust ROS environment variables one more time. Firstly we need to tell the EV3 robot that our ROS master will run on our PC which has the IP address 192.168.0.10. Use the keyboard to enter the following line to adjust the ROS_MASTER_URI variable.

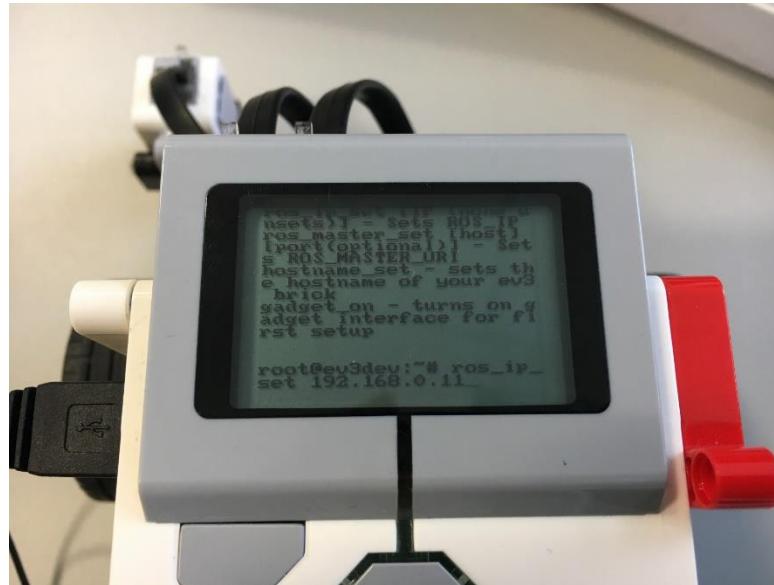
```
ros_master_set 192.168.0.10:11311
```



Second: Network Setup in EV3 Side

12. Lastly we need to identify the ROS_IP environment variable to force ROS on the EV3 to use our EV3 IP address (i.e. [192.168.0.11](#)) for all nodes generated on the EV3.

ros_ip_set [192.168.0.11](#)



EV3 Reboot and Shutdown Commands

1. Reboot command

```
reboot
```

2. Shutting down

```
Shutdown -h now
```

Tutorial#5

ROS ⇔ EV3 Communication over Network



ROS ⇔ EV3 Communication over Network

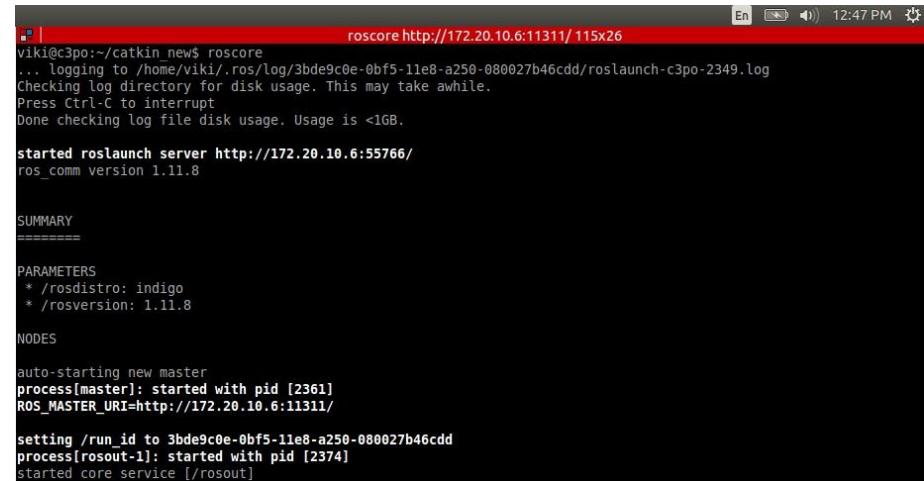
Once we finish our network setup, now we can establish the communication between the EV3 and ROS.

1. In a new terminal on your PC change to the catkin workspace.

```
$ cd ~/catkin_ws
```

2. Open another new terminal on your PC and run roscore

```
$ roscore
```



```
vikki@c3po:~/catkin_new$ roscore
... logging to /home/vikki/.ros/log/3bde9c0e-0bf5-11e8-a250-080027b46cdd/roslaunch-c3po-2349.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://172.20.10.6:11311/
ros_comm version 1.11.8

SUMMARY
=====

PARAMETERS
* /rosdistro: indigo
* /rosversion: 1.11.8

NODES

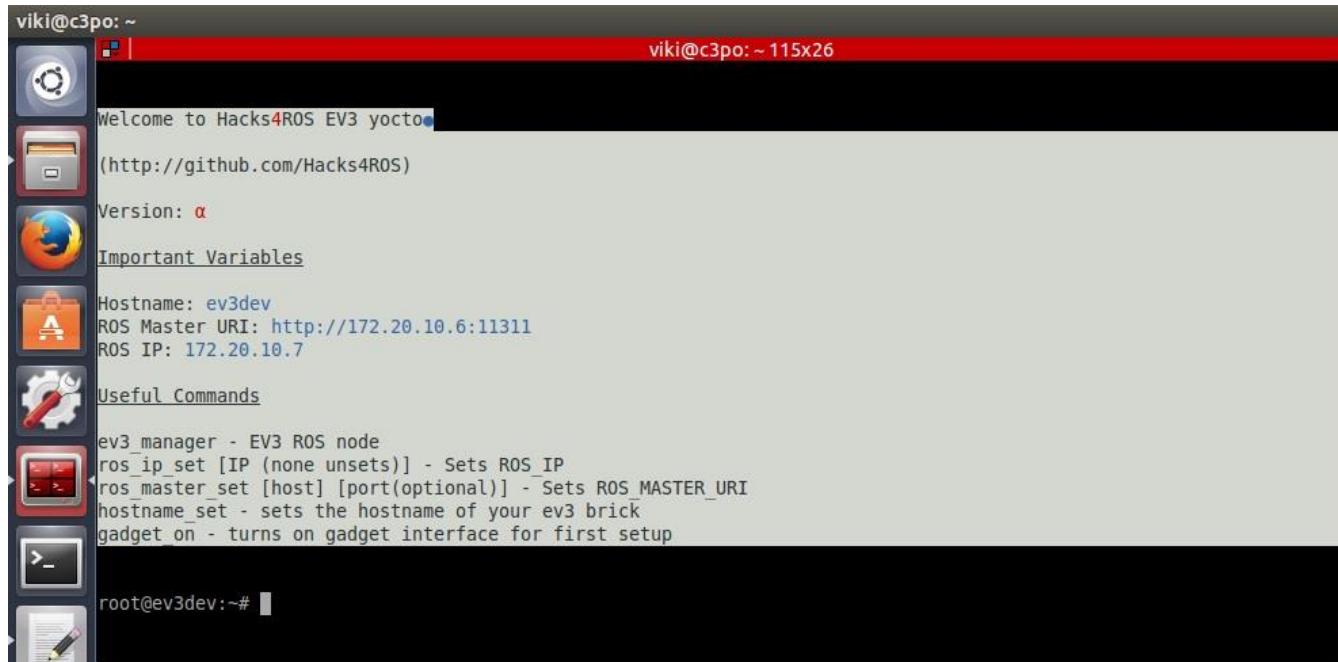
auto-starting new master
process[master]: started with pid [2361]
ROS_MASTER_URI=http://172.20.10.6:11311/

setting /run_id to 3bde9c0e-0bf5-11e8-a250-080027b46cdd
process[rosout-1]: started with pid [2374]
started core service [/rosout]
```

ROS ⇔ EV3 Communication over Network

3. In order to connect to our EV3, we need to connect via ssh by executing the following in another new terminal:

```
$ ssh root@192.168.0.11
```



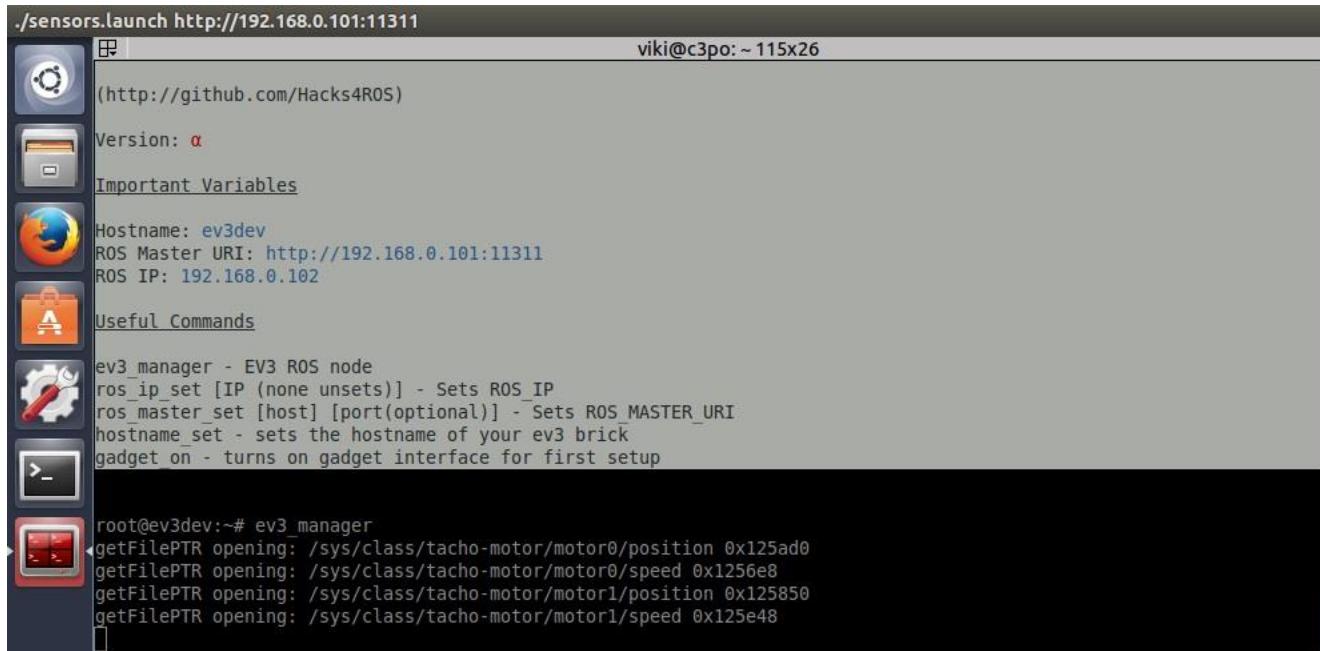
The screenshot shows a Linux desktop environment with a terminal window open. The terminal window title is "viki@c3po: ~". The window content displays the following text:

```
viki@c3po: ~
viki@c3po: ~ 115x26
Welcome to Hacks4ROS EV3 yocto
(http://github.com/Hacks4ROS)
Version: α
Important Variables
Hostname: ev3dev
ROS Master URI: http://172.20.10.6:11311
ROS IP: 172.20.10.7
Useful Commands
ev3 manager - EV3 ROS node
ros_ip_set [IP (none unsets)] - Sets ROS_IP
ros_master_set [host] [port(optional)] - Sets ROS_MASTER_URI
hostname_set - sets the hostname of your ev3 brick
gadget_on - turns on gadget interface for first setup
root@ev3dev:~#
```

ROS ⇔ EV3 Communication over Network

- Once we are connected to our EV3 through the secured shell, write on the ssh the following and press enter

```
$ ev3_manager
```



The screenshot shows a terminal window with the following content:

```
./sensors.launch http://192.168.0.101:11311
viki@c3po: ~ 115x26

(http://github.com/Hacks4ROS)

Version: α

Important Variables

Hostname: ev3dev
ROS Master URI: http://192.168.0.101:11311
ROS IP: 192.168.0.102

Useful Commands

ev3_manager - EV3 ROS node
ros_ip_set [IP (none unsets)] - Sets ROS_IP
ros_master_set [host] [port(optional)] - Sets ROS_MASTER_URI
hostname_set - sets the hostname of your ev3 brick
gadget on - turns on gadget interface for first setup

root@ev3dev:~# ev3_manager
getFilePTR opening: /sys/class/tacho-motor/motor0/position 0x125ad0
getFilePTR opening: /sys/class/tacho-motor/motor0/speed 0x125e8
getFilePTR opening: /sys/class/tacho-motor/motor1/position 0x125850
getFilePTR opening: /sys/class/tacho-motor/motor1/speed 0x125e48
```

ROS ⇔ EV3 Communication over Network

- Now we can launch our motors controllers, by executing the following in a new terminal

```
$ roslaunch ./motors.launch ev3_hostname:=ev3dev
```

```
./motors.launch http://172.20.10.6:11311 115x25
* /rosdistro: indigo
* /rosversion: 1.11.8

NODES
/ev3dev/
  controller_spawner (controller_manager/spawner)
    ev3_ctrl_node (h4r_ev3_manager/h4r_ev3_manager)

ROS_MASTER_URI=http://172.20.10.6:11311

core service [/rosout] found
ERROR: cannot launch node of type [h4r_ev3_manager/h4r_ev3_manager]: h4r_ev3_manager
ROS path [0]=/opt/ros/indigo/share/ros
ROS path [1]=/opt/ros/indigo/share
ROS path [2]=/opt/ros/indigo/stacks

process[ev3dev/controller_spawner-2]: started with pid [6729]
[INFO] [WallTime: 1518029346.130426] Controller Spawner: Waiting for service controller_manager/load_controller
[INFO] [WallTime: 1518029346.921815] Controller Spawner: Waiting for service controller_manager/swing_controller
[INFO] [WallTime: 1518029346.931092] Controller Spawner: Waiting for service controller_manager/unload_controller
[INFO] [WallTime: 1518029346.938875] Loading controller: OutPortState
[INFO] [WallTime: 1518029354.151706] Loading controller: OutPortA
[INFO] [WallTime: 1518029367.242621] Loading controller: OutPortB
[INFO] [WallTime: 1518029367.540716] Controller Spawner: Loaded controllers: OutPortState, OutPortA, OutPortB
[INFO] [WallTime: 1518029368.241784] Started controllers: OutPortState, OutPortA, OutPortB
```

["Notice here also, the changes in the secured shell"](#)

```
./motors.launch http://172.20.10.6:11311
viki@c3po: ~ 115x25
 getFileTR opening: /sys/class/tacho-motor/motor1/position 0x105118
 getFilePTR opening: /sys/class/tacho-motor/motor1/speed 0x628e8
 [INFO] [1518029367.650275432]: Controller Change
 [INFO] [1518029367.683892645]: velocity controllers/JointVelocityController requests Joint OutPortA Joint_A
 getFileTR opening: /sys/class/tacho-motor/motor0/command 0xb32c1690
 [INFO] [1518029367.779296986]: Joint control mode: velocity
 [INFO] [1518029367.8509093748]: P: 0 I: 0 D: 0
 [INFO] [1518029367.870607659]: <-----EV3 Joint Joint A----->
 [INFO] [1518029367.891409106]: <-----EV3 Joint Joint A----->
 [INFO] [1518029367.914310967]: PID 0 0 0
 getFileTR opening: /sys/class/tacho-motor/motor0/speed pid/Kp 0xb32c17f8
 getFileTR opening: /sys/class/tacho-motor/motor0/speed pid/Ki 0xb32c1b8
 getFileTR opening: /sys/class/tacho-motor/motor0/speed pid/Kd 0xb32c1cd0
 [INFO] [1518029367.958963677]: velocity controllers/JointVelocityController requests Joint OutPortB Joint_B
 getFileTR opening: /sys/class/tacho-motor/motor1/command 0xb32c1e38
 [INFO] [1518029368.012434987]: <-----EV3 Joint Joint B----->
 [INFO] [1518029368.059909033]: Joint control mode: velocity
 [INFO] [1518029368.121655965]: P: 0 I: 0 D: 0
 [INFO] [1518029368.138048868]: <-----EV3 Joint Joint B----->
 [INFO] [1518029368.151153051]: <-----EV3 Joint Joint B----->
 [INFO] [1518029368.167968869]: PID 0 0 0
 getFileTR opening: /sys/class/tacho-motor/motor1/speed pid/Kp 0xb32c1fa0
 getFileTR opening: /sys/class/tacho-motor/motor1/speed pid/Ki 0xb32c2108
 getFileTR opening: /sys/class/tacho-motor/motor1/speed pid/Kd 0xb32c2270
```

ROS ⇔ EV3 Communication over Network

- Now we can launch our sensors as well, by executing the following in a new terminal

```
$ rosrun ./sensors.launch ev3_hostname:=ev3dev
```

```
/sensors.launch http://172.20.10.6:11311 115x25
* /ev3dev/Ev3UltraSonic	mode: distance
* /ev3dev/Ev3UltraSonic/port: in2
* /ev3dev/Ev3UltraSonic/publish rate: 10
* /ev3dev/Ev3UltraSonic/topic name: /ultrasonic
* /ev3dev/Ev3UltraSonic/type: h4r_ev3_control/E...
* /rosdistro: indigo
* /rosversion: 1.11.8

NODES
/ev3dev/
  ev3_sensor_spawner (controller_manager/spawner)

ROS_MASTER_URI=http://172.20.10.6:11311

core service [/rosout] found
process[ev3dev/ev3_sensor_spawner-1]: started with pid [6840]
[INFO] [Walltime: 1518029418.669479] Controller Spawner: Waiting for service controller_manager/load_controller
[INFO] [Walltime: 1518029418.649588] Controller Spawner: Waiting for service controller_manager/switch_controller
[INFO] [Walltime: 1518029418.657190] Controller Spawner: Waiting for service controller_manager/unload_controller
[INFO] [Walltime: 1518029418.666407] Loading controller: Ev3Color
[INFO] [Walltime: 1518029425.265850] Loading controller: Ev3UltraSonic
[INFO] [Walltime: 1518029425.670467] Loading controller: Ev3Gyro
[INFO] [Walltime: 1518029425.992653] Controller Spawner: Loaded controllers: Ev3Color, Ev3UltraSonic, Ev3Gyro
[INFO] [WallTime: 1518029426.086328] Started controllers: Ev3Color, Ev3UltraSonic, Ev3Gyro
```

“Notice here also, the changes in the secured shell”

```
s.launch http://172.20.10.6:11311
viki@c3po: ~ 115x26
[INFO] [1518029426.155531984] Controller Change
getFilePTR opening: /sys/class/lego-sensor/sensor0/mode 0xb5409aa8
getFilePTR opening: /sys/class/lego-sensor/sensor1/num_values 0x62a50
getFilePTR opening: /sys/class/lego-sensor/sensor1/value0 0x62bbb8
getFilePTR opening: /sys/class/lego-sensor/sensor2/mode 0x105a80
getFilePTR opening: /sys/class/lego-sensor/sensor2/num_values 0x105be8
getFilePTR opening: /sys/class/lego-sensor/sensor2/value0 0x105d50
getFilePTR opening: /sys/class/lego-sensor/sensor0/mode 0x105eb8
getFilePTR opening: /sys/class/lego-sensor/sensor0/num_values 0x106020
getFilePTR opening: /sys/class/lego-sensor/sensor0/value0 0x106188
```

ROS ⇔ EV3 Communication over Network

7. Now we are ready to run our programs in a new terminal.

Tutorial#6: Lego Mindstorms Ev3 Sensors



Lego Mindstorms Ev3 Sensors

Ev3 Color Sensor: The EV3 controller is used to read the standard EV3 Color Sensor. The mode parameter sets the mode for the color sensor. If not set, it will use **rgb_raw**. The following modes are possible for the sensor:

1. **rgb_raw:** 'rgb_raw' mode uses the RGB_RAW mode of this sensor. It will output the color information in RGB in a **std_msgs::ColorRGBA.msg** topic.
2. **Ambient:** 'ambient' mode measures the ambient light and publishes into a **sensor_msgs::Illuminance** topic
3. **Reflect:** 'reflect' mode measures the reflected light from an obstacle and publishes into a **sensor_msgs::Illuminance** topic



Lego Mindstorms Ev3 Sensors

4. **Color:** 'color' mode publishes the a number for the following recognized colors into a `std_msgs::UInt8` topic

Value	Color
0	none
1	black
2	blue
3	green
4	yellow
5	red
6	white
7	brown

Lego Mindstorms Ev3 Sensors



EV3 Ultrasonic Sensor: The Ev3 Ultrasonic Controller is used to read the standard EV3 Ultrasonic Sensor. The sensor has two modes,

1. **Distance:** publishes the measured distance into a **sensor_msgs::Range** topic.
2. **Listen:** publishes into a **std_msgs::Boolean** topic, when it detects another Ultrasonic Sensor or a loud noise (like a clap) the value is true otherwise false.
 - **max_range** sets the maximum range of values taken into account if a value is bigger it will be replaced by infinity. The maximum value for this parameter is 2.5 because the value 2.550 is the sensors error condition.
 - **min_range** sets the minimum range of values taken into account if a value is smaller it will be replaced by negative infinity.

Lego Mindstorms Ev3 Sensors

The Ev3 Gyro Sensor: The Ev3 Gyro Controller is used to read the standard EV3 Gyro Sensor. The mode parameter sets the mode for the gyro sensor. If not set, it will use **angle**. The sensor has three modes,

1. **Rate:** publishes the turn rate into a **sensor_msgs::Imu** topic (z-axis).
2. **Angle:** publishes the turn rate into a **sensor_msgs::Imu** topic (z-axis). Output values are in radian which has range in degrees (0 to 180° and 0 to -180°).
3. **rate&angle:** publishes both turn and angle into a **sensor_msgs::Imu** topic (z-axis).



References

1. Quigley, Morgan, et al. "ROS: an open-source Robot Operating System." ICRA workshop on open source software. Vol. 3. No. 3.2. 2009.
2. Quigley, M., Gerkey, B., & Smart, W. D. (2015). Programming Robots with ROS: a practical introduction to the Robot Operating System. " O'Reilly Media, Inc."
3. <https://www.automation.com/library/articles-white-papers/robotics/robotics-software-platforms-review>
4. <http://www.ros.org/history/>
5. http://www.modulabs.co.kr/board_GDCH80/1562
6. <http://wiki.ros.org/Distributions>
7. <http://wiki.ros.org/kinetic/Installation/Ubuntu>

Questions

Thank you!

