Robot Operating System (ROS)

Lab 4: ROS launch, multiple machines, and ROSSerial

Haitham El-Hussieny, PhD

November 14, 2022

Department of Mechatronics and Robotics Engineering
Egypt-Japan University of Science and Technology (E-JUST)
Alexandria, Egypt.

OUTLINE

1. ROS Launch Files.
2. ROS Network Configuration.
3. ROSserial interface with Arduino.

ROS Launch Files.

ROS LAUNCH FILES.

- ROS can run hundreds of nodes simultaneously.

ROS Launch Files.

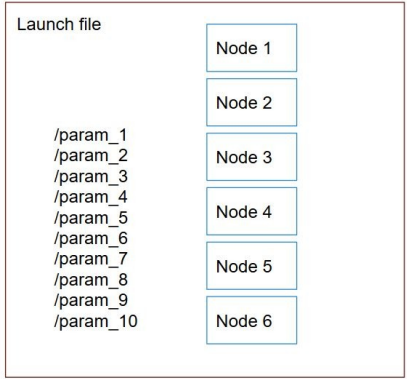
- ROS can run hundreds of nodes simultaneously.
- rosrun large number of ROS nodes each time could be **cumbersome**.

ROS LAUNCH FILES.

- ROS can run hundreds of nodes simultaneously.
- rosrun large number of ROS nodes each time could be **cumbersome**.

roslaunch is a tool for easily launching **multiple ROS nodes**, as well as setting **parameters** on the Parameter Server.

ROS parameters could be useful for setting options or configurations for your ROS nodes before running them.



ROS LAUNCH FILES.

- A launch file has an extension **.launch** and is located in a **launch** folder inside your package.
- The minimal launch file starts with the the tags <launch> < /launch>:

```
<launch>
    <node name="turtlesim_sim" pkg="turtlesim" type="turtlesim_node" />
    <node name="turtlesim_key" pkg="turtlesim" type="turtlesim_teleop_key" />
</launch>
```

in the node tag:

- **type:** the name of the node you need to run.
- **pkg:** the name of the package contains that node.
- **name:** a unique name chosen for the node.

ROS LAUNCH FILES.

in terminal:

```
~$ mkdir catkin_ws/src/my_first_package/launch
~$ gedit catkin_ws/src/my_first_package/launch/turtle_move.launch
```

```
<launch>
    <node name="turtlesim_sim" pkg="turtlesim" type="turtlesim_node" />
    <node name="turtlesim_key" pkg="turtlesim" type="turtle_teleop_key" />
</launch>
```

in terminal to run the launch file:

```
~$ roslaunch my_first_package turtle_move.launch
```

ROS Launch Files.

Output node's logs on screen:

```
<launch>
  <node name="turtlesim_sim" pkg="turtlesim" type="turtlesim_node" output="screen" />
  <node name="turtlesim_key" pkg="turtlesim" type="turtle_teleop_key" />
</launch>
```

output="screen" option allows to output the logs of this node on the screen.

ROS LAUNCH FILES.

Use parameters with launch file:

Sometimes, parameters within a launch file could be used as options for the node.

```
<launch>
  <param name="kp" value="0.4"/>
  <param name="kr" value="2.0"/>

  <node name="turtlesim" pkg="turtlesim" type="turtlesim_node" />
  <node name="server" pkg="control_turtle" type="turtle_move_server.py" />
  <node name="client" pkg="control_turtle" type="turtle_move_client.py" output="screen" />
</launch>
```

Example of setting control gains

In your node code:

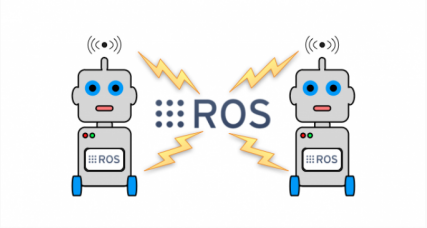
```
if __name__=="__main__":
    try:
        global kp, kr
        kp = rospy.get_param("kp")
        kr = rospy.get_param("kr")
        move_turtle_server()
```

ROS Network Configuration.

ROS Network Configuration.

ROS allows you to run nodes on a single robot and on **dozens robots as well**, as long, as your devices are in the **same network**.

- To run ROS on multiple machines you need to connect them to the same LAN network at first.
- Remember that only one devices can run ROS Master.



ROS NETWORK CONFIGURATION.

Configuring two ROS machines:

Master side (with roscore)

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

```
# Add these two lines
export ROS_MASTER_URI=http://192.168.1.7:11311
export ROS_IP=192.168.1.7
```

192.168.1.7 is the IP of this master machine.

To know your IP:

```
$ ifconfig
```

Slave side

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

```
# Add these two lines
export ROS_MASTER_URI=http://192.168.1.7:11311
export ROS_IP=192.168.1.8
```

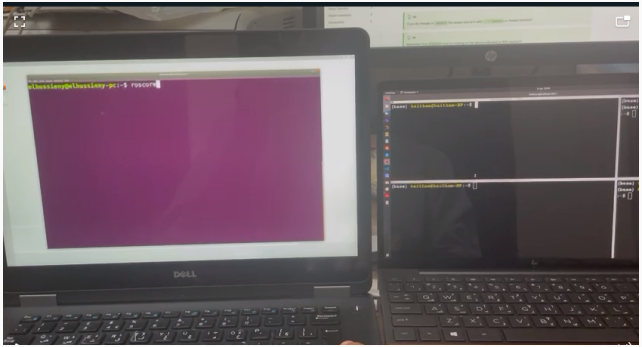
192.168.1.7 is the IP of master machine.
192.168.1.8 is the IP of this slave machine.

To know your IP:

```
$ ifconfig
```

ROS NETWORK CONFIGURATION.

Testing the communication: two PCs configured:



roscore + turtlesim node on master PC
and tele-operation node on the slave PC

For interested: Connecting ROS powered robot over the Internet.

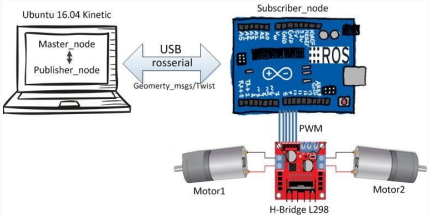
ROSserial interface with Arduino.

ROSSERIAL INTERFACE WITH ARDUINO.

What is ROS Serial?

If you have a sensor or an actuator and you need to add it to your ROS system:

- It could have it's own ROS interface (e.g. MS Kinect)
- It could be connected to Arduino and with ROS serial package you can interface it.



ROSSERIAL INTERFACE WITH ARDUINO.

Why ROS Serial?

- A communication protocol between ROS and new hardware.
- Integration of micro-controllers and embedded systems into ROS.
- Adding new embedded hardware (sensors, actuators, ...).

ROSSERIAL INTERFACE WITH ARDUINO.

Clients supported by ROS Serial.

rosserial_arduino	support for Arduino compatible boards including UNO, Leonardo, MEGA, DUE, Teensy 3.x and LC, Spark, STM32F1, STM32Duino, ESP8266 and ESP32
rosserial_embeddedlinux	support for Embedded Linux (eg, routers)
rosserial_windows	support for communicating with Windows applications
rosserial_mbed	support for mbed platforms
rosserial_tivac	support for TI's Launchpad boards, TM4C123GXL and TM4C1294XL
rosserial_vex_v5	support for VEX V5 Robot Brain
rosserial_vex_cortex	support for VEX Cortex board
rosserial_stm32	support for STM32 MCUs, based on STM32CubeMX HAL
ros-teensy	support for teensy platforms

ROSSERIAL INTERFACE WITH ARDUINO.

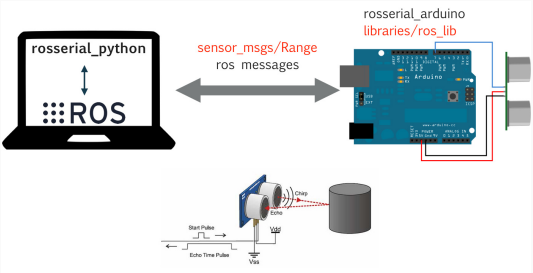
ROS-side Interface.

option 1: rosserial_python

A Python-based implementation recommended for PC side.

option 2: rosserial_server

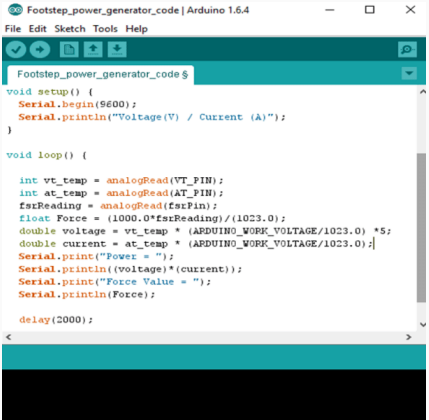
A C++ implementation, has some limitations compared to rosserial_python but recommended for high-performance applications.



ROSSERIAL INTERFACE WITH ARDUINO.

Installation of ROSserial Libraries.

Step 1: Install Arduino IDE.



ROSSERIAL INTERFACE WITH ARDUINO.

Installation of ROSserial Libraries.

Step 2: Install ROSserial package

```
$ sudo apt-get install ros-noetic-rosserial-arduino
$ sudo apt-get install ros-noetic-rosserial
```

Step 3: Go to the Arduino **libraries** folder and make sure there is no ros_lib folder

```
(base) haitham@mydevice:~/arduino-1.8.19/libraries$ ls
Adafruit_Circuit_Playground  LiquidCrystal   SpacebrewYun
Bridge                       Mouse           Stepper
Esplora                      Robot_Control   Temboo
Ethernet                     RobotIRremote   TFT
Firmata                      Robot_Motor     WiFi
GSM                          SD
Keyboard                     Servo
```

ROSSERIAL INTERFACE WITH ARDUINO.

Installation of ROSserial Libraries.

Step 4: Run the roscore command

```
$ roscore
```

Step 5: in the arduin libraries folder run the ROS node:

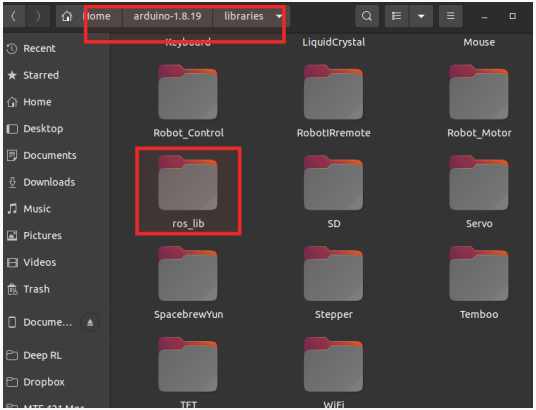
```
$ cd arduino-1.8.19/libraries/
$ rosrun rosserial_arduino make_libraries.py .
```

The sign **.** means generate the libraries in the current location.

ROSSERIAL INTERFACE WITH ARDUINO.

Installation of ROSserial Libraries.

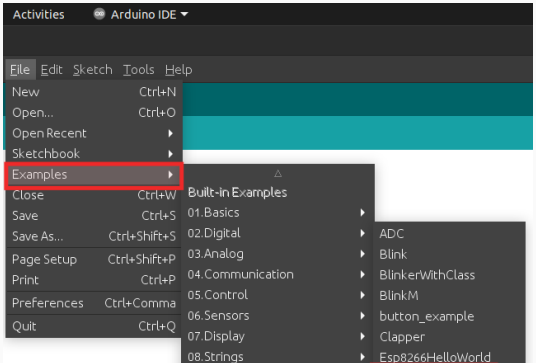
Make sure that the ros_lib folder is added in the libraries folder.

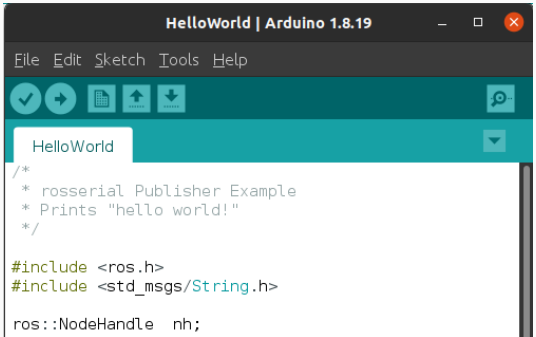


ROSSERIAL INTERFACE WITH ARDUINO.

Installation of ROSserial Libraries.

Step 6: Open the Arduino IDE and choose Examples>ros_lib>HelloWorld





ROSSERIAL INTERFACE WITH ARDUINO.

ROS Publisher in Arduino

```
#include <ros.h>
#include <std_msgs/String.h>
ros::NodeHandle nh; // Create a node handler
std_msgs::String str_msg; // Create the ROS message
ros::Publisher chatter("chatter", &str_msg); // Create the ROS publisher with topic "chatter"
char hello[13] = "hello world!"; // The message to be sent
void setup()
{
    nh.initNode(); // initialize the ROS node
    nh.advertise(chatter); // setup a ROS publisher
}

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

ROSSERIAL INTERFACE WITH ARDUINO.

ROS Publisher in Arduino

Step 7: Compile and upload the program to the Arduino board.

Step 8: To have the communication between the Arduino and ROS system we need to run the ROS serial server. So run the rosserial node and specify the Arduino port:

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

Check with rostopic echo:

```
haitham@mydevice:~$ rostopic echo /chatter
data: "hello world!"
---
data: "hello world!"
---
data: "hello world!"
---
data: "hello world!"
```

ROSSERIAL INTERFACE WITH ARDUINO.

ROS Subscriber in Arduino (Toggle LED)

```
#include <ros.h>
#include <std_msgs/Empty.h>
ros::NodeHandle nh; // Define the node handler
void messageCb(const std_msgs::Empty& toggle_msg){ // Subscriber callback
    digitalWrite (LED_BUILTIN, HIGH-digitalRead(LED_BUILTIN)); // blink the led
}
ros::Subscriber<std_msgs::Empty> sub("toggle_led", &messageCb); // Setup a subscriber that will recieve Empty message

void setup()
{
    pinMode(LED_BUILTIN, OUTPUT); // Output LED
    nh.initNode(); // Initialize the ROS node
    nh.subscribe(sub); // Setup the subscriber
}
void loop()
{
    nh.spinOnce();
    delay(1);
}
```

ROSSERIAL INTERFACE WITH ARDUINO.

ROS Subscriber in Arduino (Toggle LED)

Compile and upload the program to the Arduino board.

Run the rosserial node and specify the Arduino port:

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

Check with rostopic pub:

```
$ rostopic pub /toggle_led std_msgs/Empty "{}" -r 10
```

-r means publish continuously at a rate of 10Hz

End of Lecture

haitham.elhussieny@ejust.edu.eg