

## ROS Tutorials

By, Eng. Mina G. Sadek

# Slides Outline

---

- ▶ [Why ROS](#)
- ▶ [What is ROS](#)
  - ▶ ROS architecture
  - ▶ ROS Nodes, Messages, Topics, Master, Services, Actions, Bags.
- ▶ Catkin Package:
  - ▶ [Creating catkin workspace](#)
  - ▶ [Creating & Building Package](#)
  - ▶ Creating node [Publisher]
    - [\[C++ Publisher\]](#)
    - [\[Python Publisher\]](#)
  - ▶ Creating node [Subscriber]
    - [\[C++ Subscriber\]](#)
    - [\[Python Subscriber\]](#)
- ▶ [ROS Launch](#)
- ▶ [Message passing between multiple machines](#)



# Slides Outline

---

- ▶ GAZEBO
  - ▶ Building and controlling a robot in GAZEBO
    - ▶ 1. Pre-Built Robots
      - TIAGo robot and environment
      - HUSKY robot and environment
    - ▶ 2. Natively building robot in GAZEBO:
      - Adding a Robot to GAZEBO
      - Driving robot on GAZEBO using a node
  - ▶ Installing ROS indigo/kinetic/melodic on Linux UBUNTU Desktop:
    - ▶ ROS melodic for UBUNTU 18.04: <http://wiki.ros.org/melodic/Installation/Ubuntu>
    - ▶ ROS kinetic for UBUNTU 16.04: <http://wiki.ros.org/kinetic/Installation/Ubuntu>
    - ▶ ROS indigo for UBUNTU 14.04: <http://wiki.ros.org/indigo/Installation/Ubuntu>



# Slides Outline

---

- ▶ Real-Time Robot using RPi
  - ▶ A. Installing Raspbian and ROS on the RPi
    - ▶ [\[METHOD\\_I\]: Using pre-built image of Raspbian image + ROS](#)
    - ▶ [METHOD\_II]: Native installation
      - ☐ 1. Installing Raspbian Kinetic on RPi
      - ☐ 2. Installing ROS on RPi
  - ▶ [C. Installing wiringPi on RPi](#)
  - ▶ D. Controlling RPi's GPIO pins from ROS and wiringPi
  - ▶ E. ROS Communication with Arduino [ROS Serial]
- ▶ Useful ROS Commands

# Why ROS

---

- ▶ **Open-source** robotic software.
- ▶ Free for private and commercial use
- ▶ Has many built-in software packages [reusable code packages] essential in the robotic applications.
- ▶ Communications Infrastructure
- ▶ **Message Passing**
- ▶ Facilitate the communication required between processes even if they are on different machines.
- ▶ **Recording & Playback of Messages**
- ▶ **Remote Procedure Calls**
- ▶ **Distributed Parameter System**

# Why ROS

---

- ▶ Community support.
- ▶ The Robot Operating System [ROS] is a flexible framework for writing robot software. It is a collection of tools, libraries and conventions that aim to simplify the task of creating complex and robust robot behavior across a wide variety of robotic platforms.

# Why ROS

---

- ▶ Many other software technologies can be integrated with ROS, including:
  - ▶ MoveIt
    - ▶ Open-source software for performing motion planning
  - ▶ GAZEBO
    - ▶ High quality physics engine, that performs visual simulation
  - ▶ OpenCV
    - ▶ Open-source library supporting Computer-Vision



# What is ROS

---

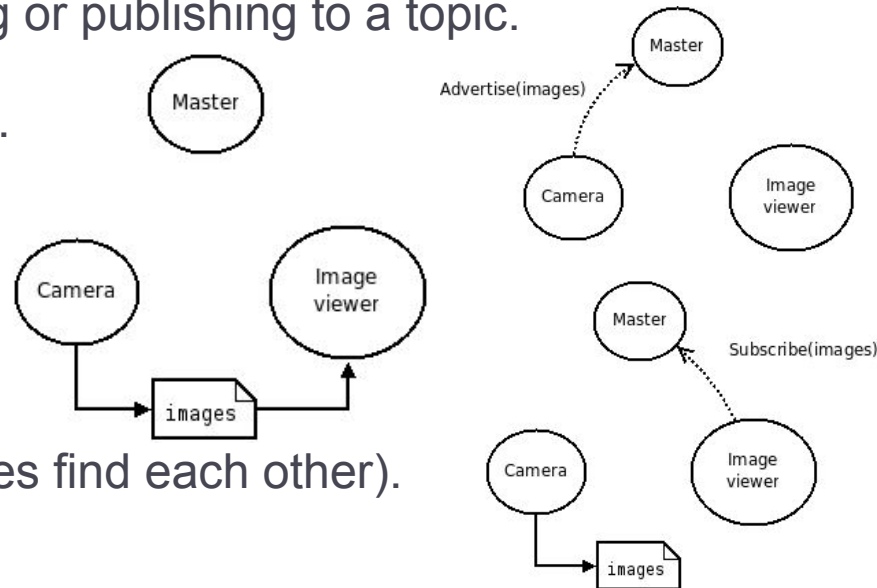
- ▶ Robot Operating System [ROS]
- ▶ Peer to Peer
  - ▶ Individual programs communicate over defined API (messages, services, etc.)
- ▶ Distributed
  - ▶ Programs can be run on multiple computers and communicate over the network.
- ▶ Programming language Independent
  - ▶ Client libraries exist for C++, Python, MATLAB, Java.
- ▶ Free and Open-Source
  - ▶ Most ROS software is open-source and free to use.





# What is ROS

- ▶ **ROS Nodes:**
  - ▶ An executable that uses ROS to communicate with other nodes.
- ▶ **ROS Messages:**
  - ▶ ROS data type used when subscribing or publishing to a topic.
  - ▶ Coded as a .msg simple text file, describing fields of the ROS message.
- ▶ **ROS Topics:**
  - ▶ Nodes can publish messages to a topic as well as subscribe to a topic to receive messages.
- ▶ **ROS Master Node:**
  - ▶ Name service for ROS (i.e. helps nodes find each other).
  - ▶ Runs using the command “roscore”.



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

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

<https://wiki.ros.org/Master>

# What is ROS

---

## ► ROS Services:

- Topics are many-to-many way of communication. This technique is not appropriate for RPC [Remote Procedure Call] Request/Reply interactions.
- Request/Reply is done via a SERVICE.
- A Service is defined by a pair of messages:
  - A message for the Request
  - A message for the Reply
- Its technique:
  - ROS node providing the service [Server Node] offers a service under a string name
  - Client ROS node calls the service by sending the request message and awaiting the reply.

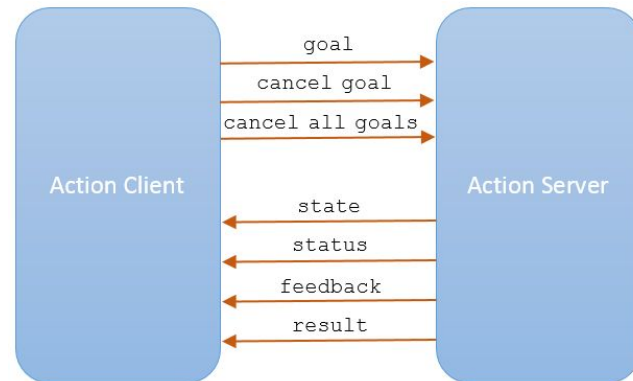
# What is ROS

## ▶ ROS Services:

- ▶ Services are defined using srv files.
- ▶ srv files are compiled by a ROS client library.
- ▶ Like Topics, Services are associated service type
- ▶ A service is created as a .srv file, composed of two parts:
  - ▶ Request & a Response

## ▶ ROS Actions:

- ▶ A client-server communication like service but Preemptive
- ▶ An action has goal or goals, each of which is identified by an id.
- ▶ Used for a goal that runs for a longer time, but provides feedback during execution, like moving a robot.



Source: <http://wiki.ros.org/ROS/Tutorials/CreatingMsgAndSrv> <http://wiki.ros.org/srv>

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

# What is ROS

---

## ▶ ROS Bags:

- ▶ Used for Recording and Playback.
- ▶ Recording:
  - ▶ Storing ROS message data, for later usage.
- ▶ Playback:
  - ▶ Bag files can be played back in ROS to the same topics they were recorded from, or even remapped to new topics
- ▶ ROS has variety of tools to store, process, analyze, and visualize rosbags.
- ▶ A rosbag subscribes to one or more ROS topics, and store the serialized message data in a file as it is received

---

▶ Source:

<http://wiki.ros.org/Bags>

<http://wiki.ros.org/ROS/Tutorials/Recording%20and%20playing%20back%20data>

# Installing ROS on UBUNTU

---

- ▶ Follow steps available in the following link [Ubuntu 18.04]:
  - ▶ <http://wiki.ros.org/melodic/Installation/Ubuntu>
- ▶ Follow steps available in the following link [Ubuntu 16.04]:
  - ▶ <http://wiki.ros.org/kinetic/Installation/Ubuntu>
- ▶ Follow steps available in the following link [Ubuntu 14.04]:
  - ▶ <http://wiki.ros.org/indigo/Installation/Ubuntu>

# Catkin Package:

## Creating catkin workspace

---



### ► Creating catkin workspace

- This is a one-time step, it is only required at the first time to create a workspace
  - This step is for PC [not RPi], as the RPi ROS installation creates a workspace calls “ros\_catkin\_ws”, as will be shown later

```
> mkdir ~/catkin_ws  
> mkdir ~/catkin_ws/src  
> cd ~/catkin_ws/src  
> catkin_init_workspace
```



# Catkin Package: Creating & Building a Package

---



## ► 1. Creating package

- `Catkin_create_pkg package_name [depend1] [depend2]`
  - depends: are the libraries required by the nodes that will be created in the package [roscpp & rospy will be almost always included]

```
> cd ~/catkin_ws/src  
> catkin_create_pkg ros_001 roscpp rospy std_msgs
```

- The package structure should look like this:

```
tornado@tornado-hp:~/catkin_ws/src/ros_001$ tree  
.  
├── CMakeLists.txt  
└── package.xml
```

## Creating catkin workspace & Building first package

---

### ▶ 1. Creating package

- ▶ The package will have two auto-generated files  
[Will be explained in more details when creating C/C++ nodes]
  - ▶ a. ***"CMakeLists.txt"***: Defines which C/C++ nodes to be built, and what dependencies the package requires, also the libraries to be linked when building
  - ▶ b. ***"Package.xml"***: Provides meta information about the package and required dependencies at build and runtime also.

### ▶ 2. Building the package

- ▶ To compile the package, first cd to ~/catkin\_ws, then use the catkin\_make command:

```
> cd ~/catkin_ws  
> catkin_make
```



- ▶ To create a C/C++ node:
  - ▶ A. Create the C/C++ source code file “publisher.cpp”
    - ▶ Create a source code C++ file of the node inside “src” directory inside the package main directory
  - ▶ B. Make some changes to the package’s “CMakeLists.txt”, located at `~/catkin_ws/src/ros_001`
    - ▶ To include the C/C++ file when building the package
  - ▶ C. Make some changes to the package’s “package.xml”, located at `~/catkin_ws/src/ros_001`
    - ▶ To define the build and run dependencies that the package will require

### ► A. Create the C/C++ source code file “publisher.cpp”

```
> mkdir ~/catkin_ws/src/ros_001/src  
> cd ~/catkin_ws/src/ros_001/src  
> gedit publisher.cpp
```

Important code snippets a ROS Publisher Node should have:

- 1. Commonly used Libraries
  - <ros/ros.h>: required for ROS functionality
  - <geometry\_msgs/Twist.h>: to be able to create a message of type Twist
  - <std\_msgs/String.h>: for standard string messages

```
#include <ros/ros.h>  
#include <geometry_msgs/Twist.h>  
#include <std_msgs/String.h>  
#include <stdlib.h>
```

# Catkin Package:

## Creating node [PUBLISHER] [C++]



### ► A. Create the C/C++ source code file “publisher.cpp”

Important code snippets a ROS Node should have:

#### ► 2. Initialization

```
int main(int argc, char **argv) {  
    // Initialize the ROS node, and assign the node a name  
    ros::init(argc, argv, "publisher");  
    // Create a node handler  
    ros::NodeHandle n;  
  
    // Ceates the publisher, and assign it to publish  
    // to the /diffBot/cmd_vel topic, with a queue size of 10  
    ros::Publisher pub=n.advertise<geometry_msgs::Twist>("diffBot/cmd_vel", 10);  
  
    // Sets the loop to publish at a rate of 10Hz  
    ros::Rate rate(10);
```

- [Optional]: In case of using GAZEBO husky package: To control the husky robot, the topic name it subscribe to is:  
“husky\_velocity\_controller/cmd\_vel”

Source: <https://www.clearpathrobotics.com/assets/guides/ros/Creating%20publisher.html>

<http://wiki.ros.org/ROS/Tutorials/WritingPublisherSubscriber%28c%2B%2B%29>

- ▶ A. Create the C/C++ source code file “publisher.cpp”

Important code snippets a ROS Node should have:

- ▶ 3. Loop until an exit interrupt happens [Ctrl+C]

```
while(ros::ok()) {  
    // Declares the message to be sent  
    geometry_msgs::Twist msg;  
  
    // Random x value between -2 and 2  
    msg.linear.x = 0.5;    // for 0.5m/sec  
    // Random y value between -3 and 3  
    msg.angular.z = 1.0;  
  
    // Publish the message  
    pub.publish(msg);  
  
    // Delays untill it is time to send another message  
    rate.sleep();  
}
```

# Catkin Package:

## Creating node [PUBLISHER] [C++]



- ▶ B. Perform some changes to the package's "CMakeLists.txt", located at `~/catkin_ws/src/ros_001`
  - ▶ 1. Make the cpp source files executable by adding it as a node to the package CMakeLists: [MANDATORY]

- ▶ Under

```
#####
```

```
## Build ##
```

```
#####
```

```
Under
```

```
## Declare a cpp executable
```

```
add the next lines:
```

```
NOTE: >>>> node_name is the given name to the node, here publisher
```

```
add_executable(node_name src/file_name.cpp)
target_link_libraries(node_name ${catkin_LIBRARIES})
```

```
add_dependencies(node_name ${${PROJECT_NAME}_EXPORTED_TARGETS} ${catkin_EXPORTED_TARGETS})
# [OPTIONAL]
```

Source: <https://www.clearpathrobotics.com/assets/guides/ros/Creating%20publisher.html>  
<http://wiki.ros.org/ROS/Tutorials/WritingPublisherSubscriber%28c%2B%2B%29>

# Catkin Package:

## Creating node [PUBLISHER] [C++]



- ▶ B. Perform some changes to the package's "CMakeLists.txt", located at ~/catkin\_ws/src/ros\_001
  - ▶ 2. Make sure dependencies are added: [optional]  
Dependencies are components imported from ROS and used in the code
    - ▶ At the first of the CMakeLists.txt  
Under  
find\_package(catkin REQUIRED)  
add the next lines:

```
find_package(catkin REQUIRED COMPONENTS
  roscpp
  rospy
  std_msgs
  geometry_msgs    # mostly used, but not required here
  message_generation    # mostly used, but not required here
)
```

- ▶ B. Perform some changes to the package's "CMakeLists.txt", located at `~/catkin_ws/src/ros_001`
- ▶ 3. Make sure catkin specific configuration added:[optional]
  - ▶ Under  
#####  
## catkin specific configuration ##  
#####  
Under  
## DEPENDS: system dependencies of this project that dependent projects also need  
edit the `catkin_package( ... )`  
to add the required configuration dependencies

```
catkin_package(  
  CATKIN_DEPENDS roscpp rospy std_msgs geometry_msgs message_runtime  
)
```

- ▶ C. Perform some changes to the package's "package.xml", located at ~/catkin\_ws/src/ros\_001
  - ▶ 1. Add Build and Run dependencies: [MANDATORY]
    - ▶ Under <buildtool\_depend>catkin</buildtool\_depend> add the next lines:

```
<build_depend>std_msgs</build_depend>
<build_depend>geometry_msgs</build_depend>

<build_export_depend>std_msgs</build_export_depend>
<build_export_depend>geometry_msgs</build_export_depend>

<exec_depend>std_msgs</exec_depend>
<exec_depend>geometry_msgs</exec_depend>

<!-- for ros indigo and before use "run_depend" instead of "exec_depend" -->
<run_depend>std_msgs</run_depend>
<run_depend>geometry_msgs</run_depend>
```



# Catkin Package:

## Creating node [PUBLISHER] [C++]



- ▶ After performing A, B, and C, now cd to ~/catkin\_ws and build the package

```
> cd ~/catkin_ws  
> catkin_make
```

- ▶ If you have error with “**geometry\_msgs package**” not found or something, install it manually, then rerun the “catkin\_make” command

```
> sudo apt-get install ros-geometry-msgs
```

- ▶ If the problem still not solved, then download the geometry\_msgs manually from github indigo repo. to “~/ros\_catkin\_ws/src”

```
> cd ~/ros_catkin_ws/src  
> git clone -b indigo-devel https://github.com/ros/common_msgs.git
```

- ▶ We may need to remove some not used packages from the downloaded folder if they generated error.
- ▶ Then re-build the **catkin\_ws** as explained before.

Source: <https://www.clearpathrobotics.com/assets/guides/ros/Creating%20publisher.html>  
<http://wiki.ros.org/ROS/Tutorials/WritingPublisherSubscriber%28c%2B%2B%29>

# Catkin Package:

## Creating node [PUBLISHER] [C++]

---



### ► To run the node

- 1. Run the Master node in a separate terminal terminal

```
> roscore
```

- 2. In a new terminal, source the catkin\_ws

```
> source ~/catkin_ws/devel/setup.bash
```

- 3. Then, run the node:

- > rosrun [package\_name] [node\_name]

```
> rosrun ros_001 publisher
```

- To list the current available topics, open a new terminal

```
> rostopic list
```

- To monitor the data published by a topic

```
> source ~/catkin_ws/devel/setup.bash
```

```
> rostopic echo /topic_name
```

# Catkin Package:

## Creating node [PUBLISHER] [Python]

---



- ▶ To create the node in Python:

- ▶ In the package you created, create a folder called “scripts”

```
> mkdir ~/catkin_ws/src/ros_001/scripts
```

- ▶ In that folder “scripts”, create your python publisher node

```
> cd ~/catkin_ws/src/ros_001/scripts  
> gedit publisher.py
```

- ▶ After finishing the python script coding, make the script executable

```
> chmod a+x publisher.py
```

- ▶ Now, the python script is ready to rosrn, catkin\_make is not required. Also editing “CMakeLists.txt” is not required

```
> source ~/catkin_ws/devel/setup.bash  
> rosrn ros_001 publisher.py
```



# Catkin Package:

## Creating node [PUBLISHER] [Python]

---



- Important python code snippets in the node:

```
#!/usr/bin/env python

import rospy
from geometry_msgs.msg import Twist

def publisher():
    # this part defines the publisher interface
    # husky_velocity_controller/cmd_vel: is the topic name
    # Twist: is the topic type is a Twist message
    # queue_size: limits the amount of queued messages if any subscriber is not receiving
    # them fast enough
    pub = rospy.Publisher('diffBot/cmd_vel', Twist, queue_size=10)

    # publisher: node name, tells rospy the name of your node --
    # until rospy has this information, it cannot start communicating with the ROS Master.
    # node name must not include any "/" characters

    # (anonymous = True): In ROS, nodes are uniquely named.
    # If two nodes with the same name are launched, the previous one is kicked off.
    # The anonymous=True flag means that rospy will choose a unique name for our
    'subscriber' node,
    # so that multiple nodes can run simultaneously.
    rospy.init_node('publisher', anonymous = True)

    # Creates a Rate object rate.
    # It offers a convenient way for looping at the desired rate, with the help of its
    # method sleep().
    # 10: means that, we will go through the loop 10 times per second. (as long as our
    # processing time does not exceed 1/10th of a second!)
    rate = rospy.Rate(10) # 10hz

    # Define an object of the type Twist, preparing for publishing it
    vel = Twist()
```

# Catkin Package:

## Creating node [PUBLISHER] [Python]

---



- ▶ Important python code snippets in the node:

```
# Standard rospy construct: checking the rospy.is_shutdown() flag and then doing work.
# You have to check is_shutdown() to check if your program should exit (e.g. if there
is a Ctrl-C or otherwise).
while not rospy.is_shutdown():
    vel.linear.x = 1;
    vel.angular.z = 1;
#     hello_str = "hello world %s" % rospy.get_time()
#     rospy.loginfo(hello_str)

    # Perform the publishing action
    pub.publish(vel)

    # Sleeps just long enough to maintain the desired rate through the loop, here "10
loops/second"
    rate.sleep()

if __name__ == '__main__':
    try:
        publisher()
    except rospy.ROSInterruptException:
        pass
```

# Sample Nodes



## Publisher Node:

```
#!/usr/bin/env python
import rospy
from std_msgs.msg import String
def simplePublisher():
    pub = rospy.Publisher('/topic_1', String, queue_size=10)
    rospy.init_node('node_1', anonymous=False)
    rate = rospy.Rate(1) # 1hz

    # The string to be published on the topic.
    topic1_content = "my first ROS topic"
    while not rospy.is_shutdown():
        pub.publish(topic1_content)
        rate.sleep()
if __name__ == '__main__':
    try:
        simplePublisher()
    except rospy.ROSInterruptException:
        pass
```

Source: <https://thepihut.com/blogs/raspberry-pi-tutorials/17789160-backing-up-and-restoring-your-raspberry-pis-sd-card>

<https://raspberrypi.stackexchange.com/questions/7177/image-of-a-16gb-card-containing-unpartitioned-space-at-the-end-truncating-poss>

- ▶ To create a C/C++ node:

- ▶ A. Create the C/C++ source code file “publisher.cpp”

- ▶ Create a source code C++ file of the node inside “src” directory inside the package main directory

```
> cd ~/catkin_ws/src/ros_001/src  
> gedit subscriber.cpp
```

- ▶ B. Make some changes to the package’s “CMakeLists.txt”, located at ~/catkin\_ws/src/ros\_001

- ▶ To include the C/C++ file when building the package

- ▶ C. Make some changes to the package’s “package.xml”, located at ~/catkin\_ws/src/ros\_001 [if required]

- ▶ To define the build and run dependencies that the package will require



# Catkin Package:

## Creating node [SUBSCRIBER] [C++]



Important code snippets a ROS Subscriber should have:

```
#include <ros/ros.h>
#include "std_msgs/String.h"
#include <geometry_msgs/Twist.h>
#include <stdlib.h>

// Topic messages callback
void velCallback(const geometry_msgs::Twist msg)
{
    ROS_INFO("[Subscriber] received:\n[x=%f]\n[y=%f]\n[z=%f]\n-----\n", msg.linear.x,
msg.linear.y, msg.linear.z);
}

int main(int argc, char **argv)
{
    // Initiate new ROS node named "subscriber"
    ros::init(argc, argv, "subscriber");
    // Create a node handle: it is reference assigned to a new node
    ros::NodeHandle n;

    // subscribe to a given topic, in this case "diffBot/cmd_vel"
    // velCallback: is the name of the callback function that will be executed each
time a message is revealed
    ros::Subscriber sub = n.subscribe("diffBot/cmd_vel", 1000, velCallback);

    // Enter a loop, pumping callbacks
    ros::spin();

    return 0;
}
```

Source: <https://www.clearpathrobotics.com/assets/guides/ros/Creating%20publisher.html>  
<http://wiki.ros.org/ROS/Tutorials/WritingPublisherSubscriber%28c%2B%2B%29>



# Catkin Package:

## Creating node [SUBSCRIBER] [C++]



- ▶ B. Perform some changes to the package's "CMakeLists.txt", located at `~/catkin_ws/src/ros_001`
  - ▶ 1. Make the cpp source files executable by adding it as a node to the package CMakeLists: [MANDATORY]

- ▶ Under

```
#####
```

```
## Build ##
```

```
#####
```

```
Under
```

```
## Declare a cpp executable
```

```
add the next lines:
```

```
NOTE: >>>> node_name is the given name to the node, here publisher
```

```
add_executable(subscriber src/subscriber.cpp)
target_link_libraries(subscriber ${catkin_LIBRARIES})

## add_dependencies(subscriber ${${PROJECT_NAME}_EXPORTED_TARGETS}
${catkin_EXPORTED_TARGETS}) [OPTIONAL]
```

# Catkin Package:

## Creating node [SUBSCRIBER] [C++]



- ▶ After performing A, B and may be C, now cd to the catkin\_ws and build the package

```
> cd ~/catkin_ws  
> catkin_make
```

- ▶ Run the subscriber node

- ▶ 1. source the catkin\_ws

```
> source ~/catkin_ws/devel/setup.bash
```

- ▶ 2. run the node

```
> rosrn ros_001 subscriber
```



# Catkin Package:

## Creating node [SUBSCRIBER] [Python]

---



- ▶ To create the node in Python:

- ▶ In the package you created, create a folder called “scripts”

```
> mkdir ~/catkin_ws/ros_001/scripts
```

- ▶ In that folder “scripts”, create your python publisher node

```
> cd ~/catkin_ws/ros_001/scripts  
> gedit subscriber.py
```

- ▶ After finishing the python script coding, make the script executable

```
> chmod a+x subscriber.py
```

- ▶ Now, the python script is ready to rosrn, catkin\_make is not required. Also editing “CMakeLists.txt” is not required

```
> source ~/catkin_ws/devel/setup.bash  
> rosrn ros_001 subscriber.py
```



# Catkin Package:

## Creating node [SUBSCRIBER] [Python]

---



- Important python code snippets in the node:

```
#!/usr/bin/env python

import rospy
from geometry_msgs.msg import Twist

def callback(msg):
    rospy.loginfo("Received a /cmd_vel message!")
    rospy.loginfo("Linear Components: [%f, %f, %f]"%(msg.linear.x, msg.linear.y,
msg.linear.z))
    rospy.loginfo("Angular Components: [%f, %f, %f]"%(msg.angular.x, msg.angular.y,
msg.angular.z))

def subscriber():
    # subscriber: node name, tells rospy the name of your node --
    # until rospy has this information, it cannot start communicating with the ROS Master.
    # node name must not include any "/" characters

    # (anonymous = True): In ROS, nodes are uniquely named.
    # If two nodes with the same name are launched, the previous one is kicked off.
    # The anonymous=True flag means that rospy will choose a unique name for our
'subscriber' node,
    # so that multiple listeners can run simultaneously.
    rospy.init_node('subscriber', anonymous = True)

    # /diffBot/cmd_vel: Define name of the topic that the node will subscribe to
    # Twist: is type of the topic
    # callback: is the function to be called when a message is successfully received from
that topic
    rospy.Subscriber("/diffBot/cmd_vel", Twist, callback)

    # spin() simply keeps python from exiting until this node is stopped
    rospy.spin()

if __name__ == '__main__':
    subscriber()
```

# ROS Launch

---

- ▶ Launch files are very common in ROS.
- ▶ They provide a method to start up multiple Nodes and a Master. We can also set parameters in the launch file.
- ▶ To create a launch file, in the package located at “~/catkin\_ws/src/ros\_001” create a directory called “launch”, inside it create a “robot\_launch.launch” file

```
> mkdir ~/catkin_ws/src/ros_001/launch
> gedit ~/catkin_ws/src/ros_001/launch/robot_launch.launch
```

# ROS Launch

---

- ▶ Here's a sample of a launch file contents:

```
<launch>
  <node pkg="ros_000" type="publisher" name="publisher"/>
  <node pkg="ros_000" type="subscriber.py" name="subscriber"/>
</launch>
```


- ▶ To run the launch file

```
> roslaunch ros_001 robot_test.launch
```

- ▶ For nodes to output data on the terminal that is running the launch file:

```
<?xml version="1.0"?>
<launch>
  <node pkg="ros_007" type="sub_cpp_node_1" name="sub_cpp_node_1"
    output="screen"/>
  <node pkg="ros_007" type="sub1.py" name="sub_py" output="screen"/>
</launch>
```

---



# ROS Launch

---

- ▶ Launch file with gazebo empty world and a robot

```
<?xml version="1.0" ?>
<launch>
  <param name="robot_description" command="$(find xacro)/xacro.py '$(find
mybot_description)/urdf/mybot.xacro' />
  <include file="$(find gazebo_ros)/launch/empty_world.launch">
    <node name="urdf_spawner" pkg="gazebo_ros" type="spawn_model" respawn="false"
output="screen"
      args="-urdf -model mybot -param robot_description"/>
    <include file="$(find mybot_control)/launch/mybot_control.launch" />
  </include>
```

# Message passing between Multiple Machines



- ▶ For passing messages between nodes located on different machines
  - ▶ 1. On the Host machine: run the master node [roscore]

```
> roscore
```

- ▶ The master node should show its URI on the network
  - ROS\_MASTER\_URI=http://tornado-hp:11311/

```
SUMMARY =====
PARAMETERS
* /roscpp: indigo
* /rosversion: 1.11.19
NODES
auto-starting new master
process[master]: started with pid [6460]
ROS_MASTER_URI=http://tornado-hp:11311/
setting /run_id to ad84416e-2da3-11e7-9f2d-38b1dba5952b
process[roscpp-1]: started with pid [6473]
started core service [/roscpp]
```



# Message passing between Multiple Machines



- ▶ For passing messages between nodes located on different machines
  - ▶ 2. On every other machine: [on the same network], on each new terminal
    - A. First, Configure the ROS\_MASTER\_URI, so that the master on the host machine is used
  - ▶

```
> export ROS_IP=machine_IP  
> export ROS_MASTER_URI=http://master_machine_IP:11311
```
  - Then, run the node you want, now all nodes on all machines are using the same ROS Master node
  - ▶ Run the publisher node on the guest machine

```
> source ~/catkin_ws/devel/setup.bash  
> rosruncatkin ros_001 publisher
```
  - ▶ On the host machine, run the subscriber node

```
> source ~/catkin_ws/devel/setup.bash  
> Rosrun ros_001 subscriber
```

# Message passing between Multiple Machines

---



## ► Troubleshooting:

- On the guest machine, if the hostname of the Master Node [roscore] machine is not identified, write its IP instead

```
> export ROS_MASTER_URI=http://192.168.1.106:11311
```

- On every terminal on the guest machine, before running a node, we have to identify the roscore before working

```
> source ~/catkin_ws/devel/setup.bash  
> export ROS_MASTER_URI=http://192.168.1.106:11311  
> rosrn ros_001 subscriber
```

# Message passing between Multiple Machines



## ▶ Troubleshooting:

- ▶ If the host-machine (running the Master Node [roscore]) can't find ip of the machine running a node, on the machine, before running the node, export its local ip
- ▶ Exporting the ROS\_IP can also be useful if any node on any machine even the master machine wasn't well recognized by the Master Node, because its IP isn't published

```
started roslaunch server http://tornado-hp:52132/
ros_comm version 1.11.19

[INFO] [1494431347.754886874]: [Subscriber] received:
=====
[2500.000000]
PARAMETERS
 * /roscore: indigo
 * /rosversion: 1.11.19
NODES
  moving FORWARD @ speed = 500
auto-starting new master
process[roscore]: started with pid [16297]
ROS_MASTER_URI=http://tornado-hp:11311/

setting /run_id to 0c6b1420-3595-11e7-9179-38b1dba5952b
process[roscore-1]: started with pid [16310]
started core service [/roscore]
Couldn't find an AF_INET address for [raspberrypi]
Couldn't find an AF_INET address for [raspberrypi]
```

```
> export ROS_IP='hostname -I'
> roslaunch ros_001 subscriber
```

# Message passing between Multiple Machines

---



## ► Troubleshooting:

- If, later, a problem happened with the master node, and the terminal can't find it, re-export it, using its name or its IP, even if it is to the localhost

```
> export ROS_MASTER_URI=http://localhost:11311
```



# Building and Controlling a Robot in GAZEBO



GAZEBO

ROS



GAZEBO + ROS

Meta Package: **gazebo\_ros\_pkgs**

## gazebo

Stand Alone Core

urdfdom

## gazebo\_ros

Formerly simulator\_gazebo/gazebo

This package wraps gzserver and gzclient by using two Gazebo plugins that provide the necessary ROS interface for messages, services and dynamic reconfigure

**ROS node name:**  
gazebo

**Plugins:**  
gazebo\_ros\_api\_plugin  
gazebo\_ros\_paths\_plugin

**Usage:**  
roslaunch gazebo\_ros gazebo  
roslaunch gazebo\_ros gzserver  
roslaunch gazebo\_ros gzclient  
roslaunch gazebo\_ros spawn\_model  
roslaunch gazebo\_ros perf  
roslaunch gazebo\_ros debug

## gazebo\_msgs

Msg and Srv data structures for interacting with Gazebo from ROS.

## gazebo\_plugins

Robot-independent Gazebo plugins.

### Sensory

gazebo\_ros\_projector  
gazebo\_ros\_p3d  
gazebo\_ros\_imu  
gazebo\_ros\_laser  
gazebo\_ros\_f3d  
gazebo\_ros\_camera\_utils  
gazebo\_ros\_depth\_camera  
gazebo\_ros\_openni\_kinect  
gazebo\_ros\_camera  
gazebo\_ros\_bumper  
gazebo\_ros\_block\_laser  
gazebo\_ros\_gpu\_laser

### Motory

gazebo\_ros\_joint\_trajectory  
gazebo\_ros\_differdrive  
gazebo\_ros\_force  
gazebo\_ros\_template

### Dynamic Reconfigure

vision\_reconfigure  
hokuyo\_node  
camera\_synchronizer

## gazebo\_tests

Merged to gazebo\_plugins

Contains a variety of unit tests for gazebo, tools and plugins.

## gazebo\_worlds

Merged to gazebo\_ros

Contains a variety of unit tests for gazebo, tools and plugins.

wg  
simple\_erratic  
simple\_office  
wg\_collada\_throttled - delete  
wg\_collada  
grasp  
empty\_throttled  
3stacks  
elevator  
simple\_office\_table  
scan  
empty  
simple  
balcony  
camera  
test\_friction  
simple\_office2  
empty\_listener

## gazebo\_tools

Removed

## gazebo\_ros\_api\_plugin

### Gazebo Subscribed Topics

~/set\_link\_state  
~/set\_model\_state

### Gazebo Published Parameters

/use\_sim\_time

### Gazebo Published Topics

/clock  
~/link\_states  
~/model\_states

### Gazebo Services

~/spawn\_urdf\_model  
~/spawn\_sdf\_model  
~/delete\_model

### State and properties getters

...

### State and properties setters

...

### Simulation control

~/pause\_physics  
~/unpause\_physics  
~/reset\_simulation  
~/reset\_world

### Force control

~/apply\_body\_wrench  
~/apply\_joint\_effort  
~/clear\_joint\_forces  
~/clear\_body\_wrenches

## gazebo\_ros\_paths\_plugin

Provides ROS package paths to Gazebo

ROS packages

Gazebo Plugin

Deprecated from simulator\_gazebo

# Building and Controlling a Robot in GAZEBO [Pre-Built]



## ► First, let's download and try a pre-built robot and world. **[TIAGO]**

### ► A. Make sure that the Tiago robot and world is installed:

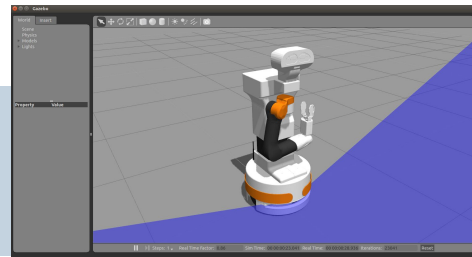
- 1. Download this file and name it "tiago\_public.rosinstall" in the catkin\_ws directory:  
[https://raw.githubusercontent.com/pal-robotics/tiago\\_tutorials/kinetic-devel/tiago\\_public-melodic.rosinstall](https://raw.githubusercontent.com/pal-robotics/tiago_tutorials/kinetic-devel/tiago_public-melodic.rosinstall)

- 2. Install the environment as follows:

```
> cd ~/catkin_ws  
> rosinstall src /opt/ros/melodic tiago_public.rosinstall  
> sudo rosdep init  
> rosdep update
```

- 3. Run the following instruction to make sure all dependencies are installed

```
> rosdep install -y --from-paths src --ignore-src --rosdistro melodic --skip-keys="opencv2 opencv2-nonfree  
pal_laser_filters speed_limit sensor_to_cloud hokuyo_node libdw-dev python-graphitesend-pip python-statsd  
pal_filters pal_vo_server pal_usb_utils pal_pcl pal_pcl_points_throttle_and_filter pal_karto pal_local_joint_control  
camera_calibration_files pal_startup_msgs pal-orbbec-openni2 dummy_actuators_manager pal_local_planner  
gravity_compensation_controller current_limit_controller dynamic_footprint dynamixel_cpp tf_lookup opencv3"
```



# Building and Controlling a Robot in GAZEBO [Pre-Built]



GAZEBO



ROS

- ▶ First, let's download and try a pre-built robot and world. **[TIAGO]**
  - ▶ A. Make sure that the Tiago robot and world is installed:

- ▶ 4. Build the workspace and source it

```
> cd ~/catkin_ws  
> catkin_make  
> source ~/catkin_ws/devel/setup.bash
```

Note:

The public simulation of TIAGo allows two different versions:

- TIAGo Steel: in this configuration the end-effector is a parallel gripper
- TIAGo Titanium: the wrist has a 6-axis force/torque sensor and the end-effector is the under-actuated 5-finger Hey5 hand.

- ▶ 5. Launch simulation:

To launch simulation of TIAGo steel or titanium, run one the following commands:

```
> roslaunch tiago_gazebo tiago_gazebo.launch public_sim:=true robot:=steel
```

```
> roslaunch tiago_gazebo tiago_gazebo.launch public_sim:=true robot:=titanium
```

# Building and Controlling a Robot in GAZEBO [Pre-Built]



## ► First, let's download and try a pre-built robot and world. **[TIAGO]**

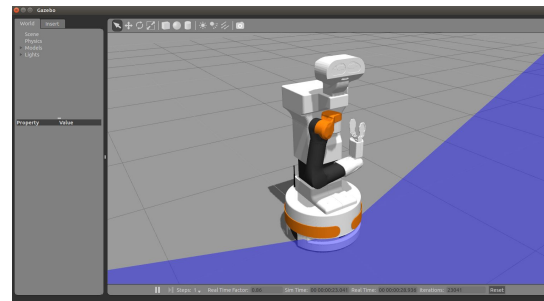
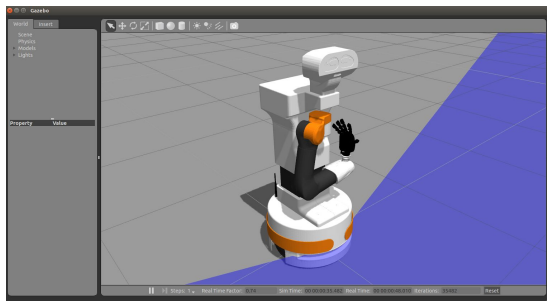
### ► B. Teleoperating the mobile base with keyboard:

- 1. Launch the simulation by running the following command in a terminal:

```
> roslaunch tiago_gazebo tiago_gazebo.launch public_sim:=true robot:=titanium  
world:=simple_office_with_people
```

- 2. Open a new terminal and run the key\_teleop using the following command:

```
> rosrun key_teleop key_teleop.py
```





# Building and Controlling a Robot in GAZEBO [Pre-Built]



## ► First, let's download and try a pre-built robot and world. **[TIAGO]**

### ► C. Moving the base through velocity commands:

- 1. Launch the simulation by running the following command in a terminal:

```
> roslaunch tiago_gazebo tiago_gazebo.launch public_sim:=true robot:=titanium  
world:=simple_office_with_people
```

- 2. Open a new terminal and run the following command to find exact name of the required node to talk to:

```
> rostopic list
```

- 3. Send velocity commands through terminal

- ☐ a. Moving forward and backward:

```
> rostopic pub /mobile_base_controller/cmd_vel geometry_msgs/Twist -r 3 -- '[0.5,0.0,0.0]'  
'[0.0, 0.0, 0.0]'
```

- ☐ b. Turning left and right:

```
> rostopic pub /mobile_base_controller/cmd_vel geometry_msgs/Twist -r 3 -- '[0.0,0.0,0.0]'  
'[0.0, 0.0, 0.5]'
```

# Building and Controlling a Robot in GAZEBO [Pre-Built]



- ▶ First, let's download and try a pre-built robot and world. **[HUSKY]**

- ▶ Make sure that the husky robot is installed

```
> sudo apt-get update  
> sudo apt-get install ros-melodic-husky-desktop  
> sudo apt-get install ros-melodic-husky-simulator
```

- ▶ Running a virtual husky in gazebo, in new terminal write:

```
> roslaunch husky_gazebo husky_empty_world.launch
```

- ▶ Running a virtual husky in RViz, in a new terminal write:

```
> roslaunch husky_viz view_robot.launch
```

- ▶ Now, to make the robot move, publish a geometry twist message with the required linear and angular velocities:

```
> rostopic pub -r 10 /husky_velocity_controller/cmd_vel geometry_msgs/Twist '{linear: {x: 1.0,  
y: 0.0, z: 0.0}, angular: {x: 0.0,y: 0.0,z: 0.0}}'
```

# Building and Controlling a Robot in GAZEBO [Pre-Built]



GAZEBO



ROS

- ▶ First, let's download and try a pre-built robot and world.
  - ▶ Now, to make the robot move, publish a geometry twist message with the required linear and angular velocities:
    - ▶ “/husky\_velocity\_controller/cmd\_vel”: topic name
    - ▶ “geometry\_msgs/Twist”: topic type
    - ▶ “linear x: 1.0”: means go forward at 1m/s speed
    - ▶ “-r 10”: means publish at rate 10 Hz (10 messages/sec)

```
> rostopic pub -r 10 /husky_velocity_controller/cmd_vel geometry_msgs/Twist '{linear: {x: 1.0, y: 0.0, z: 0.0}, angular: {x: 0.0,y: 0.0,z: 0.0}}'
```
  - ▶ We may run the “rqt\_graph” ROS node to show the current nodes and topics and their publishers and subscribers relations visually

```
> rosrun rqt_graph rqt_graph
```
  - ▶ We may use teleop\_twist\_keyboard node to control the robot

```
> sudo apt-get install ros-melodic-teleop-twist-keyboard  
> rosrun teleop_twist_keyboard teleop_twist_keyboard.py
```

# Building and Controlling a Robot in GAZEBO [Native]



- ▶ For this, we will make a new folder to include all our packages in `~/catkin_ws/src` named “diffBot\_001”

```
> cd ~/catkin_ws/src  
> mkdir diffBot_001
```

- ▶ Inside this directory, create some packages
  - ▶ “diffBot\_description”: a package for the robot definition
    - sdf, urdf, meshes, materials, etc...
  - ▶ “diffBot\_gazebo”: for the world description and the launch files.
  - ▶ “diffBot\_control”: a package for the control nodes and scripts.

```
> cd diffBot_001  
> catkin_create_pkg diffBot_control  
> catkin_create_pkg diffBot_description  
> catkin_create_pkg diffBot_gazebo
```

# Building and Controlling a Robot in GAZEBO [Native]



GAZEBO

ROS

## ▶ 1. Create the diffBot model description

```
> cd diffBot_001/diffBot_description  
> mkdir urdf  
> gedit urdf/diffBot_model.xacro
```

- ▶ Place the robot description code in the diffBot\_model.xacro file

## ▶ 2. Create a GAZEBO world

```
> cd ~/catkin_ws/src/diffBot_001/diffBot_gazebo  
> mkdir launch worlds  
> gedit worlds/diffBot_world.world
```

- ▶ Place the world code in the diffBot\_world.world file

## ▶ 3. Create a roslaunch file

```
> gedit launch/diffBot_launch.launch
```

- ▶ Place the launch file code



# Building and Controlling a Robot in GAZEBO [Native]

---



GAZEBO



ROS

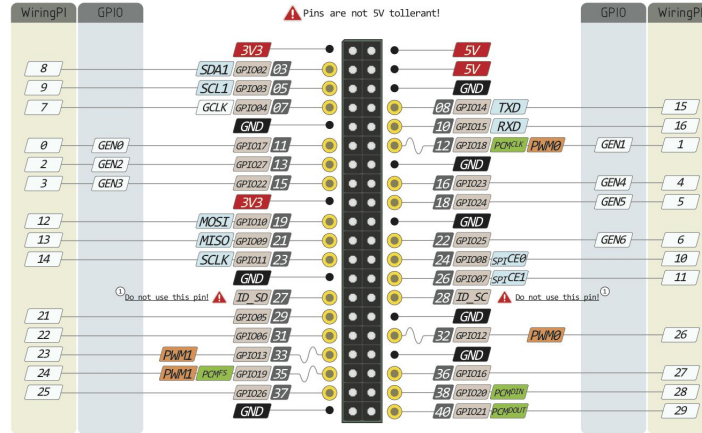
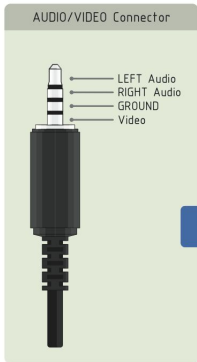
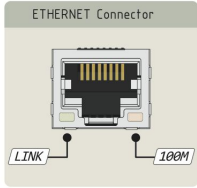
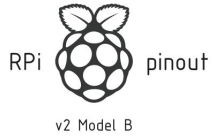
- ▶ We can also follow **step 1** shown in this tutorial:

<http://www.theconstructsim.com/ros-projects-exploring-ros-using-2-wheeled-robot-part-1/>



# Real-Time Robot using RPi

# ROS



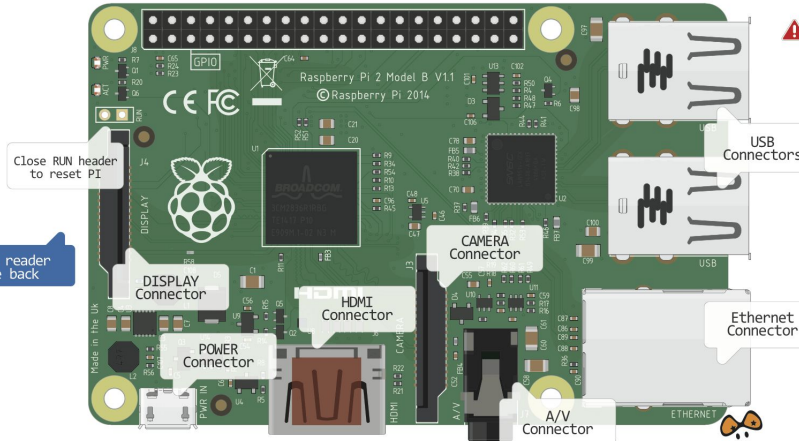
⚠ The PWM pin available on the GPIO header is shared with the Audio system



These pins are reserved for ID EEPROM

At boot time this I2C interface will be interrogated to look for an EEPROM that identifies the attached board and allow automatic setup of the GPIOs (and optionally, Linux drivers).

DO NOT USE these pins for anything other than attaching an I2C ID EEPROM. Leave unconnected if ID EEPROM not required.



⚠ Absolute MAX per pin 16mA recommended 8mA

⚠ Absolute MAX 50mA for entire package

## Installing Raspbian Stretch & ROS on the RPi

---

- ▶ Choose Your LINUX distribution, you can choose from:
  - ▶ **METHOD\_I:** ROSberryPi image: [Recommended for new users]
    - ▶ Prebuilt SD card image of Raspbian or Ubuntu with ROS already installed and may be OpenCV installed too:
      - i. ROSbots provides Raspbian Stretch Lite + ROS Kinetic + OpenCV  
[https://github.com/ROSbots/rosbots\\_setup\\_tools](https://github.com/ROSbots/rosbots_setup_tools)
      - ii. ROSberryPi, approved by ROS, provide ubuntu + ROS Kinetic  
<https://downloads.ubiquityrobotics.com/pi.html>
  - ▶ **METHOD\_II:** New & clean installation, then install ROS manually  
Two of the most famous distributions to use:
    - **Raspbian Stretch** [Latest Raspbian distro to the moment], Raspbian Lite is preferred.
    - **UBUNTU Core**, which is very trending in the IOT industry.



# Real-Time Robot using RPi [METHOD\_I]

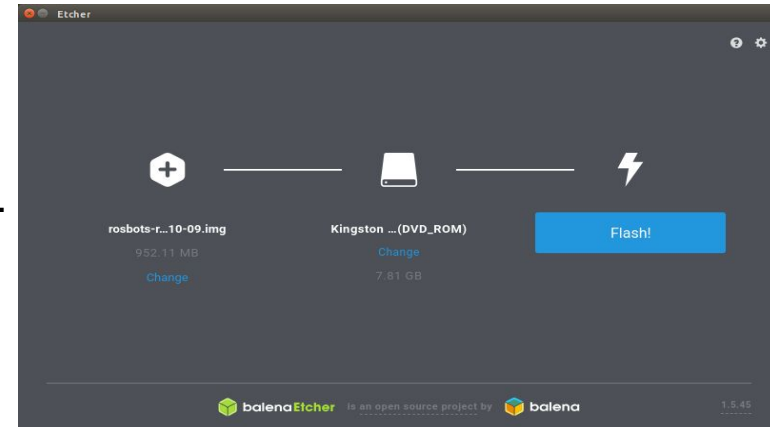
## Installing Raspbian Stretch & ROS on the RPi



### ► **METHOD\_I:**

i. Installing ROSberryPi image with Raspbian Stretch Lite + ROS Kinetic + OpenCV:

- 1. Download the SD card image:
  - ❑ Open the github link of the project  
[https://github.com/ROSBots/rosbots\\_setup\\_tools](https://github.com/ROSBots/rosbots_setup_tools)
  - ❑ Find the image download link and download it.
- 2. Flash the SD card image to your SD card
  - Using Etcher app [linux].
    - ❑ Run etcher application.
    - ❑ Select the downloaded SD card image.
    - ❑ Select your SD card.
    - ❑ Click on Flash button.



Resources:

[https://github.com/ROSBots/rosbots\\_setup\\_tools](https://github.com/ROSBots/rosbots_setup_tools)

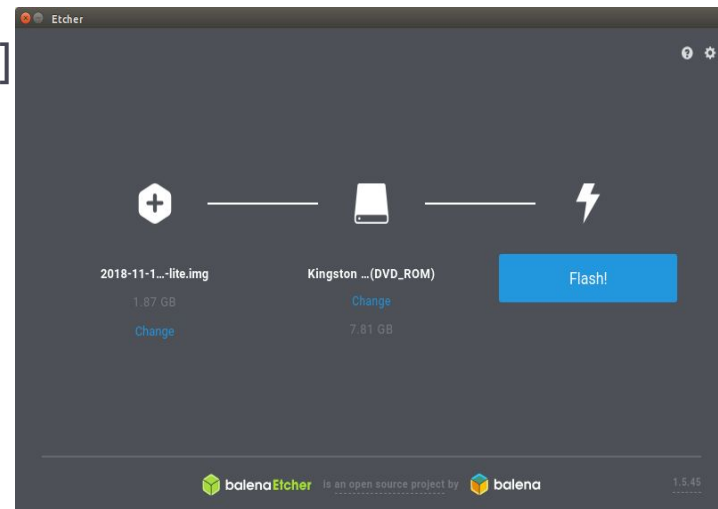
<https://www.balena.io/etcher/>

# Real-Time Robot using RPi [METHOD\_II]



## A. Installing Raspbian Stretch on the RPi

- ▶ Install Clean Raspbian Stretch Lite on the RPi -- **METHOD 1**
  - ▶ 1. Download Raspbian Stretch Lite:  
<https://www.raspberrypi.org/downloads/raspbian/>
  - ▶ 2. Download the etcher application for linux or windows from this link:  
<https://www.balena.io/etcher/>
  - ▶ 3. Insert the SD Card [Recommended: >16 GB]
  - ▶

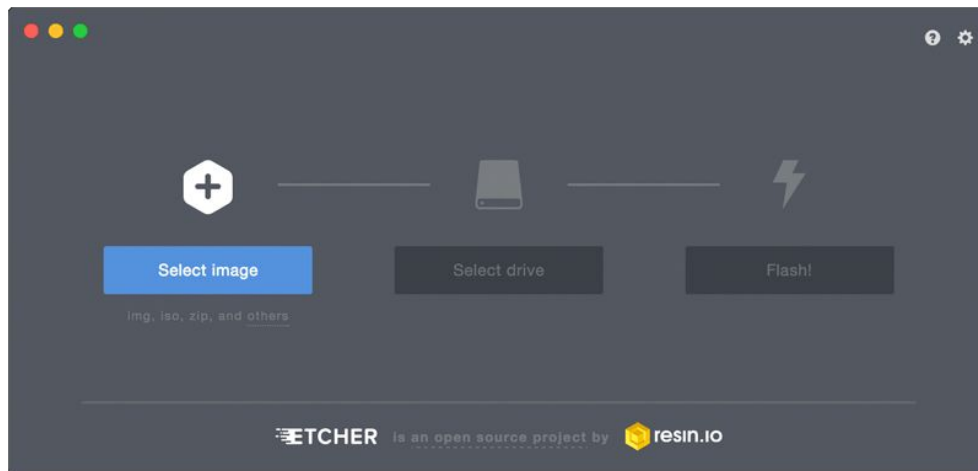


Download Link for Raspbian images:

<https://downloads.raspberrypi.org/raspbian/images/>

## A. Installing Raspbian Stretch on the RPi

- ▶ 1. Install Raspbian Stretch distribution on the RPi -- **METHOD 1**
  - ▶ Download the etcher application for linux or windows from this link:  
`> https://etcher.io/`
  - ▶ Use the etcher application to choose the required image then choose the target SD card, then flash it.



## A. Installing Raspbian Stretch on the RPi

---

### ▶ 1. Install Raspbian Stretch distribution on the RPi -- **METHOD 2**

- ▶ 1. Download Raspbian Stretch Lite SD card image.
- ▶ 2. Insert the SD Card [Recommended: 16 GB]
- ▶ 3. Detect the SD card path [/dev/sdx]

```
> df -h
```

- ▶ 4. Unmount all SD card drives

```
> umount /dev/sdb1  
> umount /dev/sdb2
```

- ▶ 5. Copy the image to the SD card

- ▶ May require “sudo” to run

```
> sudo dd if=2017-01-11-raspbian-jessie.img | pv | sudo dd of=/dev/sdb bs=4M conv=fsync
```

If the pv application is not installed, then install it using the command

```
> sudo apt-get install pv
```

- ▶ Wait until the copying has finished

---

Sources:

<https://www.raspberrypi.org/downloads/raspbian/>

<https://www.raspberrypi.org/documentation/installation/installing-images/linux.md>

<https://askubuntu.com/questions/215505/how-do-you-monitor-the-progress-of-dd>

## A. Installing Raspbian Stretch on the RPi

---

### ▶ 2. Preparing the Raspbian Stretch for booting

#### ▶ Activating SSH:

In the new Raspbian Stretch, SSH could be disabled by default

To Activate the SSH:

- ▶ Just open the boot partition on the SD card, and add an empty file with the name “ssh”, without any extensions

```
> cd /media/$USER/boot  
> touch ssh
```

## A. Installing Raspbian Stretch on the RPi

---

### ▶ 2. Preparing the Raspbian Stretch [Networking]

- ▶ If you want the RPi to connect to a wifi hotspot/router to connect to your LAN wirelessly:

Add ssid and password of your network to the file “wpa\_supplicant.conf” located on the other drive (drive other than “boot” drive) at “etc/wpa\_supplicant/”

- ▶ The file, “wpa\_supplicant.conf”, needs to be edited from the command line, using any text editor you have, also it needs “sudo ” permission

```
> sudo gedit etc/wpa_supplicant/wpa_supplicant.conf
```

- ▶ Add network information at the end of the form:

```
country=GB
ctrl_interface=DIR=/var/run/wpa_supplicant GROUP=netdev
update_config=1
network={
    ssid="NETWORK_SSID"
    psk="NETWORK_PASS"
}
```

## A. Installing Raspbian Stretch on the RPi

---

### ► 2. Preparing the Raspbian Stretch [Networking]

- To enable the onboard (RPi3) or USB WiFi (RPi2) and make it try to access networks defined in the `wpa_supplicant.conf` file:
  - Edit the file “interfaces” found at “/etc/network/”, it may need “sudo ” permission

```
> sudo gedit /etc/network/interfaces
```

- At the end of the file, add these lines, to define using the wlan0 trying to access networks found at `wpa_supplicant.conf`

```
auto lo
iface lo inet loopback

iface eth0 inet manual

allow-hotplug wlan0
iface wlan0 inet manual
wpa-conf /etc/wpa_supplicant/wpa_supplicant.conf
```

## A. Installing Raspbian Stretch on the RPi

---

### ▶ 3. Booting and accessing the Raspbian Stretch

- ▶ Install the SD card on the RPi, and power it up.
- ▶ Find the RPi's ip, to be able to access it using SSH
  - ▶ We may find IPs of devices connected on the network using one of the following commands:

```
> nmap -sP 192.168.1.0/24  
> arp -a
```

- ▶ Then connect to the RPi through SSH:  
use username "pi", and password "raspberry"

```
> ssh pi@192.168.1.100
```

- ▶ If the CLI showed this warning:  
"REMOTE HOST IDENTIFICATION HAS CHANGED!"  
then, remove the previous stored known\_hosts file located at  
"/home/\$USER/.ssh/known\_hosts" using the next command:

```
> rm /home/$USER/.ssh/known_hosts
```

---



## A. Installing Raspbian Stretch on the RPi

---

### ▶ 3. Booting and accessing the Raspbian Stretch

- ▶ Now, re access the Rpi using the same command and authentications

```
> ssh pi@192.168.1.100
```

- ▶ Once on the RPi enter the raspi-config page

```
> sudo raspi-config
```

- ▶ Expand the file system & Enable the SSH from advanced settings
  - ▶ In Interfacing Options, choose “SSH”, and enable it.
  - ▶ In Advanced Options, choose “Expand Filesystem”



## A. Installing Raspbian Stretch on the RPi

---

- ▶ 3. Booting and accessing the Raspbian Stretch
  - ▶ To view and control the Raspbian from the GUI through your laptop, install VNC server on the RPi and the VNC viewer on the laptop.
    - ▶ On the RPi, install the tightvncserver:

```
> sudo apt-get install tightvncserver
```
    - ▶ To run the vncserver, enter the following command:

```
> vncserver :1
```
    - ▶ **On the laptop (your PC, not the RPi)**, install vncviewer, open it and enter your RPi's ip followed by ":1" as the server, e.g. "192.168.1.100:1", enter the password and the RPi's desktop will show.

```
> sudo apt-get install vncviewer
```

## A. Installing Raspbian Stretch on the RPi

---

### ▶ 3. Booting and accessing the Raspbian Stretch

- ▶ To Run the VNC server at startup on the RPi:
  - ▶ Create a file named “tightvnc.desktop” in “~/ .config/autostart/”

```
> cd ~/.config/  
> mkdir autostart  
> cd autostart/  
> touch tightvnc.desktop  
> nano tightvnc.desktop
```

- ▶ Then, edit the file and add the following lines:

```
[Desktop Entry]  
Type=Application  
Name=TightVNC  
Exec=vncserver :1  
StartupNotify=false
```

- ▶ Now, next time you reboot your RPi, the vncserver will start automatically.

## B. Installing ROS on RPi

---

### ► Install ROS kinetic on the Raspbian Stretch

- One way to get ROS on your Raspbian Stretch OS, is to use a pre-compiled image of Jessie with ROS indigo already installed on it.
  - ❑ Check this link for such image:
  - ❑ <https://neverbenever.wordpress.com/2017/12/20/install-ros-and-opencv-in-raspberry-pi-raspbian-stretch/>
  - ❑ Image link:  
**ROS Kinetic** (Robot Operating System) and **OpenCV-3.3.1** on the Raspberry Pi 2 or 3 with **Raspbian Stretch**.  
[https://drive.google.com/file/d/1KMP9R-Yg3GXGDGbAxtfe\\_mwYa9hFUwmh/view](https://drive.google.com/file/d/1KMP9R-Yg3GXGDGbAxtfe_mwYa9hFUwmh/view)
- If you installed a fresh native copy of the Jessie OS, then you need to manually install the required ROS Distro on your Raspbian Stretch OS

## B. Installing ROS on RPi

---

### ► Install ROS kinetic on the Raspbian Stretch

#### ► 1. Preparing prerequisites:

```
> sudo apt-get install dirmngr
```

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

```
> sudo apt-key adv --keyserver hkp://ha.pool.sks-keyservers.net:80 --recv-key
421C365BD9FF1F717815A3895523BAEEB01FA11
```

```
> sudo apt-get update
```

```
> sudo apt-get upgrade # [optional]
```

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

Source: [https://dev.px4.io/en/ros/raspberrypi\\_installation.html](https://dev.px4.io/en/ros/raspberrypi_installation.html)

<http://wiki.ros.org/ROSBerryPi/Installing%20ROS%20Kinetic%20on%20the%20Raspberry%20Pi>  
<https://neverhaveover.wordpress.com/2017/12/20/install-ros-and-encyc-in-raspberry-pi-raspbian-stretch/>

## B. Installing ROS on RPi

---

### ► Install ROS kinetic on the Raspbian Stretch

#### ► 1. Preparing prerequisites:

- If the pip command failed, add “-H” option to the command, if it still raise errors, install the packages separately **[this step is optional]**

```
> sudo -H pip install rosdep      # [optional]
> sudo -H pip install rosinstall_generator    # [optional]
> sudo -H pip install wstool      # [optional]
> sudo -H pip install rosinstall  # [optional]
```

#### ► Now, initialize and update the rosdep

```
> sudo rosdep init
> rosdep update
```

- **If the “rosdep update” failed**, remove the initialized file then re-run the previous 2 commands.

```
> sudo rm /etc/ros/rosdep/sources.list.d/20-default.list    # [optional]
```

## B. Installing ROS on RPi

---

### ► Install ROS kinetic on the Raspbian Stretch

- 2.1 Now, we will download and build ROS indigo, two variants are available: desktop (full version - recommended) & ROS-Comm (light version)

```
> mkdir ~/ros_catkin_ws  
> cd ~/ros_catkin_ws  
> rosinstall_generator ros_comm --rostdistro kinetic --deps --wet-only --tar >  
kinetic-ros_comm-wet.rosinstall  
> wstool init -j8 src kinetic-ros_comm-wet.rosinstall
```

- This indigo packages are for robotic applications, can be added after adding the ros\_comm, [more support robots]: **[optional]**

```
> rosinstall_generator robot --rostdistro kinetic --deps --wet-only --tar > kinetic-robot-wet.rosinstall  
> wstool update -t src  
> wstool merge -t src kinetic-robot-wet.rosinstall
```

- ◻ If wstool init fails or is interrupted, you can resume the download by running the next command. It may require sudo privilege:

```
> wstool update -t src # [optional]
```

## B. Installing ROS on RPi

---

### ► Install ROS kinetic on the Raspbian Stretch

#### ► 2.2 Resolving Dependencies for ros\_comm on Stretch

##### ► 2.2.1. Resolving Unavailable Dependencies: **[optional]**

```
> mkdir -p ~/ros_catkin_ws/external_src
> cd ~/ros_catkin_ws/external_src
> wget
http://sourceforge.net/projects/assimp/files/assimp-3.1/assimp-3.1.1_no_test_models.zip/download -O assimp-3.1.1_no_test_models.zip
> unzip assimp-3.1.1_no_test_models.zip
> cd assimp-3.1.1
> cmake .
> make
> sudo make install
```

##### ► 2.2.2. Resolving Dependencies with rosdep, and make sure all dependencies are installed:

```
> cd ~/ros_catkin_ws
> rosdep install -y --from-paths src --ignore-src --rosdistro kinetic -r --os=debian:stretch
```



## B. Installing ROS on RPi

---

### ► Install ROS kinetic on the Raspbian Stretch

#### ► 2.3 Building the catkin workspace

- Before installing ROS, it is recommended to increase the swap space on the RPi, to avoid out-of-memory error **[optional]**. **Note: make sure to expand file system**  
**Note that: It is recommended to reset the swap size to 100 after ros installation.**

```
> sudo nano /etc/dphys-swapfile
```

- The default value in Raspbian is "CONF\_SWAPSIZE=100"  
Change it to: 1024

```
CONF_SWAPSIZE=1024
```

- Then, stop and start the service that manages the swapfile:

```
> sudo /etc/init.d/dphys-swapfile stop  
> sudo /etc/init.d/dphys-swapfile start
```

- To verify the amount of memory + swap:

```
> free -m
```

## B. Installing ROS on RPi

---

### ► Install ROS kinetic on the Raspbian Stretch

#### ► 2.3 Building the catkin workspace

- This will install ROS in the equivalent file location to Ubuntu in /opt/ros/kinetic however you can modify this as you wish

This step may take too much time to finish:

```
> sudo ./src/catkin/bin/catkin_make_isolated --install -DCMAKE_BUILD_TYPE=Release  
--install-space /opt/ros/kinetic
```

- ◻ If compilation fail with an "internal compiler error", it may be because the RPi is out of memory: Add swap space to the Pi and recompile.
- ◻ If the error persists try building with the -j2 option instead of the default -j4 option:

```
> sudo ./src/catkin/bin/catkin_make_isolated --install -DCMAKE_BUILD_TYPE=Release  
--install-space /opt/ros/kinetic -j2      # [optional]
```

- ◻ If the compilation raised error due to the cmake file, follow the solution available in this link:
  - <https://answers.ros.org/question/266665/ros-indigo-installation-problem-on-raspbian-jessie/>

## B. Installing ROS on RPi

---

### ► Install ROS kinetic on the Raspbian Stretch

#### ► 2.3 Building the catkin workspace

- Now, ROS should be installed. Don't forget to source the new installation, it is recommended to source the "setup.bash" in the ~/.bashrc, so that ROS environment variables are automatically added to your bash session every time a new shell is launched

```
> echo "source /opt/ros/kinetic/setup.bash" >> ~/.bashrc  
> echo "source /home/pi/ros_catkin_ws/devel/setup.bash" >> ~/.bashrc  
> source ~/.bashrc
```

- Now, you can compile all the workspace by running "catkin\_make", or compile certain packages using:

```
> catkin_make --pkg pkg_1 pkg_2
```

- If for some reason catkin\_make required root permissions, change the ownership of the ros\_catkin\_ws to "pi"

```
> cd ~  
> sudo chown -R pi ros_catkin_ws
```

## B. Installing ROS on RPi

---

### ► Adding Released Packages [***Optional for Future***]

- To add additional packages to the installed ROS workspace:

```
> cd ~/ros_catkin_ws  
> rosinstall_generator ros_comm ros_control joystick_drivers --rostdistro kinetic --deps  
--wet-only --tar > kinetic-custom_ros.rosinstall  
  
> wstool merge -t src kinetic-custom_ros.rosinstall  
> wstool update -t src
```

- After updating the workspace, run rosdep to install new dependencies that are required

```
> rosdep install -y --from-paths src --ignore-src --rostdistro kinetic -r --os=debian:stretch
```

- Then, rebuild the workspace:

```
> sudo ./src/catkin/bin/catkin_make_isolated --install -DCMAKE_BUILD_TYPE=Release  
--install-space /opt/ros/kinetic
```

# Real-Time Robot using RPi

## C. Installing wiringPi on RPi

---



### ► 1. Installing wiringPi [C++] on Raspbian Stretch:

```
> sudo apt-get install wiringpi
```

- To make sure wiringPi is well installed, and to show the RPi's pinout configuration:

```
> gpio -v  
> gpio readall
```



# Real-Time Robot using RPi

## C. Installing wiringPi on RPi



### ► 1. Installing wiringPi [C++] on Raspbian Stretch:

- To compile a C/C++ code, the wiringPi library has to be added as a library to the linker.

```
> gcc -Wall -o file_output file.c -lwiringPi
```

- To run a compiled file, a sudo privilege must be given.

- Either use sudo to run a file

```
> sudo ./file_output
```

- Or use “su” command to enter the root privileged shell, then all commands in that shell will run as super user privileged.

```
> su
```

Then “./file\_output” can be run directly without “sudo” in this shell

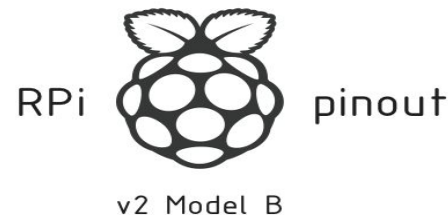
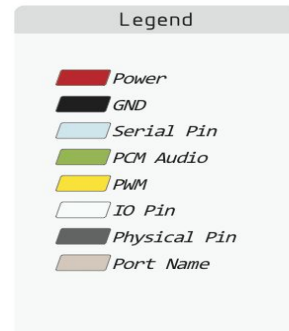
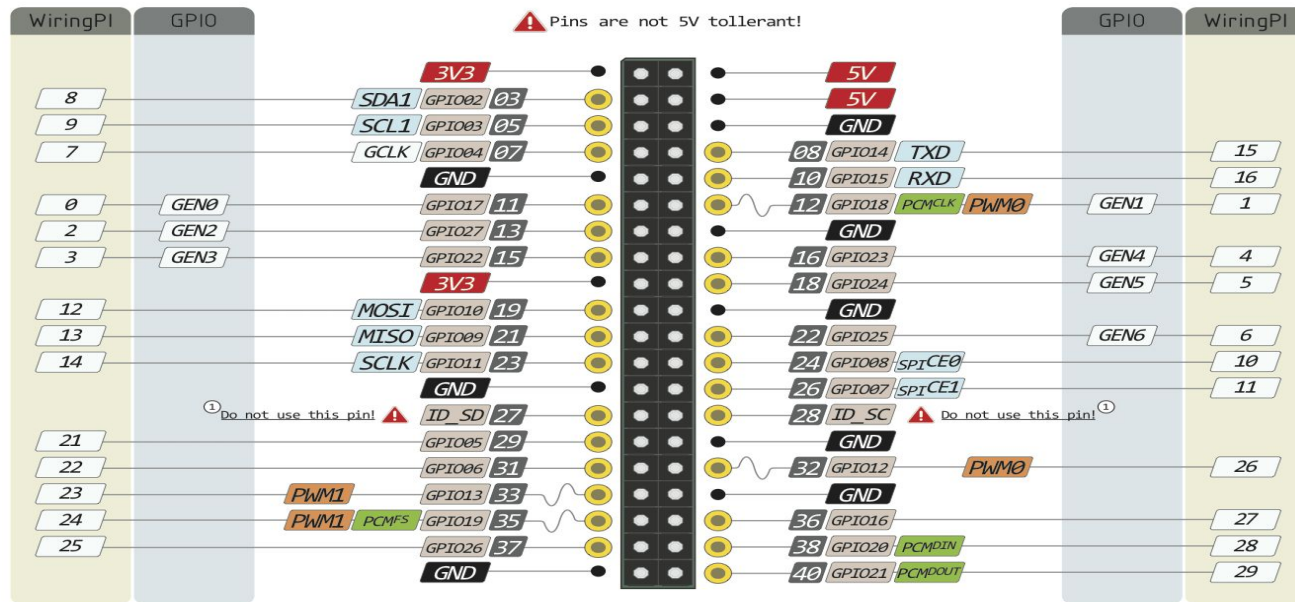
```
> ./file_output
```

# Real-Time Robot using RPi

## C. Installing wiringPi on RPi



- 1. Installing wiringPi [C++] on Raspbian Stretch:
  - This is the pinout mapping of the RPi with the wiringPi lib.:



⚠ The PWM pin available on the GPIO header is shared with the Audio system

Source:

<http://wiringpi.com/download-and-install/>

# Real-Time Robot using RPi

## C. Installing wiringPi on RPi

---



### ► 2. Using wiringPi [C++]: LED Blinking:

- To organize our work, all codes will be in “~/ros\_catkin\_ws/src”

```
> cd ~/ros_catkin_ws/src
```

- We will create and build a simple blink code to test the wiringpi library:

```
> mkdir blink  
> cd blink  
> nano blink.c
```





# Real-Time Robot using RPi

## C. Installing wiringPi on RPi



### ► 2. Using wiringPi [C++]: LED Blinking:

- Then write this code in the blink.c file, the LED is connected to pin 12 in the RPi, which is pin “1” in wiringPi

```
#include <wiringPi.h>
int main (void) {
    wiringPiSetup ();
    pinMode (1, OUTPUT);
    for (;;) {
        digitalWrite (1, HIGH); delay (500);
        digitalWrite (1, LOW); delay (500);
    }
    return 0;
}
```

- To compile the code use

```
> gcc -Wall -o blink blink.c -lwiringPi
```

- To run the compiled file use: (don't forget “sudo”)

```
> sudo ./blink
```

# Real-Time Robot using RPi

## C. Installing wiringPi on RPi

---



### ► 3. Installing wiringPi [Python] on Raspbian Stretch:

- For python wiringPi, first install these prerequisites:

```
> sudo apt-get install python-dev python-setuptools swig
```

- Then, download and install python wiringPi:

```
> cd ~  
> git clone --recursive https://github.com/WiringPi/WiringPi-Python.git  
> cd ~/WiringPi-Python  
> sudo python setup.py install
```



# Real-Time Robot using RPi

## C. Installing wiringPi on RPi



### ► 4. Using wiringPi [Python]: LED Blinking:

- Create a blink.py file in "> cd ~/ros\_catkin\_ws/src/blink":

```
> cd ~/ros_catkin_ws/src/blink  
> nano blink.py
```

- And add this code to the blink.py file:

```
import wiringpi  
import time  
wiringpi.wiringPiSetup()  
wiringpi.pinMode(1,1)  
while True:  
    time.sleep(0.5)  
    wiringpi.digitalWrite(1,1)  
    time.sleep(0.5)  
    wiringpi.digitalWrite(1,0)
```

- To run blink.py: (sudo is not required to run python codes)

```
> python blink.py
```

## D. Controlling RPi's GPIO: ROS & wiringPi

---

- ▶ Using wiringPi in a ROS node:
  - ▶ Used to make RPi with ROS has full access and control over its GPIO pins.
  - ▶ To compile a C++ node that uses wiringPi library, just add the work "wiringPi" to the CMakeLists.txt file of the package at "target\_link\_libraries"
  - ▶ To run a ROS node that uses wiringPi and access the hardware GPIO, you may need a root user privilege.
    - ▶ To do so, make the terminal run into the root user using "su" command, and if asked, enter your password (mostly "raspberrypi"), then run the ROS node.

```
> sudo su  
> rosruncat ros_001 diffBot
```



If the root password is not set, set it using the command, and enter desired pass

```
> sudo passwd root
```

# Real-Time Robot using RPi

## diffBot package control RPi based robot

---



- ▶ Download the ros\_001 package:
  - ▶ First download and make the teleop\_twist\_keyboard as we will use it

```
> cd ~/ros_catkin_ws/src  
> git clone https://github.com/ros-teleop/teleop\_twist\_keyboard  
> cd ..  
> catkin_make --pkg teleop_twist_keyboard
```

# Real-Time Robot using RPi

## diffBot package control RPi based robot

---



- ▶ Download the ros\_001 package:
  - ▶ Download the package from the github repository to get the installation shell script at any location

```
> mkdir ~/ros_001_source  
> cd ~/ros_001_source  
> git clone https://github.com/mina-sadek/ROS-dev2.git  
> cd ROS-dev2
```

- ▶ Then, make “ros\_001\_install.sh” executable by running:

```
> chmod a+x ./ros_001_install.sh
```

- ▶ Then, run it by just calling it

```
> ./ros_001_install.sh
```

# Real-Time Robot using RPi

## diffBot package control RPi based robot

---



- ▶ Install and run the ros\_001 package:
  - ▶ After finishing building, run the package launcher to start your robot.
    - ▶ First enter the root shell and enter the root password

```
> su  
> source /home/pi/.bashrc
```

- ▶ Note: The root password is not set by default, to set it use:

```
> sudo passwd root
```

Then enter your new password twice.

- ▶ After entering the root shell, run your launcher

```
> roslaunch ros_001 ros_001_launcher.launch
```

## E. ROS Communication with Arduino [ROS Serial]

---

### ► ROS Serial:

- Used to make RPi with ROS installed communicate with Arduino.
- 1. Setup arduino environment on the developing PC

```
> sudo apt-get install arduino arduino-core ros-melodicrosserial ros-melodic-rosserial-arduino
```



## Using Arduino with RPi using ROSSERIAL

---

- ▶ Setup roserial for Arduino IDE - on the Desktop machine

- ▶ To install the roserial library on the Arduino IDE

- ▶ 1. Install roserial for Arduino by running:

```
> sudo apt-get install ros-melodic-roserial-arduino  
> sudo apt-get install ros-melodic-roserial
```

- ▶ 2.1. Install ros\_lib into the Arduino Environment

```
> cd ~/Arduino/libraries  
> rm -rf ros_lib  
> rosrunc roserial_arduino make_libraries.py .
```

- ▶ 2.2. Another method is to install roserial only from library manager in Arduino:

- ◻ Just open the Library Manager from the IDE menu in Sketch -> Include Library -> Manage Library. Then search for "roserial". This is useful if you need to work on an Arduino sketch but don't want to setup a full ROS workstation.

- ▶ Setup roserial - on RPi

- ▶ Method 1:

- ▶ 1. Install roserial for Arduino by running:

```
> sudo apt-get install ros-kinetic-roserial-arduino  
> sudo apt-get install ros-kinetic-roserial
```

- ▶ Method 2:

```
> cd ~/ros_catkin_ws/src  
> git clone https://github.com/ros-drivers/roserial.git  
> cd ..  
> catkin_make
```

## Using Arduino with RPi using ROSSERIAL

---

- ▶ Running listener node of the Arduino node - on Desktop or RPi
  - ▶ 1. Upload an example project “Hello World!” to the Arduino, you can find it in the IDE in File -> Examples -> roserial Arduino library -> helloworld
    - ▶ Then connect your arduino board to the Desktop or the RPi, and on the host device run the roscore in a terminal

```
> roscore
```

- ▶ Then run the roserial client application that forwards you areduino messages to the rest of ROS in a separate terminal:

```
> rosrn roserial_python serial_node.py /dev/ttyACM0
```

## Using Arduino with RPi using ROSSERIAL

---

- ▶ Running listener node of the Arduino node - on Desktop or RPi
  - ▶ If you want to echo the message sent from the arduino us the simple echo form in a terminal run:

```
> rostopic echo /topic_name
```

- ▶ To publish a simple message from the terminal use:

```
> rostopic pub topic_name msg_type msg
```

-  E.g. to send integer message to toggle a LED use:

```
> rostopic pub toggle_led std_msgs/UInt16 1
```

# Installing ROS Kinetic on Linux Ubuntu Desktop

---

- ▶ For Linux UBUNTU 16.04 and later, ROS Kinetic is the recommended distribution.
- ▶ To install ROS Kinetic distro. on Linux Ubuntu on your desktop machine, follow these steps
  - ▶ 1. Configure your Ubuntu repositories to allow "restricted," "universe," and "multiverse". Follow the guide available in this link:
    - ▣ <https://help.ubuntu.com/community/Repositories/Ubuntu>
  - ▶ 2. Setup ubuntu sources.list
 

```
> sudo sh -c 'echo "deb http://packages.ros.org/ros/ubuntu $(lsb_release -sc) main" > /etc/apt/sources.list.d/ros-latest.list'
```
  - ▶ 3. Set up ROS indigo keys
 

```
> sudo apt-key adv --keyserver hkp://ha.pool.sks-keyservers.net:80 --recv-key 421C365BD9FF1F717815A3895523BAEEB01FA116
```

    - ▣ If a problem happened, try replacing hkp://ha.pool.sks-keyservers.net:80 with hkp://pgp.mit.edu:80 or hkp://keyserver.ubuntu.com:80

# Installing ROS Kinetic on Linux Ubuntu Desktop

- ▶ To install ROS Kinetic distro. on Linux Ubuntu on your desktop machine, follow these steps

- ▶ 4. Installing ROS Kinetic Desktop-Full

```
> sudo apt-get update
> sudo apt-get install ros-kinetic-desktop-full
```

- ☐ To find available packages use

```
> apt-cache search ros-kinetic
```

- ▶ 5. Initialize rosdep

```
> sudo rosdep init
> rosdep update
```

- ▶ 6. Environment setup

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

# Installing ROS Kinetic on Linux Ubuntu Desktop

---

- ▶ To install ROS Kinetic distro. on Linux Ubuntu on your desktop machine, follow these steps
  - ▶ 7. Getting rosinstall

```
> sudo apt-get install python-roscpp python-roscpp-generator python-wstool build-essential
```

# Installing ROS indigo on Linux Ubuntu Desktop

---

- ▶ For Linux UBUNTU 14.04 and earlier, ROS Kinetic is the recommended distribution.
- ▶ To install ROS indigo distro. on Linux Ubuntu on your desktop machine, follow these steps
  - ▶ 1. Configure your Ubuntu repositories to allow "restricted," "universe," and "multiverse". Follow the guide available in this link:
    - ☐ <https://help.ubuntu.com/community/Repositories/Ubuntu>
  - ▶ 2. Setup ubuntu sources.list
 

```
> sudo sh -c 'echo "deb http://packages.ros.org/ros/ubuntu $(lsb_release -sc) main" > /etc/apt/sources.list.d/ros-latest.list'
```
  - ▶ 3. Set up ROS indigo keys
 

```
> sudo apt-key adv --keyserver hkp://ha.pool.sks-keyservers.net:80 --recv-key 421C365BD9FF1F717815A3895523BAEEB01FA116
```

    - ☐ If a problem happened, try replacing hkp://ha.pool.sks-keyservers.net:80 with hkp://pgp.mit.edu:80 or hkp://keyserver.ubuntu.com:80



# Installing ROS indigo on Linux Ubuntu Desktop

---

- ▶ To install ROS indigo distro. on Linux Ubuntu on your desktop machine, follow these steps

- ▶ 4. Installing ROS indigo Desktop-Full

```
> sudo apt-get update
> sudo apt-get install ros-indigo-desktop-full
```

- ☐ To find available packages use

```
> apt-cache search ros-indigo
```

- ▶ 5. Initialize rosdep

```
> sudo rosdep init
> rosdep update
```

- ▶ 6. Environment setup

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

## Installing ROS indigo on Linux Ubuntu Desktop

---

- ▶ To install ROS indigo distro. on Linux Ubuntu on your desktop machine, follow these steps
  - ▶ 7. Getting rosinstall

```
> sudo apt-get install python-roscpp python-roscpp-generator python-wstool build-essential
```

# Useful ROS Commands

---

- ▶ To create a new package:

```
> catkin_create_pkg package_name roscpp rospy std_msgs
```

- ▶ To run the Master node in a separate terminal:

```
> roscore
```

- ▶ To source the catkin\_ws to the current terminal:

```
> source ~/catkin_ws/devel/setup.bash
```

- ▶ To source the catkin\_ws permanently, source it in the bash.rc file found at ~/.bashrc , add the following line:

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

- ▶ To run a node:

```
> rosrn package_name node_name
```

---



# Useful ROS Commands

---

- ▶ To list the current available topics, open a new terminal:

```
> rostopic list
```

- ▶ To monitor the data published by a topic:

```
> rostopic echo /topic_name
```

- ▶ To find all information about the topic:

```
> rostopic info /topic_name
```

- ▶ To list the current available nodes:

```
> rosnode list
```

- ▶ To list the current available nodes on a certain machine on the network, working on the same Master node:

```
> rostopic machine machine_name
```

- ▶ To find all information about a node:

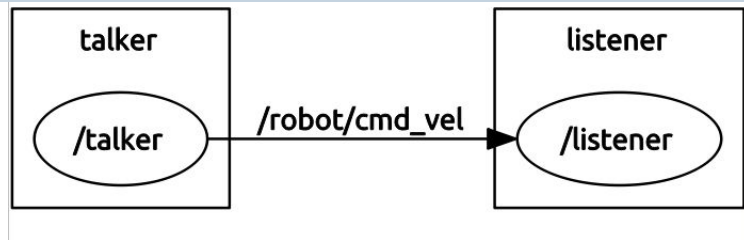
```
> rosnode info /node_name
```

# Useful ROS Commands

---

- ▶ To visualize how the nodes and topics running on the current ROS Master are related:

```
> rosrun rqt_graph rqt_graph
```



- ▶ To launch a .launch file:

```
> roslaunch package_name filename.launch
```

# Useful ROS Commands

---

- ▶ To create a topic and publish a message on it from the terminal, without creating a node:

```
> rostopic pub -1 /topic_name message_type 'message'
```

- ▶ Example on the cmd\_vel message:

```
> rostopic pub -r 10 /diffBot/cmd_vel geometry_msgs/Twist '{linear: {x: 500, y: 0.0, z: 0.0},  
angular: {x: 0.0,y: 0.0,z: 0.0}}'
```

# Useful ROS Commands

---

- ▶ To Install a package from github

```
> cd ~/ros_catkin_ws/src
```

- ▶ Then, clone the repository using:

- ▶ > git clone -b <branch> <address>

```
> git clone -b indigo-devel https://github.com/ros/common_msgs.git
```

- ▶ Once it is completed, catkin\_make the workspace

```
> cd ~/ros_catkin_ws
```

```
> catkin_make
```

- ▶ Then, source the setup.bash again

```
> source ~/catkin_ws/devel/setup.bash
```

```
> source ~/.bashrc
```

```
> cd ~/ros_catkin_ws
```

```
> catkin_make
```

# Useful ROS Packages

---

## ▶ 1. common\_msgs

```
> cd ~/ros_catkin_ws/src
> git clone -b indigo-devel https://github.com/ros/common_msgs.git
> cd ~/ros_catkin_ws
> catkin_make
```

## ▶ 2. teleop\_twist\_keyboard

```
> cd ~/ros_catkin_ws/src
> git clone -b indigo-devel https://github.com/ros-teleop/teleop_twist_keyboard.git
> cd ~/ros_catkin_ws
> catkin_make
```

## ▶ 3. tf

```
> cd ~/ros_catkin_ws/src
> git clone -b indigo-devel https://github.com/ros/geometry.git
> cd ~/ros_catkin_ws
> catkin_make
```



# Useful ROS Packages

---

## ▶ 4. Image processing and other packages

```
> sudo apt-get install ros-indigo-usb-cam ros-indigo-mavlink ros-indigo-mavros  
ros-indigo-cv-bridge ros-indigo-image-proc
```

## ▶ 5. Find which ros version is installed over the system:

```
> rosversion -d
```

# Useful ROS Commands

---

- ▶ To make only certain packages, and now the whole workspace:

```
> catkin_make --pkg pkg_1
```

# Troubleshooting

- ▶ If a problem happened with autocomplete
  - ▶ Add “ export LC\_ALL="C" ” to ~/.bashrc :

```
> echo " export LC_ALL="C"" >> ~/.bashrc
> source ~/.bashrc
```

OR set system locale by running:

```
> sudo update-locale LANG=C LANGUAGE=C LC_ALL=C LC_MESSAGES=POSIX
```

OR

```
> echo "export LC_ALL=C; unset LANGUAGE" >> ~/.bashrc
> source ~/.bashrc
```

[https://github.com/ros/common\\_msgs/tree/indigo-devel](https://github.com/ros/common_msgs/tree/indigo-devel)

<https://github.com/ros/geometry>

[https://github.com/orocos/orocos\\_kinematics\\_dynamics/blob/master/.travis.yml](https://github.com/orocos/orocos_kinematics_dynamics/blob/master/.travis.yml)

# Useful General Linux Commands

---

- ▶ Linux bash history with PageUp / PageDown

This helps to get old commands run on the terminal

```
> sudo nano /etc/inputrc
```

- ▶ Then uncomment

```
# alternate mappings for "page up" and "page down" to search the history
"\e[5~": history-search-backward
"\e[6~": history-search-forward
```

- ▶ Then, restart the shell, or tell the current shell to re-read the inputrc file by running ctrl+x then ctrl+r
- ▶ Now, just write start of your old command and press PageUp or PageDown to cycle the old commands starting by the one you wrote.

# Backup & Restore SD Card

- ▶ To backup your SD card, we take image of it to the hard disk.
  - ▶ 1. Use the next command to backup your sd card which is located at “/dev/sdb”

```
> sudo dd if=/dev/sdb of=./jessie_indigo.img
```

- ▶ If the image is much large than the actually contained data, it can be truncated

- ☐ Use this to find the actual data size

```
> fdisk -lu jessie_indigo.img
```

- ☐ Now, truncate the image using number of sectors which is the value of the “End” sector number plus “1”

```
> truncate --size=$((5872026+1)*512) image.img
```

- ☐ A better and complete way to truncate the image using GParted app. can be found in this link:

<https://softwarebakery.com//shrinking-images-on-linux>

# Backup & Restore SD Card

---

- ▶ To restore the image on a SD card, follow the same instructions of restoring any image to the SD card:

```
> umount of=/dev/sdb1
> umount of=/dev/sdb2
> sudo dd if=./jessie_indigo.img of=/dev/sdb
```

# Sample Nodes



## Publisher Node:

```
#!/usr/bin/env python
import rospy
from std_msgs.msg import String
def simplePublisher():
    pub = rospy.Publisher('/topic_1', String, queue_size=10)
    rospy.init_node('node_1', anonymous=False)
    rate = rospy.Rate(1) # 1hz

    # The string to be published on the topic.
    topic1_content = "my first ROS topic"
    while not rospy.is_shutdown():
        pub.publish(topic1_content)
        rate.sleep()
if __name__ == '__main__':
    try:
        simplePublisher()
    except rospy.ROSInterruptException:
        pass
```

Source: <https://thepihut.com/blogs/raspberry-pi-tutorials/17789160-backing-up-and-restoring-your-raspberry-pis-sd-card>

<https://raspberrypi.stackexchange.com/questions/7177/image-of-a-16gb-card-containing-unpartitioned-space-at-the-end-truncating-poss>

# Sample Nodes

---



## Subscriber Node:

```
## Node to subscribe to a string and print the string on terminal.
#!/usr/bin/env python
import rospy
from std_msgs.msg import String

# Topic callback function.
def stringListenerCallback(data):
    rospy.loginfo(' The contents of topic1: &s', data.data)

def stringListener():
    rospy.init_node('node_2' , anonymous = False)

    rospy.Subscriber('topic_1' , String, stringListenerCallback)

# spin() simply keeps python from exiting until this node is stopped
rospy.spin()

if __name__ == '__main__':
    stringListener()
```

Source: <https://thepihut.com/blogs/raspberry-pi-tutorials/17789160-backing-up-and-restoring-your-raspberry-pis-sd-card>

<https://raspberrypi.stackexchange.com/questions/7177/image-of-a-16gb-card-containing-unpartitioned-space-at-the-end-truncating-poss>



## Resources

---

- ▶ ROS Home Page
  - ▶ <http://wiki.ros.org/>
- ▶ ROS Concepts
  - ▶ <http://wiki.ros.org/ROS/Concepts>
- ▶ ROS indigo installation on UBUNTU 14.04 LTS
  - ▶ <http://wiki.ros.org/indigo/Installation/Ubuntu>
- ▶ ROS indigo installation on Raspberry Pi
  - ▶ <http://wiki.ros.org/ROSberryPi/Installing%20ROS%20Indigo%20on%20Raspberry%20Pi>
- ▶ Running ROS across multiple machines
  - ▶ <http://wiki.ros.org/ROS/Tutorials/MultipleMachines>

# Resources

---

- ▶ ROS Serial

- ▶ <http://wiki.ros.org/rosterial>

- ▶ ROS & Arduino

- ▶ [http://wiki.ros.org/rosterial\\_arduino/Tutorials](http://wiki.ros.org/rosterial_arduino/Tutorials)

- ▶ Useful ROS Tutorial

- ▶ <http://wiki.ros.org/Courses>
  - ▶ <http://clearpathrobotics.com/guides/ros/Intro%20to%20the%20Robot%20Operating%20System.html>
  - ▶ <http://www.rsl.ethz.ch/education-students/lectures/ros.html>

-  <http://barzinm.com/robotics/2016/odriod-setup.html>
- 



# Resources

---

- ▶ ROS-dev2
  - ▶ <https://github.com/mina-sadek/ROS-dev2>

# Resources

---

## ► Useful links

- <http://www.venelinpetkov.com/how-to-install-ros-robot-operating-system-on-raspberry-pi-3-with-raspbian-stretch/>
- <https://neverbenever.wordpress.com/2017/12/20/install-ros-and-opencv-in-raspbian-stretch/>
- <http://barzinm.com/robotics/2016/odroid-setup.html>
- [https://github.com/olinrobotics/Odroid\\_Setup/wiki/Odroid-and-ROS](https://github.com/olinrobotics/Odroid_Setup/wiki/Odroid-and-ROS)
-