FREE eBook

LEARNING

ros

Free unaffiliated eBook created from
Stack Overflow contributors.

#ros

Table of Contents

About

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: ros

It is an unofficial and free ros ebook created for educational purposes. All the content is extracted from Stack Overflow Documentation, which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official ros.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

Chapter 1: Getting started with ros

Remarks

Robotic Operating System (ROS) is a Robotics Middleware for robots software development providing operating-system like functionalities on heterogeneous computer clusters and platforms.

Originally started in 2007 by the Stanford Artificial Intelligence Laboratory in the support of the Stanford AI Robot STAIR, development, from 2008 to 2013, migrated to be performed at Willow Garage, a robotics research institute. In 2013, ROS stewardship transitioned to the Open Source Robotics Foundation.

ROS provides libraries and tools to help software developers create robot applications. It provides hardware abstraction, device drivers, libraries, visualizers, message-passing, package management, and more. ROS is licensed under an open source, BSD license.

Today, ROS integrates more than a hundred of robots (see full robots list), ranging from autonomous cars, to UAVs to humanoid robots and using a multitude of ROS supported sensors (see complete sensors list)... ROS is heavily utilised by the research community for service robotics applications, but its technology can be applied to other application areas, including industrial robotics. Its applications such as advanced perception, path/grasp planning, motion tracking can enable manufacturing robotic applications that were previously technically infeasible or cost prohibitive.

ROS currently only runs on Unix-based platforms. Software for ROS is primarily tested on Ubuntu and Mac OS X systems, though the ROS community has been contributing support for Fedora, Gentoo, Arch Linux and other Linux platforms. Eventually, ROS coding can be written in any programming language provided it has it's client library, though, the current focus is on providing strong C++ and Python Support.

ROS today presents it's 10th release *ROS Kinetic*.

To find more about ROS and ROS Community efforts, visit http://www.ros.org/

Versions

Ros Distro	Supported Ubuntu Versions	Release Date
Kinetic Kame	15.10, 16.04	2016-05-23
Jade Turtle	14.04, 14.10, 15.04	2015-05-23
Indigo Igloo	13.10, 14.04	2014-07-22
Hydro Medusa	12.04, 12.10, 13.04	2013-09-04
Groovy Galapagos	11.10, 12.04, 12.10	2012-12-31

Ros Distro	Supported Ubuntu Versions	Release Date
Fuerte Turtle	10.04, 11.10, 12.04	2012-04-23
Electric Emys	10.04, 10.10, 11.04, 11.10	2011-08-30
Diamondback	10.04, 10.10, 11.04	2011-03-02
C Turtle	9.04, 9.10, 10.04, 10.10	2010-08-02
Box Turtle	8.04	2010-03-02

Examples

Installation

Depending on your target machine, you need to choose a supported ROS Version (or vice-versa). Although ROS installation is well documented in the ROS wiki, It might be confusing to find them. So, here's a table of the ROS Version, target platforms & architecture and the links for the appropriate install guides :

ROS Version	Platform	Arch	Status	Install Guide Link
Kinetic	Ubuntu 16.04 (Xenial)	amd64 / i386 / armhf	Supported	Kinetic-Xenial-guide
	Ubuntu 15.10 (Wily)	amd64 / i386	Supported	Kinetic-Wily-guide
	Debian 8 (Jessie)	amd64 / arm64	Supported	Kinetic-Jessie-guide
	OS X (Homebrew)	--	Experimental	Kinetic-Homebrew-guide
	Gentoo	--	Experimental	Kinetic-Gentoo-guide
	OpenEmbedded/Yocto	--	Experimental	Kinetic-Yocto-guide

Work in progress...!

Hello World Publisher

Create a workspace

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

```
catkin_init_workspace
```

Build your workspace

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

Source your setup file

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

Create a new package named hello_world with some basic dependencies

```
catkin_create_pkg hello_world std_msgs rospy roscpp
```

Navigate to your src directory and create a new file called talker.cpp

```
cd hello_world/src
touch talker.cpp
```

Edit your new file and paste this code in to publish a "hello world" message

```
#include "ros/ros.h"
#include "std_msgs/String.h"

#include <sstream>

int main(int argc, char **argv)
{
  ros::init(argc, argv, "talker");

  ros::NodeHandle n;

  ros::Publisher chatter_pub = n.advertise<std_msgs::String>("chatter", 1000);

  ros::Rate loop_rate(10);

  int count = 0;
  while (ros::ok())
  {
    std_msgs::String msg;

    std::stringstream ss;
    ss << "hello world " << count;
    msg.data = ss.str();

    ROS_INFO("%s", msg.data.c_str());

    chatter_pub.publish(msg);

    ros::spinOnce();

    loop_rate.sleep();
    ++count;
  }
```

```
  return 0;
}
```

Return to the root of your package directory

```
cd ..
```

Add/uncomment these lines to your CMakeLists.txt

```
catkin_package(
 INCLUDE_DIRS include
 LIBRARIES hello_world
#  CATKIN_DEPENDS roscpp rospy std_msgs
#  DEPENDS system_lib
)

include_directories(include ${catkin_INCLUDE_DIRS})

add_executable(talker src/talker.cpp)
target_link_libraries(talker ${catkin_LIBRARIES})
add_dependencies(talker hello_world_generate_messages_cpp)
```

Return to the root of your workspace

```
cd ..
```

Build your new publisher

```
catkin_make
```

Source your setup file again so that you have the new package and publisher

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

Start ROS

```
roscore
```

Leave roscore running and in a new terminal tab/window, start your publisher

```
rosrun hello_world talker
```

Leave the publisher running and in ANOTHER new terminal tab/window, echo the output

```
rostopic echo /chatter
```

Read Getting started with ros online: https://riptutorial.com/ros/topic/7287/getting-started-with-ros

Chapter 2: Creating a package

Introduction

This tutorial shows how to create a package in ROS. Packages sit inside workspaces, in the src directory. Each package directory must have a `CMakeLists.txt` and a `package.xml` files.

Examples

Creating a package using rospy

Assuming a workspace named `workspace_name` has been previously created in the home directory, a package named `package_name` can be created by executing the following command lines.

```
$ cd ~/workspace_name/src/
$ catkin_create_pkg package_name rospy
```

Read Creating a package online: https://riptutorial.com/ros/topic/8314/creating-a-package

Chapter 3: creating a workspace

Introduction

This tutorial shows how to create a workspace. A workspace is a set of directories in which a related set of ROS code lives. One can have multiple ROS workspaces, but it is possible to work only in one at a time.

Examples

Creating a Workspace

In order to create a workspace, one should run the following in the terminal:

```
$ mkdir -p ~/workspace_name/src
$ cd ~/workspace_name/src
$ catkin_init_workspace
$ cd ~/workspace_name/
$ catkin_make
```

The previous commands creates a workspace named `workspace_name`. Once a workspace has been created, it is important to source it in order to work with it:

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

Read creating a workspace online: https://riptutorial.com/ros/topic/8313/creating-a-workspace

Chapter 4: roslaunch

Remarks

'node form the package' should be 'node from the package'

initially your say "starting" and "Stopping", but you don't explain how to do a controlled shutdown.

Examples

launch ros nodes and load parameters from Yaml file

roslaunch is an important tool that manages the start and stop of ROS nodes. It takes one or more "*.launch" files as arguments.

For this example, I will refer to the following (as asked in this question), so how can we execute those commands consecutively & automatically :

```
roscd stereo_camera
rosparam load marvin_cameras.yaml
rosrun stereo_camera stereo_camera __name:=bumblebeeLeft
rosrun stereo_camera stereo_camera __name:=bumblebeeCenter

roslaunch openni_launch_marvin kinect_left.launch
roslaunch openni_launch_marvin kinect_center.launch
```

First of all, let's `break up these commands in pieces`. As it can be seen, 4 ros commands are needed : *roscd*, *rosparam*, *rosrun* and *roslaunch*. Now let's start !

This is an example of why roslaunch is powerful, In fact, all those commands could be included in `one and only roslaunch file in ROS`. then all what we have to do is to launch this "solution.launch" file for a consecutive and automatic call for those commands.

To start, launch files are based on XML formatting, here's a basic launch file in ROS, we will name it "basic_example.launch" and it's included in a ROS package named "roslaunch_example":

```
<launch>

</launch>
```

the command to execute this launch file is

```
$ roslaunch roslaunch_example basic_example.launch
```

following the specification :

```
$ roslaunch package_name launch_file_name.launch
```

Because of our launch file doesn't include any commands, nothing will be executed, more on that later...

Including nodes in Launch files :
any ROS node in any ROS package installed is call-able in launch files. to that we have to specify the package containing the node and it's name as specified in the package. for example :

```
rosrun stereo_camera stereo_camera __name:=bumblebeeLeft
rosrun stereo_camera stereo_camera __name:=bumblebeeCenter
```

`stereo_camera` is a node form the package `stereo_camera` and the arguments specified are it's name `__name:=bumblebeeLeft` and `__name:=bumblebeeCenter`.

To add those nodes, we have to add the following lines:

```
<launch>
    <node name="$(arg name)" pkg="stereo_camera" type="stereo_camera" output="screen">
          <param name="name" value="bumblebeeLeft" />
    </node>

    <node name="$(arg name)" pkg="stereo_camera" type="stereo_camera" output="screen">
          <param name="name" value="bumblebeeCenter" />
    </node>
</launch>
```

Executing this launch file, we'll have the two nodes running.

adding a parameter to a ROS node :

As can be seen, we added a parameter "name" for the nodes as :

```
<param name="name" value="bumblebeeCenter" />
```

In fact, we can add as much parameters as we want (as created above) and then refer to them by calling `"$(arg parameter_name)"` instead of fixing it's value.

Specifying the output :

The output tag can be set to "screen", if you need to see the node log `on the terminal` or "log" to save the log to the log files in `(~/.ros)`.

Including other ROS launch files in a ROS launch file :

As stated here, The tag enables you to import another roslaunch XML file into the current file. It will be imported within the current scope of your document, including and tags. All content in the include file will be imported except for the tag: the tag is only obeyed in the top-level file.

So, all to executes these commands

```
roslaunch openni_launch_marvin kinect_left.launch
roslaunch openni_launch_marvin kinect_center.launch
```

from a launch file, all we have to do is adding the following lines :

```
<include file="$(find openni_launch_marvin)/launch/kinect_left.launch" />
<include file="$(find openni_launch_marvin)/launch/kinect_center.launch" />
```

roscd to a package in ROS launch file

In order to find the launch file than we want to include, we don't need to specify the full path. Instead, roslaunch provides the `"$(find package_name)"` directive, this way, we can refer to our launch file `relative to the package racine`.
In the above example, I assumed that the file "kinect_center.launch" is in the "openni_launch_marvin)/launch/" folder.

Loading parameters from YAML File :

In order to load parameters from a YAML file in ROS, ROS provides the "rosparam" tag. As stated in the wiki : "The tag enables the use of rosparam YAML files for loading and dumping parameters from the ROS Parameter Server. It can also be used to remove parameters. The tag can be put inside of a tag, in which case the parameter is treated like a private name."

Using this tag, we can load our YAML file in the launch file by adding this line :

```
<rosparam command="load" file="$(find marvin_cameras)/config/marvin_cameras.yaml" />
```

As used above, I assumed that the YAML file "marvin_cameras.yaml" is in the "marvin_cameras/config/" folder.

Assembling all pieces

Now that we have created separately our launch file contents, let's assemble them in one big launch file "solution.launch".

solution.launch

```
<launch>

    <rosparam command="load" file="$(find marvin_cameras)/config/marvin_cameras.yaml" />

    <node name="$(arg name)" pkg="stereo_camera" type="stereo_camera" output="screen">
          <param name="name" value="bumblebeeLeft" />
    </node>

    <node name="$(arg name)" pkg="stereo_camera" type="stereo_camera" output="screen">
          <param name="name" value="bumblebeeCenter" />
    </node>

    <include file="$(find openni_launch_marvin)/launch/kinect_left.launch" />
    <include file="$(find openni_launch_marvin)/launch/kinect_center.launch" />

</launch>
```

Now, we have our one and only roslaunch file for executing all the commands consecutively and

automatically.

Read roslaunch online: https://riptutorial.com/ros/topic/7361/roslaunch

Credits

S. No	Chapters	Contributors
1	Getting started with ros	ckirksey3, Community, Photon, Vtik
2	Creating a package	lmiguelvargasf, Michael
3	creating a workspace	lmiguelvargasf
4	roslaunch	ensonic, Vtik