Tutorial 1: The Robot Operating System: Basics 1

The F1/10 Team

Introduction

This article contains the transcript of the Video Tutorial 1: The Basics of ROS (Part 1). This tutorial is a collection of the the first few tutorials covered by the official ROS tutorials. The aim of this tutorial is to familiarize you with the framework of the Robot Operating System. You will be introduced to some visualizations and you will learn how to navigate within the ROS file system. You will also be downloading and launching some basic executables. This tutorial covers aspects from tutorials - 2,3,4,5,6,8 in the 'Beginner level' ROS tutorials [**?**].

Transcript of Video Tutorial

In this tutorial we will provide an introduction to the ROS Filesystem, run some example programs and also implement basic ROS code. This follows the ROS Indigo Installation on ubuntu 14.04. The organizational hierarchy in ROS can be described with 3 broad layers.

- The lower end of the hierarchy is the node. Nodes are programs and scripts - your C++/python code. Most of the times, a collection of nodes are required to implement a larger goal.
- These nodes are kept in the middle layer of the hierarchy - called a ROS package
- A collection of packages is kept in a container called the workspace - the top most layer of the hierarchy

Think of this setup in terms of our lab. The workspace will be the lab itself, a package can be a particular action like drinking coffee or printing and each package will contain related executables like a cup, coffee machine or a A4 size paper. We will now go through each layer familiarizing ourselves with few useful ROS commands and tools.

Install ROS using the Tutorial 0 installation guide. Once it is done make sure you have the ROS environemnt sourced by typing

```
$ echo $ROS_PACKAGE_PATH.
```

This essentially means that the location of the installed ROS components is stored in your environment. This lets you access installed ROS components from anywhere in your system.

Now lets run a example program called turtlesim which lets you control a turtle on the screen using your keyboard. First we need to initiate the ROS framework by running the command

```
$ roscore
```

It is the master that communicates with all ROS subsystems and provides an interconnect among them. It needs to be running in the background before any ros code can be launched. its like the clock in any microcontroller. Once we have roscore running, in another terminal or tab we run a new node called turtlesim_node from the turtlesim package by running

```
$ rosrun turtlesim turtlesim_node
```

Now u can see a stagnant turtle in the middle of an ocean. To control the turtle we need to run an other node that captures the keyboard strokes and sends it to our first node displaying the turtle. Start an other

terminal and run

```
$ rosrun turtlesim turtle_teleop_key
```

Make sure u have selected the turtle_teleop_key terminal and use the arrow keys to control the turtle.

Now that you run your first program in ROS lets look at whats happening behind the scenes. Lets see all the nodes running in our program. In a new terminal enter

```
$ rosnode list
```

We see that there are 3 nodes running. rosout is a logging mechanism that always runs by default. The other two nodes are the nodes that we started. A node publishes its data on channels called topics. These topics are predefined in code and carry data of the defined data type. We use ros methods like rostopic list, echo, info to examine topics and related entities.

```
$ rostopic list
```

Here, when we do a rostopic list, we see a number of /turtle1 topics that is active. /rosout is the topic that is active the moment you start roscore. Its responsible for logging ROS events/messages.

On performing a

```
$ rostopic info /turtle1/cmd_vel
```

we see the node that is responsible for publishing this topic. We also see that the message type that this topic carries is the Twist data type. This data type is provided for in the geometry_msgs which we will learn about more in the next tutorial.

To see the details of a particular msg, we use rosmsg show, and in this case,

```
$ rosmsg show geometry_msgs/Twist
```

We can view the raw data that is being published on this topic by performing

```
$ rostopic echo /turtle1/cmd_vel
```

Select the turtle_teleop_key window and use the arrow keys to see the data published in the topic.

Another tool that ROS provdes us is the rqt_graph tool. In another terminal, type in

```
$ rosrun rqt_graph rqt_graph
```

This tool shows us the various topics and nodes that are active and how they are communicating with each other. Right now the turtle_teleop_key node is publishing cmd_vel topic and the turtlesim_node has subscribed to it.

In the next tutorial, we will see how to write our own publisher and subsciber nodes.

References

[1] ROS Beginner Tutorials