Robotic Operating System (ROS) Basics

J Veejay Karthik, Deepak Mallya

Systems and Control
IIT Bombay

August 17, 2023

Robotic Operating System (ROS)

- ROS is an **open-source framework** designed for building robotic systems.
- It provides a collection of tools, libraries, and conventions to help developers create and manage complex robotic applications.
- ROS offers simulation tools like **gazebo** that allows testing and validating control and planning algorithms in virtual environments before deploying them on real robots.
- It has a large and active community of developers, researchers, and robotics enthusiasts.
- Several robotic companies provide ROS packages associated with their commercial robotic platforms for seamless interfacing and testing.

Fundamentals - ROS Nodes

- They are individual software modules for performing specific tasks within a robotic system.
- They can be programmed in different languages like Python, and they can run on different machines or the same machine.
- Capable of communicating with each other to transfer information.
- The **ROS Master** is an entity that **maintains a registry of active nodes**, allowing them to discover and communicate with each other.

Terminal Command

'roscore' - Invokes the ROS Master

- ROS Nodes can **publish (send out)** or **subscribe (receive)** information.

Fundamentals - ROS Topics

Question

How do ROS Nodes transfer information (data)?

- Information is transferred through channels are **ROS Topics**.
- ROS Nodes can publish data to a topic, and other nodes can subscribe to receive that data.
- They enable **asynchronous** communication, allowing nodes to operate independently and exchange information efficiently.
- They are identified by unique names - **'/topic_name'**

Fundamentals - ROS Messages

Question

What flows through ROS Topics?

- ROS messages define the **data structures** used for **communication between ROS nodes via ROS Topics.**
- They are defined in **'.msg'** files, which specify the structure of the data and its types.
- ROS Nodes that publish and subscribe to the same topic must use **compatible message types** for communication.
- ROS has inbuilt standard message types such as **'std_msgs/String'**, **'nav_msgs/Odometry'**, etc., and also provides provisions for describing custom message types for specific applications for enabling communication.

RQT Graphs

A graphical representation of how nodes are connected and how data flows between them.

Figure: An Simple RQT Graph

- A visualization tool provided by ROS to help developers understand the communication relationships between different nodes and topics within a ROS-based robotic system.
- They play a crucial role in debugging, monitoring, and understanding the behavior of a ROS system. (Terminal command - **'rqt_graph'**)

Basic System Setup - ROS

ROS Workspaces

- In simple terms, a ROS workspace refers to the directory where we store ROS projects.
- It helps in managing, building, updating and deploying ROS projects.
- Before we can start creating ROS nodes and topics, we need to configure a ROS workspace

The terminal commands to set up (create) a ROS Workspace are as follows,

```
$ mkdir –p ~/catkin_ws/src
$ cd ~/catkin_ws/src
$ catkin_init_workspace
$ cd ~/catkin_ws/
$ catkin_make
```

Activating the ROS Workspace

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

Creating a ROS Package (Project)

- **Navigate to the directory (folder) ~/catkin_ws/src** and run the command,

  $ catkin_create_pkg test_pub_sub rospy

- Create a **scripts** directory

  $ mkdir scripts

- Inside the **scripts** directory, create two python scripts as follows,

  $ touch listener.py talker.py

- Copy the contents of the corresponding files from the given git repository into these scripts.

- Make these scripts executable as follows,

  $ chmod +x talker.py listener.py

- Navigate to ~/catkin_ws, and run $ catkin_make