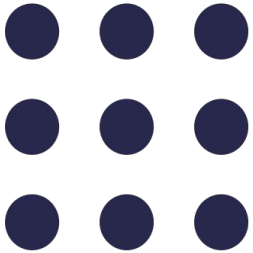
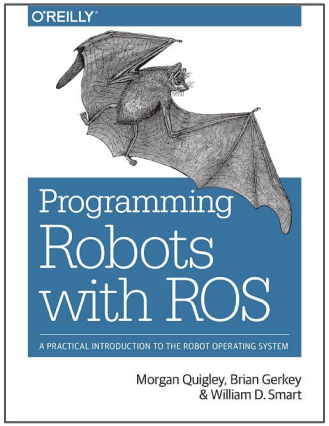


ROS Basics

Workspace, Packages;
Publishers & Subscribers;
Messages

Resources for the next few lectures...



Book for ROS:

Programming Robots with ROS - Morgan Quigley, Brian Gerkey, William D. Smart

Online Resources:

ROS Wiki Tutorials - http://wiki.ros.org/ROS/Tutorials

BITS Goa QSTP (2019):

https://github.com/hardesh/QSTP-Introduction_to_ROS/

Preliminaries

The ROS setup needs to be sourced on every terminal that uses ROS. This is done using the following command:

```
source /opt/ros/melodic/setup.bash
```

The *.bashrc* file in linux is executed each time a new terminal is opened. Hence, this line can be added to *.bashrc* to avoid running it each time a terminal is opened.

Also add the following line in *.bashrc* to configure the environment variable:

```
export ROS_PACKAGE_PATH=/opt/ros/melodic/share
```

Beginning with ROS:

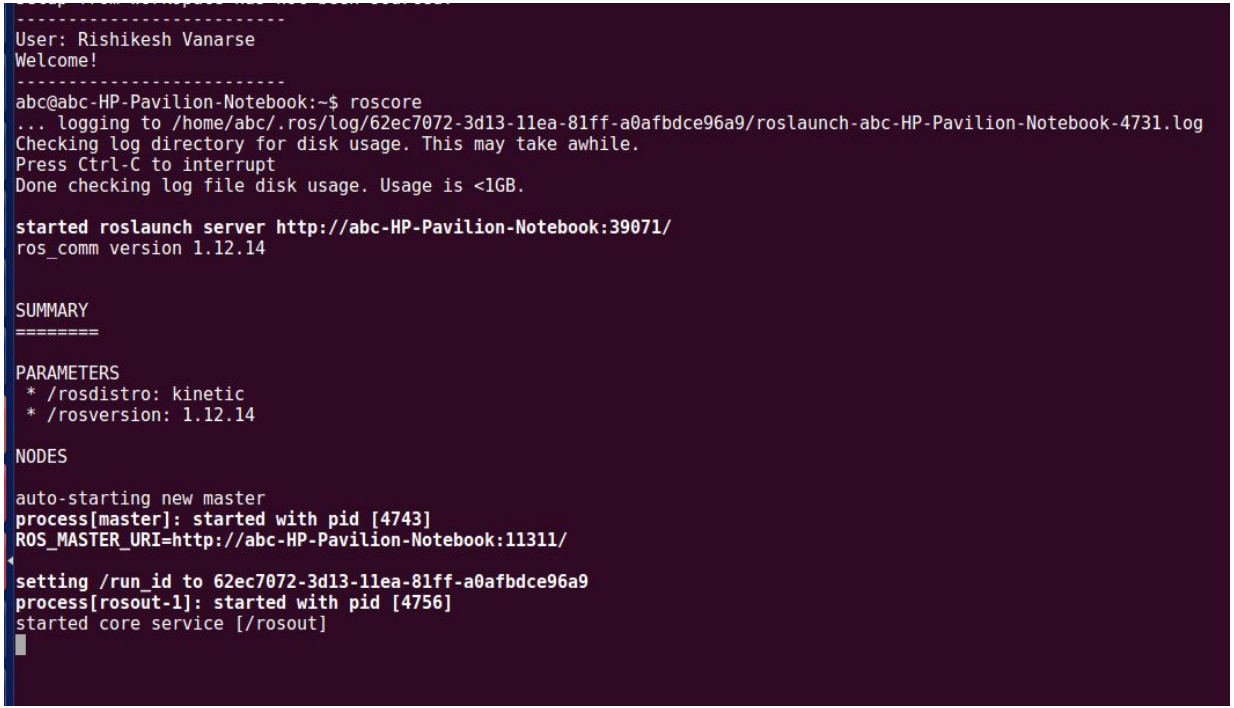
ROSMaster: A program that stores and broadcasts information about nodes and topics so that nodes can send messages to each other

To start working with ROS, the ROS Master has to be started by executing the following command:

```
roscore
```

The ROScore has to be kept running on one terminal while ROS is being used.

The roscore terminal



Finding Active nodes and topics:

Execute the following commands:

```
rostopic list
```

```
-------------------------
abc@abc-HP-Pavilion-Notebook:~$ rosnode list
/rosout
```

```
rostopic list
```

```
-------------------------
abc@abc-HP-Pavilion-Notebook:~$ rostopic list
/rosout
/rosout_agg
abc@abc-HP-Pavilion-Notebook:~$
```

ROS Workspace & catkin

- **Workspace:** A set of directories where a set of related ROS code lives. Only one workspace can be used at a time.
- Contains the following spaces:
  - Source Space (/src)
  - Development Space (/devel)
  - Build Space (/build)

- **Catkin:** The build system used by ROS to generate executables, libraries, interfaces, etc.

Creating a catkin Workspace

```
mkdir -p <workspace name>/src
```

```
cd <workspace name>/src
```

```
catkin_init_worspace
```

Building the Workspace

```
catkin_make
```

After opening any new terminal where the workspace is to be used, execute the following from the workspace folder:

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

Files in the Workspace

Explanation	Directory
C++ include headers	/include
Source files	/src
Folder containing Message (msg) types	/msg
Folder containing Service (srv) types	/srv
Folder containing launch files	/launch
The package manifest	package.xml
CMake build file	CMakeLists.txt

ROS Packages

- Contain the source files of nodes, along with a *package.xml* file
- **Creating a ROS Package**
  - Go to the */src* folder and execute the following command:

```
catkin_create_pkg <pkg_name> std_msgs rospy roscpp <other_dependencies>
```

Writing a Publisher Node in Python

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

import rospy

from std_msgs.msg import Int32


rospy.init_node('topic_publisher')

pub = rospy.Publisher('counter', Int32)

rate = rospy.Rate(2)

count = 0
while not rospy.is_shutdown():
    pub.publish(count)
    count += 1
    rate.sleep()
```

Running the Node

The following command can be used to run a node:

```
rosrun <package_name> <node_name>
```

To check the data being published on a topic, the following command is used:

```
rostopic echo <topic_name>
```

Writing a Subscriber in Python

```
import rospy
from std_msgs.msg import Int32


def callback(msg):
    print msg.data


rospy.init_node('topic_subscriber')

sub = rospy.Subscriber('counter', Int32, callback)

rospy.spin()
```

Message Types

ROS type	Serialization	C++ type	Python type	Notes
bool	Unsigned 8-bit integer	uint8_t	bool	
int8	Signed 8-bit integer	int8_t	int	
uint8	Unsigned 8-bit integer	uint8_t	int	uint8[] is treated as a string in Python
int16	Signed 16-bit integer	int16_t	int	
uint16	Unsigned 16-bit integer	uint16_t	int	
int32	Signed 32-bit integer	int32_t	int	
uint32	Unsigned 32-bit integer	uint32_t	int	
int64	Signed 64-bit integer	int64_t	long	
uint64	Unsigned 64-bit integer	uint64_t	long	
float32	32-bit IEEE float	float	float	
float64	64-bit IEEE float	double	float	
string	ASCII string	std::string	string	ROS does not support Unicode strings; use a UTF-8 encoding
time	secs/nsecs unsigned 32-bit ints	ros::Time	rospy.Time	duration

Predefined Message types

Getting the list of existing message types:

```
rosmsg list
```

Getting information on a particular message type

```
rosmsg info <message_name>
```

```
sound_play/SoundRequestFeedback
sound_play/SoundRequestGoal
sound_play/SoundRequestResult
std_msgs/Bool
std_msgs/Byte
std_msgs/ByteMultiArray
std_msgs/Char
std_msgs/ColorRGBA
std_msgs/Duration
std_msgs/Empty
std_msgs/Float32
std_msgs/Float32MultiArray
std_msgs/Float64
std_msgs/Float64MultiArray
std_msgs/Header
std_msgs/Int16
std_msgs/Int16MultiArray
std_msgs/Int32
std_msgs/Int32MultiArray
std_msgs/Int64
std_msgs/Int64MultiArray
std_msgs/Int8
std_msgs/Int8MultiArray
std_msgs/MultiArrayDimension
std_msgs/MultiArrayLayout
std_msgs/String
std_msgs/Time
std_msgs/UInt16
std_msgs/UInt16MultiArray
std_msgs/UInt32
std_msgs/UInt32MultiArray
std_msgs/UInt64
std_msgs/UInt64MultiArray
std_msgs/UInt8
std_msgs/UInt8MultiArray
stdr_msgs/CO2SensorMeasurementMsg
stdr_msgs/CO2SensorMsg
stdr_msgs/CO2Source
stdr_msgs/CO2SourceVector
stdr_msgs/DeleteRobotAction
stdr_msgs/DeleteRobotActionFeedback
stdr_msgs/DeleteRobotActionGoal
stdr_msgs/DeleteRobotActionResult
stdr_msgs/DeleteRobotFeedback
stdr_msgs/DeleteRobotGoal
stdr_msgs/DeleteRobotResult
stdr_msgs/FootprintMsg
stdr_msgs/KinematicMsg
stdr_msgs/LaserSensorMsg
stdr_msgs/Noise
stdr_msgs/RegisterRobotAction
stdr_msgs/RegisterRobotActionFeedback
stdr_msgs/RegisterRobotActionGoal
stdr_msgs/RegisterRobotActionResult
stdr_msgs/RegisterRobotFeedback
stdr_msgs/RegisterRobotGoal
stdr_msgs/RegisterRobotResult
stdr_msgs/RfidSensorMeasurementMsg
```

List of existing message types from all installed packages

```
Global Git Username configured successfully!


Exporting ROS Package path
ROS Path exported
Sourcing kinetic/setup.bash ...
Sourced kinetic setup.bash
Setup from workspace has not been sourced.
-------------------------
User: Rishikesh Vanarse
Welcome!
-------------------------
abc@abc-HP-Pavilion-Notebook:~$ rosmsg info std_msgs/
std_msgs/Bool                 std_msgs/Header               std_msgs/String
std_msgs/Byte                 std_msgs/Int16                std_msgs/Time
std_msgs/ByteMultiArray       std_msgs/Int16MultiArray      std_msgs/UInt16
std_msgs/Char                 std_msgs/Int32                std_msgs/UInt16MultiArray
std_msgs/ColorRGBA            std_msgs/Int32MultiArray      std_msgs/UInt32
std_msgs/Duration             std_msgs/Int64                std_msgs/UInt32MultiArray
std_msgs/Empty                std_msgs/Int64MultiArray      std_msgs/UInt64
std_msgs/Float32              std_msgs/Int8                 std_msgs/UInt64MultiArray
std_msgs/Float32MultiArray    std_msgs/Int8MultiArray       std_msgs/UInt8
std_msgs/Float64              std_msgs/MultiArrayDimension  std_msgs/UInt8MultiArray
std_msgs/Float64MultiArray    std_msgs/MultiArrayLayout
abc@abc-HP-Pavilion-Notebook:~$ rosmsg info std_msgs/Time
time data

abc@abc-HP-Pavilion-Notebook:~$ rosmsg info std_msgs/Float32MultiArray
std_msgs/MultiArrayLayout layout
  std_msgs/MultiArrayDimension[] dim
    string label
    uint32 size
    uint32 stride
  uint32 data_offset
float32[] data

abc@abc-HP-Pavilion-Notebook:~$
```

Definition of a particular message type
Eg: std_msgs/Float32MultiArray

Creating custom messages

- A message definition consists of a list of previously defined message types
- Message definitions lie in the ***msg*** folder in the package
- Once the message is defined, ***package.xml*** and ***CMakeLists*** need to be updated to enable building and using the message with ROS.
- *catkin_make* creates a Python/C++ class for newly defined message types.

Custom message example

(Example from Morgan Quigley Ex 3.3)

Creating a message type for a Complex Number

1. Create a folder called ***msg*** in the package and create a file named ***Complex.msg*** with the following content:

```
float32 real
float32 imaginary
```

Custom message example (continued)

2. For the *first* custom message of the package, make the following changes in ***CMakeLists***:

```
#1
find_package(catkin REQUIRED COMPONENTS
   roscpp
   rospy
   std_msgs
   message_generation
)
```

```
#2

catkin_package(
  CATKIN_DEPENDS
  message_runtime
)
```

Custom message example (continued)

3. For *every* custom message of the package, do the following changes in ***CMakeLists***

```
#1

add_message_files(
   FILES
   Complex.msg
)
```

```
#2

generate_messages(
   DEPENDENCIES
   std_msgs
)
```

Custom message example (continued)

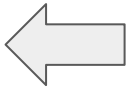
4. For the *first* custom message you build, add the following lines in ***package.xml***:

```
<build_depend>message_generation</build_depend>

<exec_depend>message_runtime</exec_depend>
```

5. Go back to the workspace folder and run `catkin_make`.

Using the new message

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

import rospy

from basics.msg import Complex

from random import random


rospy.init_node('message_publisher')

pub = rospy.Publisher('complex', Complex)

rate = rospy.Rate(2)

while not rospy.is_shutdown():
    msg = Complex()
    msg.real = random()
    msg.imaginary = random()

    pub.publish(msg)
    rate.sleep()
```



Python code for a publisher that publishes a random complex number every half second.

Example from Morgan Quigley (Ex 3.5)

** Replace **basics.msg** with **<your_package_name>.msg***

References & Further Reading:

Morgan Quigley - Chapter 1 to 3

ROSWiki Tutorials - Tutorials 1 to 13

Next Class - Services & Actions