Introduction to ROS

Ivan Marković Matko Orsag Damjan Miklić

Automation and Robotics
Robot Programming and Simulation

UNIVERSITY OF ZAGREB
Faculty of Electrical Engineering and Computing

Review

Before we proceed

- Navigating Linux filesystem, creating files and folders
- Searching for things, installing packages
- Python basics
- Install packages

Review

Before we proceed

- Navigating Linux filesystem, creating files and folders
- Searching for things, installing packages
- Python basics
- Install packages
- Install Desktop-Full ROS distro variant (recommended)
  wiki.ros.org/ROS/Installation

Review

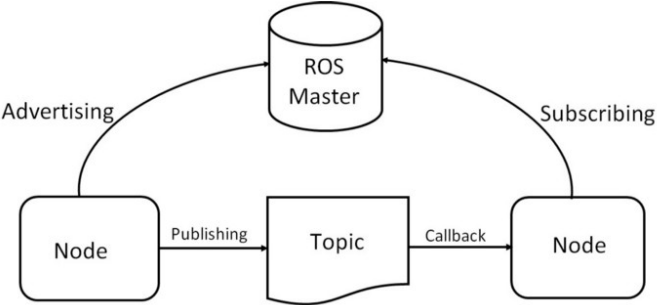
Before we proceed

- Navigating Linux filesystem, creating files and folders
- Searching for things, installing packages
- Python basics
- Install packages
- Install Desktop-Full ROS distro variant (recommended)
  wiki.ros.org/ROS/Installation
- Tell Linux which ROS distribution you want to use (optional)

```
$ source /opt/ros/<distro>/setup.bash
```

ROS: what is ROS?

- As meta operating system ROS provides a structured communications layer above the host operating systems of a heterogeneous compute cluster.



ROS: a robot computational graph

- A node is an executable that uses ROS to communicate with other nodes

```
$ roscore
$ rosrun turtlesim turtlesim_node &
$ rosrun rqt_graph rqt_graph &
```

- Nodes communicate by exchanging messages
- Messages are published and subscribed to on specific topics

```
$ rosrun turtlesim turtle_teleop_key
```

- ROS Master helps node find each other (Name service)
- ROS is an inter-process communication middleware

Command line tools

- rosnode

```
$ rosnode list
$ rosnode info /turtlesim
$ rosnode -h
```

- rostopic

```
$ rostopic list
$ rostopic info /turtle1/pose
$ rostopic echo /turtle1/cmd_vel
$ rostopic pub -r 10 /cmd_vel geometry_msgs/Twist
'{linear:  {x: 0.1, y: 0.0, z: 0.0},
 angular: {x: 0.0,y: 0.0,z: 0.0}}'
```

- rosmsg

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

Exercise

Make the robot drive around in circles using `rostopic`.

ROS services

- Request-reply type communication

```
$ rosservice list
$ rosservice type clear
```

- Calling services

```
$ rosservice call clear
$ rosservice type spawn | rossrv show
$ rosservice call spawn 2 2 0.2 ""
```

ROS parameters

- Storing and manipulating globally accessible parameters

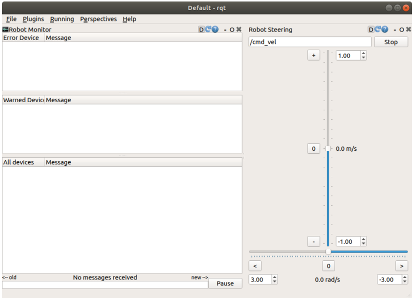
```
$ rosparam list
```

- Manipulating parameters

```
$ rosparam set background_r 150
$ rosservice call clear
```

RQT - ROS GUI

- Qt-based framework for GUI development for ROS
- Plugins: Tools for interacting with robots
- `$ rqt`



Exercise

Make the robot drive around in circles using `rqt`.

ROS filesystem

- ROS filesystem is organized into packages

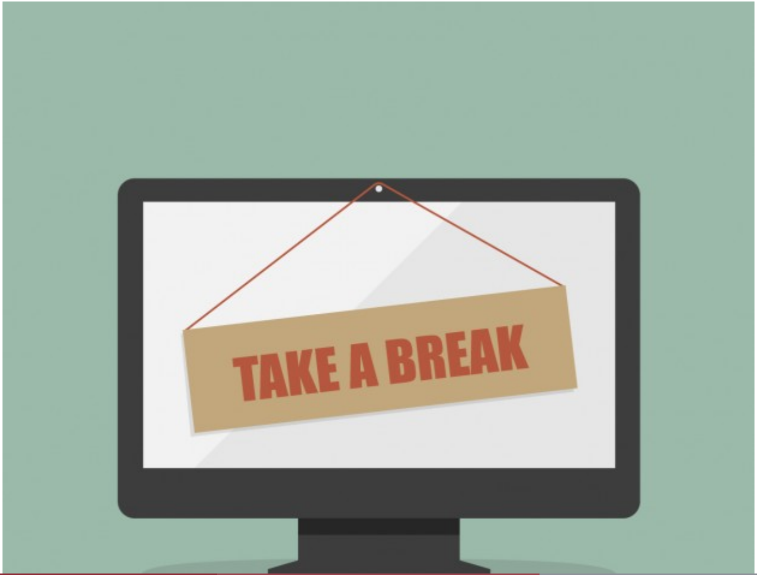
```
$roscd turtlesim
```

- `package.xml`
- Command line tools

```
$rospack find turtlesim
$rosls turtlesim/msg
$roscd log
```

Take a break

Before we continue



Installation (www.ros.org)

Creating your own workspace

- What is a workspace and why do I need one?
- Using catkin

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

- Lets build the workspace

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

- Setup/update environment (optional)

```
$ echo "source $HOME/catkin_ws/devel/setup.sh" >> \
~/.bashrc
$ source ~/.bashrc
$ echo $ROS_PACKAGE_PATH
/home/<username>/catkin_ws/src:/home/<username>...
```

Creating a new package

- Create a package

```
$roscd
$cd ../src
$catkin_create_pkg turtlecontrol rospy roscpp turtlesim
$roscd turtlecontrol
$gedit package.xml &
```

- Checking a package's dependencies

```
$rospack depends turtlecontrol
```

Writing your own publisher in Python (1/2)

```
$ mkdir scripts
$ cd scripts
$ gedit publisher.py &
```

```python
#!/usr/bin/env python
import rospy
from geometry_msgs.msg import Twist
def commander(v,w):
    while not rospy.is_shutdown():
        vel=Twist()
        vel.linear.x = v
        vel.angular.z = w
        pub.publish(vel)
        rospy.sleep(1.0)
```

Writing your own publisher in Python (2/2)

```python
if __name__ == '__main__':
    pub = rospy.Publisher('cmd_vel',Twist,queue_size=1)
    rospy.init_node('publisher')
    v=1; w = 2
    try:
        while not rospy.is_shutdown():
            commander(v,w)
    except rospy.ROSInterruptException:
```

```
$chmod +x publisher.py
$rosrun turtlecontrol publisher.py cmd_vel:=turtle1/cmd_vel
```

- Notice the argument remapping!
- The queue size is a balance between latency and completeness. If your usage only wants the latest data queue size $:= 1$.

Writing your own subscriber in Python

```
$ gedit subscriber.py &
```

```python
#!/usr/bin/env python
import rospy
from turtlesim.msg import Pose
def pose_callback(data):
    #rospy.loginfo(data)
    rospy.loginfo(rospy.get_caller_id()+" I saw turtlebot"+
    " at x = %d y =%d", data.x,data.y)
def subscriber():
    rospy.init_node('subscriber',anonymous=True)
    rospy.Subscriber("pose", Pose, pose_callback)
    rospy.spin()
if __name__ == '__main__':
```

```
$ rosrun turtlecontrol subscriber pose:=turtle1/pose
```

Writing multiple subscribers and publishers in Python (1/3)

We need classes when we want to have several subscribers and publishers.

```
$ gedit color_class.py &
```

```
#!/usr/bin/env python
#Must import rospy and msgs
import rospy
from geometry_msgs.msg import Twist
from turtlesim.msg import Color
# Node example class.
class NodeExample():
  # Must have callback for msgs
  def color_callback(self,data):
      rospy.loginfo(rospy.get_caller_id()+" Color I see"+
      " is r = %d g =%d b =%d", data.r,data.g,data.b)
```

Writing multiple subscribers and publishers in Python (2/3)

```
# Must have __init__(self) function for a class
# similar to a C++ class constructor.
  def __init__(self):
        # Create a publisher for turtle commands
        self.pub=rospy.Publisher('cmd_vel',Twist,queue_size=1)
        # Set the message to publish as command.
        # self.variable means you can access it from class fn
        self.cmd_vel = Twist()
        # Initialize message variables.
        self.cmd_vel.linear.x = 0.1
        self.cmd_vel.angular.z = 0.1
        # Create a subscriber for color msg
        rospy.Subscriber("color", Color, self.color_callback)
```

Writing multiple subscribers and publishers in Python (3/3)

```
      def run(self):
      # Main while loop.
                while not rospy.is_shutdown():
                # Publish our command.
                        self.pub.publish(self.cmd_vel)

if __name__ == '__main__':
# Initialize the node and name it.
        rospy.init_node('pyclass')
# Go to class functions that do all the heavy lifting.
        try:
                ne = NodeExample()
                ne.run()
```

```
$ rosrun turtlecontrol color_class cmd_vel:=/turtle1/cmd_vel
color:=/turtle1/color_sensor
```

Closing the loop

Exercise

Write a subscriber for the background color message and store the value in a class variable color. Write a controller that drives the turtle straight when the background is more blue than red, and runs in circles when the background is more red than blue.

Homework

Spawn two turtlebots in turtlesim. Write a turtle controller that drives the first turtle in arbitrary pattern ($v = 2$, $w = random(-5, 5)$ - use numpy.random.random_integers(low, high)). Use rqt or turtle_teleop_key to steer the second turtle and try to catch the first. Write a node that detects when both turtles are less then 1m apart and publishes a message "You win!".Use classes to pass information between turtles.

Some useful tools

- Plotting with `rqt_plot`
- Monitoring with `rqt_console` and `rqt_logger_level`
- Launching several nodes at once with `roslaunch`
- Logging with rosbag

What is ROS

- Not a computer operating system
- A robot "computational graph"
- A robot filesystem
- A build and packaging system
- A collection of debugging and visualization tools
- A repository of robotic algorithms
- A community

Useful links

- ROS community wiki
- Search the ROS wiki for: "concepts", "command line tools", ...
- ROS Q&A Forum
- ROS code repositories at github