Writing ROS Programs In Python

MRE/EME 5983 Robot Operating Systems

Overview

- Creating a ROS workspace
- Creating a ROS package
- ROS, Hello World
- Simple ROS Example
- Summary

Overview

- Creating a ROS workspace
- Creating a ROS package
- ROS, Hello World
- Simple ROS Example
- Summary

Creating a Workspace

- First step in writing ROS programs is creating a ROS workspace

- Here are the steps to create ROS workspace
  - Source the ROS installation environment

```
$ source /opt/ros/noetic/setup.bash
```

  - Create and build a catkin workspace

```
$ mkdir -p ~/catkin_ws/src
$ cd ~/catkin_ws/
$ catkin build
```

- We will be using catkin build (not catkin_make)

Notes:
These steps were completed on the base VirtualBox image that was provided to the class
There is another ROS workspace ~/install_ws that contains other packages that we will use

Overview

- Creating a ROS workspace
- Creating a ROS package
- ROS, Hello World
- Simple ROS Example
- Summary

Creating a ROS Package – 1 of 4

- Let's create a package for the work we are completing
  - http://wiki.ros.org/ROS/Tutorials/CreatingPackage

1. Change directories
   - cd ~/catkin_ws/src/

2. Run ROS utility to create the package
   - catkin_create_pkg hello_tutorial std_msgs **rospy** roscpp

3. Check the contents of the package

Creating a ROS Package – 2 of 4

- Package content

```
student@student-VirtualBox:~/catkin_ws/src$ tree hello_tutorial/
hello_tutorial/
├── CMakeLists.txt
├── include
│   └── hello_tutorial
├── package.xml
└── src
```

- CMakeLists.txt
  - CMake build system definition

- package.xml
  - Package manifest

Creating a ROS Package – 3 of 4

- As constructed, the CMakeLists.txt is very extensive. We can reduce the content to the following

```
cmake_minimum_required(VERSION 3.0.2)
project(hello_tutorial)
add_compile_options(-std=c++11)

# Find catkin macros and libraries
find_package(catkin REQUIRED COMPONENTS
  roscpp
  rospy
  std_msgs
)

# Define the catkin_package
catkin_package()

# Specify additional locations of header files
include_directories( ${catkin_INCLUDE_DIRS} )
```

Creating a ROS Package – 4 of 4

- We can also update the package.xml file

```xml
<?xml version="1.0"?>
<package format="2">
  <name>hello_tutorial</name>
  <version>0.0.0</version>
  <description>The hello_tutorial package</description>
  <maintainer email="student@todo.todo">student</maintainer>
  <license>TODO</license>

  <buildtool_depend>catkin</buildtool_depend>

  <build_depend>roscpp</build_depend>
  <build_depend>rospy</build_depend>
  <build_depend>std_msgs</build_depend>

  <build_export_depend>roscpp</build_export_depend>
  <build_export_depend>rospy</build_export_depend>
  <build_export_depend>std_msgs</build_export_depend>

  <exec_depend>roscpp</exec_depend>
  <exec_depend>rospy</exec_depend>
  <exec_depend>std_msgs</exec_depend>
</package>
```

Overview

- Creating a ROS workspace
- Creating a ROS package
- ROS, Hello World
- Simple ROS Example
- Summary

Writing Hello, ROS Program

- In the `~/catkin_ws/src/hello_tutorial/script` directory, create hello.py

```python
#!/usr/bin/env python3

# Import base ROS
import rospy

# Main function
if __name__ == '__main__':

    # Initialize the node and name it
    rospy.init_node('hello_ros_node_py')

    # Display a message
    rospy.loginfo('Hello, ROS!')
```

- **`catkin build`** required for the initial package build
- hello.py must have execution permissions
- Execute

```
student@student-VirtualBox:~$ rosrun course_tutorials hello.py
[INFO] [1650309965.263259]: Hello, ROS!
```

Overview

- Creating a ROS workspace
- Creating a ROS package
- ROS, Hello World
- Simple ROS Example
- Summary

Creating a Simple ROS Example

- For this example, we would like to create two nodes, a publisher and a subscriber
- The publisher node sends a message containing an integer counter to the subscriber node
- Below is the resource graph for this system

Simple ROS Publisher

- In our simple example, we first wish to create a publisher node that publishes a counter at a given publish rate
- To accomplish this, we can leverage the ROS std_msgs package
- This package contains the following message types

Bool
Byte
ByteMultiArray
Char
ColorRGBA
Duration
Empty
Float32
Float32MultiArray
Float64
Float64MultiArray
Header
Int16
Int16MultiArray
Int32
Int32MultiArray
Int64
Int64MultiArray
Int8
Int8MultiArray
MultiArrayDimension
MultiArrayLayout
String
Time
UInt16
UInt16MultiArray
UInt32
UInt32MultiArray
UInt64
UInt64MultiArray
UInt8
UInt8MultiArray

ROS std_msgs Examples

std_msgs/Int32 Message

File: std_msgs/Int32.msg

Raw Message Definition

```
int32 data
```

Compact Message Definition

```
int32 data
```

std_msgs/Float64 Message

File: std_msgs/Float64.msg

Raw Message Definition

```
float64 data
```

Compact Message Definition

```
float64 data
```

ROS Node Python Code Structure

- When creating our ROS nodes, we wish to leverage OOP concepts

- Our source code will contain the following
  - Node class definition
  - Constructor
  - Supporting methods and functions

Simple ROS Publisher

publisher.py

```python
#!/usr/bin/env python3

# Import base ROS
import rospy

# Import ROS message information
from std_msgs.msg import Int32

################################
# PublishNode class definition
################################
class PublishNode():
    def __init__(self):
        """Example publisher node"""

        # Variables
        self.counter = 0

        # Define publishers
        self.pub_int = rospy.Publisher('int_msg', Int32, queue_size=10)

        # Define subscribers

        # Set ROS rate
        self.rate = rospy.Rate(1)

        # Start ROS loop
        while not rospy.is_shutdown():

            # Call publisher
            self.publish_int_message()

            # Control time step
            self.rate.sleep()

        return
```

Publisher definition
- Topic = “int_msg”
- Queue size = 10

ROS publisher variable

ROS main loop

publisher.py

```python
    ################################################################
    # publish_int_message: Function to publish an integer message
    ################################################################
    def publish_int_message(self):

        # Define message
        msg = Int32()
        msg.data = self.counter

        # Publish messge up to coutner of 100
        if( self.counter < 100 ):
            self.pub_int.publish(msg)
            rospy.loginfo('Published int = %d' % msg.data)

        # Increment integer counter
        self.counter += 1


#################
# Main function
#################
if __name__ == '__main__':

    # Initialize the node and name it
    rospy.init_node('publisher_node')
    print("Publisher node initialized")

    # Start node
    try:
        PublishNode()
    except rospy.ROSInterruptException:
        pass
```

Define message

Publish message

Simple ROS Subscriber

subscriber.py

```python
#!/usr/bin/env python3

# Import base ROS
import rospy

# Import ROS message information
from std_msgs.msg import Int32


################################
# SubscribeNode class definition
################################
class SubscribeNode():
    def __init__(self):
        """Example subscriber node"""

        # Define publishers

        # Define subscribers
        self.sub_int = rospy.Subscriber('int_msg', Int32,
                                        self.int_message_callback,queue_size=10)

        # Enter ROS loop
        rospy.spin()

        return

    ##############################################################
    # int_message_callback: Function to process an integer message
    ##############################################################
    def int_message_callback(self, msg):

        rospy.loginfo('Received int = %d' % msg.data)
```

Subscriber definition
- Topic = “int_msg”
- Callback function defined
- Queue size = 10

ROS subscriber variable

ROS main loop

Display message

subscriber.py

```python
#################
# Main function
#################
if __name__ == '__main__':

    # Initialize the node and name it.
    rospy.init_node('subscriber_node')
    print("Subscriber node initialized")

    # Start node
    try:
        SubscribeNode()
    except rospy.ROSInterruptException:
        pass
```

Simple ROS Example

- Next steps
  - $ catkin build
  - $ source devel/setup.bash

  } Only necessary for package definition
  - Execute roscore, subscriber node and publisher node
  - Adjust publishing rate, queue sizes and subscriber processing times…

Results

```
student@student-VirtualBox: ~/catkin_ws
student@student-VirtualBox:~/catkin_ws$ rosrun course_tutorials subscriber.py
Subscriber node initialized
[INFO] [1650312038.960951]: Received int = 1
[INFO] [1650312039.961400]: Received int = 2
[INFO] [1650312040.961904]: Received int = 3
[INFO] [1650312041.961840]: Received int = 4
[INFO] [1650312042.962502]: Received int = 5
[INFO] [1650312043.962699]: Received int = 6
[INFO] [1650312044.961391]: Received int = 7
[INFO] [1650312045.963423]: Received int = 8
[INFO] [1650312046.962528]: Received int = 9
[INFO] [1650312047.963353]: Received int = 10
[INFO] [1650312048.963350]: Received int = 11
[INFO] [1650312049.963703]: Received int = 12
[INFO] [1650312050.963072]: Received int = 13
[INFO] [1650312051.961461]: Received int = 14
```

```
student@student-VirtualBox: ~/catkin_ws
student@student-VirtualBox:~/catkin_ws$ rosrun course_tutorials publisher.py
Publisher node initialized
[INFO] [1650312037.959400]: Published int = 0
[INFO] [1650312038.959949]: Published int = 1
[INFO] [1650312039.960781]: Published int = 2
[INFO] [1650312040.961351]: Published int = 3
[INFO] [1650312041.961008]: Published int = 4
[INFO] [1650312042.967237]: Published int = 5
[INFO] [1650312043.961388]: Published int = 6
[INFO] [1650312044.960848]: Published int = 7
[INFO] [1650312045.961852]: Published int = 8
[INFO] [1650312046.967558]: Published int = 9
[INFO] [1650312047.961803]: Published int = 10
[INFO] [1650312048.964583]: Published int = 11
[INFO] [1650312049.962476]: Published int = 12
[INFO] [1650312050.964652]: Published int = 13
[INFO] [1650312051.961073]: Published int = 14
```

rosbash

- We have used a few tools from the rosbash package already

- Here is a full list of tools
  - **roscd** - change directory starting with package, stack, or location name
  - **rospd** - pushd equivalent of roscd
  - **rosd** - lists directories in the directory-stack
  - **rosls** - list files of a ros package
  - **rosed** - edit a file in a package
  - **roscp** - copy a file from a package
  - **rosrun** - run executables of a ros package

Overview

- Creating a ROS workspace
- Creating a ROS package
- ROS, Hello World
- Simple ROS Example
- Summary

Summary

- We learned how to create a ROS package and execute simple ROS Python-based nodes