ROS Package Com

Thursday, August 1, 2024

6:16 PM

Day 1

------------------------------------------------ Starting with ROS ----------------------------------------------

* + Get familiar with the framework
  + Installing ROS Pre-requisites (Ubuntu Linux, WSL, etc.)
    - [Installation/Ubuntu - ROS Wiki](https://wiki.ros.org/Installation/Ubuntu)
    - [How to Use Linux Terminal in Windows 11 (allthings.how)](https://allthings.how/how-to-use-linux-terminal-in-windows-11/)
    - Also you can have a virtual machine with Ubuntu running (preferred)
  + Installing ROS1
    - [ROS/Tutorials/InstallingandConfiguringROSEnvironment - ROS Wiki](https://wiki.ros.org/ROS/Tutorials/InstallingandConfiguringROSEnvironment)

Day 2

-------------------------------------------------Workspace Setup----------------------------------------------------

* + Creating Workspace (If you haven't already)
    - Run the following commands

$ mkdir -p ~/catkin\_ws/src

$ cd ~/catkin\_ws/

$ catkin\_make

$ source devel/setup.bash

$ source ~/catkin\_ws/devel/setup.bash

* + IMPORTANT/optional so that you don't have to source your workspace every time you run a file from it, add the last command at the bottom of the bashrc file from ROS:

$ nano ~/.bashrc

A computer screen with white and blue text

Description automatically generated

-------------------------------------- ROS Package Creation ----------------------------------------------

* + Creating my ROS Package. Follow the following tutorial: [ROS/Tutorials/WritingPublisherSubscriber(python) - ROS Wiki](https://wiki.ros.org/ROS/Tutorials/WritingPublisherSubscriber%28python%29)
    - Run the following commands

# You should have created this in the Creating a Workspace Tutorial

$ cd ~/catkin\_ws/src

$ catkin\_create\_pkg talker\_pkg std\_msgs rospy roscpp

#std\_msgs is the messages dependency, this means we can send/receive string messages in this workspace

#rospy is the python dependency & roscpp is the C++ dependency, this means that both types of files/node can be created in this workspace

$ cd ~/catkin\_ws

$ catkin\_make

------------------------------- ROS Node Creation & Communication ----------------------------------

* + Creating Node 1: talker.py

#!/usr/bin/env python

import rospy

from std\_msgs.msg import String

def talker():

pub = rospy.Publisher('chatter', String, queue\_size=10) #"chatter" is the topic, any package subscribed to this topic will receive "hello world"

rospy.init\_node('talker', anonymous=True)

rate = rospy.Rate(1) # 1 Hz

while not rospy.is\_shutdown():

hello\_str = "hello world %s" % rospy.get\_time()

rospy.loginfo(hello\_str)

pub.publish(hello\_str)

rate.sleep()

if \_\_name\_\_ == '\_\_main\_\_':

try:

talker()

except rospy.ROSInterruptException:

pass

* + Creating Node 2: listener.py

#!/usr/bin/env python

import rospy

from std\_msgs.msg import String

def callback(data):

rospy.loginfo(rospy.get\_caller\_id() + "I heard %s", data.data)

def listener():

# In ROS, nodes are uniquely named. If two nodes with the same

# name are launched, the previous one is kicked off. The

# anonymous=True flag means that rospy will choose a unique

# name for our 'listener' node so that multiple listeners can

# run simultaneously.

rospy.init\_node('listener', anonymous=True)

rospy.Subscriber("chatter", String, callback) #subscribed to the topic "chatter", can receive information from any node publishing to that topic

# spin() simply keeps python from exiting until this node is stopped

rospy.spin()

if \_\_name\_\_ == '\_\_main\_\_':

listener()

* + Communicate
  + Build Nodes

$ cd ~/catkin\_ws

$ catkin\_make

* + In a terminal run: roscore
  + In another terminal run: rosrun talker\_pkg talker.py
  + In another terminal run: rosrun talker\_pkg listener.py

---------------------------------------- Distinct Package Communication ----------------------------------

The following is supplementary workflow to proof that nodes from different packages can communicate too:

* + Creating Python pkg 2
    - Creating Node 1
  + Communicating amongst python packages (Node 1 from pkg1 -> Node 1 from pkg 2)

Day 3:

--------------------------------- Distinct Workspaces + Languages Communication -----------------------

|  |  |  |
| --- | --- | --- |
| Workspaces | catkin\_ws | catkin\_cpp\_ws |
| Packages | talker\_pkg | listener\_cpp |
| Nodes | talker.py | listener.cpp |

Follow the previous steps to create catkin\_ws\_cpp & the listener\_cpp package insiden the new workspace.

For code and guidance on building the listener.cpp node go to: [roscpp\_tutorials/Tutorials/WritingPublisherSubscriber - ROS Wiki](https://wiki.ros.org/roscpp_tutorials/Tutorials/WritingPublisherSubscriber)

RUN COMMANDS:

In a terminal run: roscore

In another terminal: rosrun talker\_pkg talker.py

In another terminal:

$ source ~/catkin\_cpp\_ws/devel/setup.bash #sourcing your new workspace

$ ~/.bashrc #modify the file to include the line above and automatically source your workspace with ROS

$ rosrun listener\_cpp listener

Day 5

----------------------------------- Standard ROS JavaScript Library Integration -----------------------------

* + Standard ROS JavaScript Library, roslibjs: [roslibjs - ROS Wiki](https://wiki.ros.org/roslibjs), [RobotWebTools/roslibjs: The Standard ROS JavaScript Library (github.com)](https://github.com/RobotWebTools/roslibjs)

Steps to run a JavaScript Node:

1. $ npm install roslib (install npm if you don’t have it)
2. Install the rosbridge\_suit, this is a server to communicate JavaScript packages with ROS

sudo apt-get install ros-noetic-rosbridge-suite

1. In the terminal run: roscore
2. In a separate terminal run the rosbridge server: roslaunch rosbridge\_server rosbridge\_websocket.launch
3. In another terminal create in the same workspace you installed roslibjs a listener.js file: nano listener.js

The following is a modified code from: [roslibjs/Tutorials/BasicRosFunctionality - ROS Wiki](https://wiki.ros.org/roslibjs/Tutorials/BasicRosFunctionality)

// Import the roslibjs library

const ROSLIB = require('roslib');

// Create a ROS connection instance

const ros = new ROSLIB.Ros({

url: 'ws://localhost:9090' // URL of the ROS bridge server

});

// Handle connection events

ros.on('connection', function() {

console.log('Connected to websocket server.');

});

ros.on('error', function(error) {

console.log('Error connecting to websocket server: ', error);

});

ros.on('close', function() {

console.log('Connection to websocket server closed.');

});

// Create a ROS topic subscriber for the "chatter" topic

const chatterListener = new ROSLIB.Topic({

ros: ros,

name: '/chatter', // Subscriber to the "chatter" topic, this means that it will receive messages from any node publishing into that same topic.

messageType: 'std\_msgs/String'

});

// Define a callback function to handle incoming messages

chatterListener.subscribe(function(message) {

console.log('Received message on /chatter: ', message.data);

});

1. Run JavaScript Node

$ node listener.js

1. Now, in a different terminal, run the created python script for a publisher in the "chatter" topic -> talker.py

$ rosrun talker\_pkg talker.py

1. This will lead to you javascript terminal to hear and show the message "hello world" your python node is sending. You should have the following 4 terminals running in your computer:

A screenshot of a computer

Description automatically generated