Software Lecture 2:

Getting Started With ROS

Jacques Cloete

Time to Install ROS!

http://wiki.ros.org/noetic/Installation/Ubuntu

- Load Terminal and follow steps 1.1-1.6 from the linked tutorial to install ROS Noetic, noting the following:

1. Make sure to choose the Desktop-Full Install!

2. Pay extra attention to step 1.5; make sure to run the command listed for Bash

This modifies the bashrc file so that the script that sets up ROS Bash commands is automatically called every time you open a new terminal – this will save you from a lot of tedium in the future!

Time to Test the Installation

- Open up a new Terminal and run roscore
- You should see something very similar to the following:

```
jacques@JC-Workstation:~$ roscore
... logging to /home/jacques/.ros/log/dbbf55aa-ce4d-11ec-b088-314771286010/rosla
unch-JC-Workstation-8725.log
Checking log directory for disk usage. This may take a while.
Press Ctrl-C to interrupt
Done checking log file disk usage. Usage is <1GB.

started roslaunch server http://JC-Workstation:37707/
ros_comm version 1.15.14


SUMMARY
========

PARAMETERS
 * /rosdistro: noetic
 * /rosversion: 1.15.14

NODES

auto-starting new master
process[master]: started with pid [8735]
ROS_MASTER_URI=http://JC-Workstation:11311/

setting /run_id to dbbf55aa-ce4d-11ec-b088-314771286010
process[rosout-1]: started with pid [8745]
started core service [/rosout]
```

What is ROSCORE?

- roscore is a collection of core nodes and programs that form the foundations of a ROS-based system
- roscore MUST be running in order for your ROS nodes to communicate with each other
- Think of it as the underlying web that automatically connects all of your nodes together!

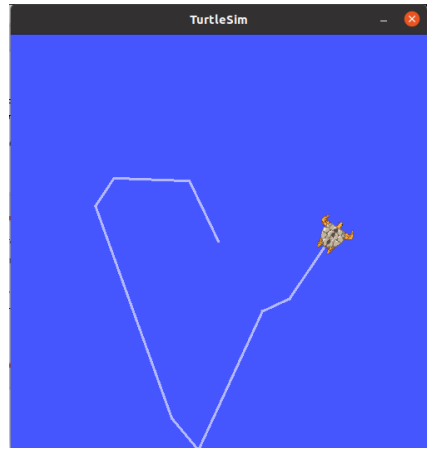
Running Your First ROS Program

- Open two more terminals and run the following:

Terminal 2: rosrun turtlesim turtlesim_node

Terminal 3: rosrun turtlesim turtle_teleop_key

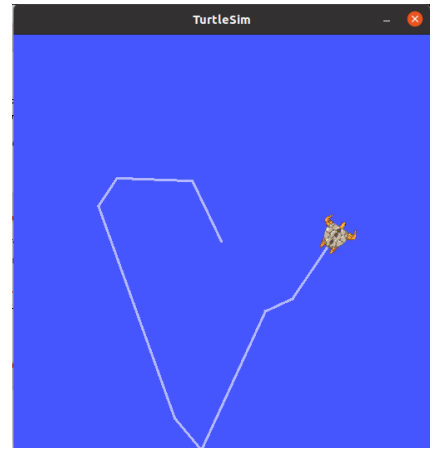
- You should see a turtle that can be controlled by your keyboard – try driving it around!



Publishers and Subscribers in Action

- The commands you ran set up two nodes

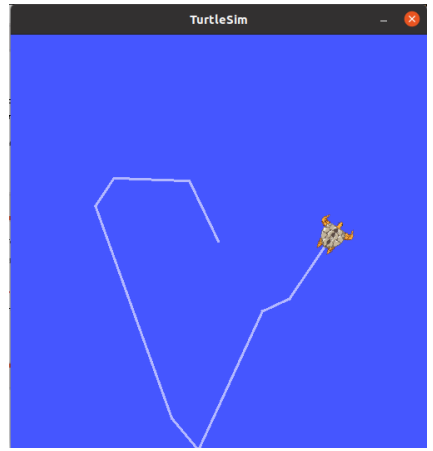
- turtlesim_node generates the turtlesim environment which acts as both a subscriber and a publisher – it listens to incoming velocity messages for the turtle, and sends out messages about the turtle's pose (i.e. position and orientation)



- turtle_teleop_key acts as a publisher – it sends out velocity messages for the turtle according to your inputs into the keyboard!

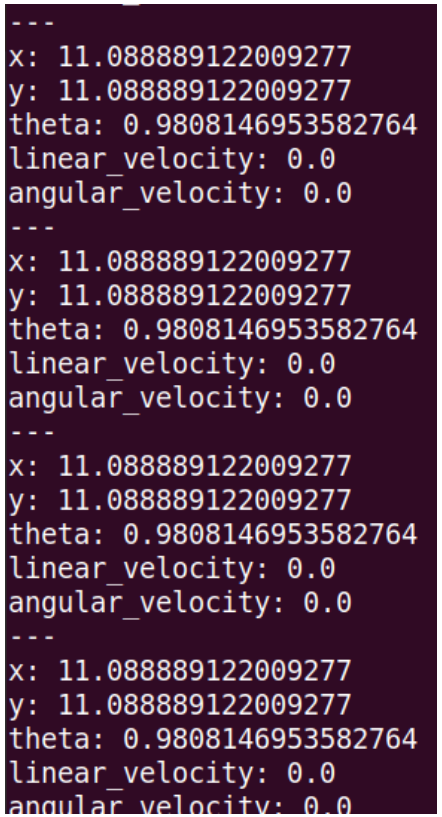
Publishers and Subscribers in Action

- Open a fourth terminal and run rostopic list
- This (very helpful) command shows you the names of all topics that are currently active
- In this case, the /turtle1/cmd_vel topic is used to exchange velocity command messages
- Meanwhile, the /turtle1/pose topic is used to exchange messages describing the turtle's pose



Topics and Messages in Action

- We can directly read the contents of messages!
- Now run rostopic echo /turtle1/pose and observe the output
- You will see a stream of messages describing the turtle's live pose, continually being published by the node running the turtlesim
- Go to the Terminal running turtle_teleop_key and make the turtle move – see how the messages published change as it does so!



Packages and the Workspace

- Programs and software in ROS are neatly organised and bundled together into directories called packages
- A (catkin) workspace is a folder where you can modify, build, and install these packages
- When you want to use some ROS software, you install the package for it into your workspace

Creating a Workspace

Exit all Terminals and load up a new one;

- Install catkin: sudo apt install ros-noetic-catkin python3-catkin-tools python3-osrf-pycommon
- Also install wstool: sudo apt install python3-wstool
- Now create a directory to host the workspace: mkdir -p ~/tutorial_ws/src
- Navigate to the source folder: cd ~/tutorial_ws/src

Creating a Workspace

- Run the following command to add missing dependencies:
  rosdep install -y --from-paths . --ignore-src --rosdistro noetic
- Navigate to the main workspace directory: cd ~/tutorial_ws
- Run the following: source /opt/ros/noetic/setup.bash
- Configure the properties for the workspace:
  catkin config --extend /opt/ros/noetic
- Build the workspace: catkin build

Sourcing the Workspace

- We must also source the workspace! To do this, run this line:
  source ~/tutorial_ws/devel/setup.bash
- Add the above command to your bashrc to automate it:
  echo 'source ~/tutorial_ws/devel/setup.bash' >> ~/.bashrc

The Workspace

- The workspace is made up of four main directories
- Source space contains the source code for all your installed and custom packages – this is where we will create new packages
- The other three directories are important for the functionality of the workspace but not covered here

An Aside – Building the Workspace

- Why did we have to build the workspace?
- Building is the process of compiling source code into stuff like executable programs
- In our case, each ROS package needs to be built before we can use the stuff they provide
- The catkin build command automatically builds every package in the workspace whilst handling the interactions and dependencies between packages when building!

Creating a Package

- To properly start making use of ROS, we will write our own programs and code – we will do this in custom packages
- Navigate into the Source folder: cd ~/tutorial_ws/src
- Create a new package:
  catkin_create_pkg tutorial_scripts std_msgs rospy
- The first argument is the package name, and all following arguments are the dependencies, i.e. the other packages required for use by the new package

Creating a Package

- Now return to the main workspace directory (cd ..) and once again build the workspace: catkin build
- Now we have a custom package ready to house our own programs and code!
- Note that every time you add a new package to the workspace, you will need to rebuild the workspace in order to incorporate that new package!

Summary

We covered:

- Installing ROS
- Roscore and a simple ROS Program
- Creating a workspace to hold our custom packages
- Creating a package to hold our programs and code

Next time, we will code up our first ROS program!