Lab Session 1

Release date: Monday, 1 February 2021
Due date and time: 10 am on Monday, 8 February 2021

This lab session will introduce you to the Robot Operating System (ROS). By the end of this session you will:

- Understand basic ROS concepts;
- Learn how to publish/listen ROS messages;
- Learn useful ROS command-line tools.

This session will introduce you only to the most basic concepts in ROS. You can explore and learn more using online resources, particularly the ROS wiki at: http://wiki.ros.org/ .

What is ROS?

The Robot Operating System (ROS) is a set of software libraries and tools that help you build robot applications.

It is named an "operating system" because its main goal is similar to a regular operating system on a PC: to provide abstraction. On a PC you should not need to re-write the low-level code to control a printer every time you write a new application that has print functionality. Instead, there is a driver for the printer that takes care of the low-level control, and when you write a new application you simply communicate with this driver on a higher abstraction level (e.g. a postscript file). Similarly, on a robot, you should not need to re-write the low-level code to control a wheel motor every time you write a new robot application. Instead, there is a separate process taking care of the low-level motor control, and your robot application communicates with this controller on an abstracted layer (e.g. the desired robot velocity). ROS' main functionality is to provide a mechanism for this abstracted communication between processes. Each such process is named a *node* in ROS. For example, the motor controller may be a node and your application may be another node. They speak through ROS *messages*. In this lab session, you will learn how to write two ROS nodes that communicate through a ROS message. There will not be much about robots today; instead we will learn about the basic functionality. Using ROS messages for controlling a robot will be the subject of the next lab session.

While ROS is similar to a regular operating system in terms of the abstraction it provides, the similarities do not go much further. For example, you do not install ROS instead of an OS. On the contrary, ROS needs to be installed on an OS, and ROS is best supported on Ubuntu Linux.

Since the official university Linux distribution is CentOS (not Ubuntu), in our module, you will be using a container (singularity) to run your ROS-based programs. Below we first describe how to do that. Additionally, since we expect most students to work from their homes this year, we describe how to make the remote connection to the university computers to run ROS programs. (Instead, if you are physically sitting in front of a computer in the Dec-10 lab, please see the section “Alternatively, if you are physically in Dec-10”.)

Running ROS programs remotely on university computers

The first thing you need to do is to make a VPN connection to the university. If you don't know how to do this, please follow the instructions on the IT page here and connect to the university VPN: https://it.leeds.ac.uk/it?id=kb_article&sysparm_article=KB0014410

Afterwards, open a browser window (e.g. a Firefox, Chrome, or Internet Explorer window) and go to the address: feng-linux.leeds.ac.uk/gpu/

(Don't forget the "gpu" at the end of this address! It is important!)

Login with your university credentials.

You should now see the desktop of a CentOS Linux computer on university servers.

Open a Terminal and execute the following command:

singularity shell --nv /vol/scratch/ros/VGL.img

This will put you in a container running Ubuntu Linux.

Then you should create a new xterm terminal window by running the following command:

xterm &

You can create as many xterm terminal windows as you want by repetitively executing the command above. Try it and create multiple xterm terminal windows.

Important: These xterm windows are terminal windows running inside the singularity container running Ubuntu Linux, which is what we need. Therefore, all the following commands of this lab session (and future lab sessions) should be executed in these xterm windows. Whenever we refer to a "terminal window" in the worksheets, we will be talking about these windows you have created using "xterm &".

You can now start running ROS programs.

After you are done working on the lab worksheet and using the remote connection, please remember to disconnect from university VPN. Otherwise, your internet connections and data may continue to go through university servers even if they are not related to your module work.

Alternatively, if you are physically in Dec-10:

The instructions above assume you are making a remote connection to the university computers. Alternatively, if you are physically in the School of Computing DEC-10 computer labs, please use the commands described below.

After you login to a DEC-10 CentOS computer, open a terminal and execute the following command:

singularity shell --nv /vol/scratch/ros/comp3631.img

(Most computers in Dec-10 use nvidia graphics, but a few of them don't. For those computers that do **not** use nvidia, you should exclude the "--nv" from the above command. To understand if the computer you use has nvidia, execute the command "lspci | grep -i nvidia". If you see no output, then the computer does **not** use nvidia, and you should exclude "--nv" from the above command.)

This will put you in a container running Ubuntu Linux.

Then you can create a new terminal window by running the following command:

xterm &

You can create as many xterm windows as you want by repetitively executing the command above. Try it and create multiple xterm windows.

All the following commands of this lab session (and future lab sessions) should be executed in these xterm windows.

If you are following these lab worksheets physically in Dec-10 computers, you may need to run slightly different versions of some of the commands in the lab worksheets. Please watch out for blue boxes like this one, where we will give the alternative version of a command you should run.

Setting up a ROS Catkin workspace

Before you write any ROS code, you need to set up a ROS workspace.

Your code must reside in a workspace to be discovered by ROS. ROS workspace management is performed by a piece of software called "catkin". That is why you will see that these workspaces are sometimes referred to as "catkin workspace".

1. Under your home directory, create a folder to house your catkin workspace (Name it catkin_ws)

mkdir catkin_ws

2. Clone your git repository under your catkin_ws.

cd catkin_ws

git clone https://gitlab.com/comp3631/<YOUR-UNI-USERNAME>.git src

Don't forget the 'src' at the end of the command. It will clone the contents of your git repository under a directory named 'src'.

Go into the src directory, and look at the directories cloned from the git repo.

cd src

ls

You will see one directory for each lab session and the project you will work on. **You must put the files you create in a lab session into the corresponding directory.** For example in this lab session, you must put all your code under catkin_ws/src/lab1 . You must also add, commit and push all your files back to git, so that you can access them later on the real robots.

3. Now initialise the catkin workspace from within the src directory

catkin_init_workspace

4. You can now build (compile) the code in your workspace. Navigate back to the top level of your catkin workspace (e.g. cd $HOME/catkin_ws) then execute:

catkin_make

source devel/setup.bash

It would also be worth your time to add this last command into your .bashrc to ensure that you do not need to source the workspace every time a new terminal is opened. (You cannot use the relative path devel/setup.bash in your .bashrc file. Use the absolute path $HOME/catkin_ws/devel/setup.bash)

You will use one workspace for all the code you will develop in this module. In other words, you do **not** need to create a workspace (steps 1 to 3 above) for every different worksheet you will be working on. You should simply use the lab1, lab2, lab3 … directories under the catkin_ws/src that you created here. You may need to compile your code (step 4 above) every time you change the code in your workspace.

You can find more information about ROS Catkin workspaces here:
http://wiki.ros.org/catkin/Tutorials/create_a_workspace

Catkin Packages

A package houses related code and files in a named directory. Each of the lab1, lab2, … directories under your catkin_ws/src are packages.

Navigate into the src folder within the lab1 package, this is where you will edit your first python scripts.

cd $HOME/catkin_ws/src/lab1/src

You can find more information about creating ROS packages here:
http://wiki.ros.org/catkin/Tutorials/CreatingPackage

A simple publisher and subscriber

You can use Python or C++ to write ROS nodes. Here, we will use Python. There will be two nodes, one *talker.py* and one *listener.py*. The talker will publish a simple message and the listener will subscribe to that message and print what it hears to the terminal.

In case they are not, make these scripts executable:

chmod +x talker.py listener.py

(Whenever you create or download a python script, you will need to make them executable by running chmod +x on them, as above. Please keep this in mind for the rest of the semester.)

Before you run these scripts, inspect them. Go to this link

http://wiki.ros.org/ROS/Tutorials/WritingPublisherSubscriber(python)

and read the section "1.2 The code explained" which explains talker.py. Scroll down, there is another section named "2.2 The code explained" which explains listener.py. Read both sections and make sure you understand what is happening inside both the talker and the listener.

You can use ROS commands to run your nodes. However, for the nodes to be able to communicate, a ROS master must be available. So, in a new terminal window, we first run a ROS master using the roscore command:

roscore

Now use rosrun to run the publisher node in a separate terminal window:

rosrun lab1 talker.py

(If you got the error: "[rospack] Error: package 'lab1' not found", then you probably did not put "source devel/setup.bash" into your .bashrc yet, and you have not executed it in your new terminal window either. You have to do at least one of these for ROS to be able to find your packages.)

You should notice the publisher logging its outputs in the terminal you ran it in.

Open another terminal window and rosrun the subscriber node:

rosrun lab1 listener.py

Launch files

A robot runs tens, sometimes hundreds of nodes like this. As you can imagine getting the robot operational by running each node separately would be far too tedious to handle. Therefore ROS employs launch files to run many nodes in one go.

Now Crtl-c your roscore, publisher and listener, and we will re-launch them this time using a single launch file.

The .launch file specifies nodes from packages and any parameters they may need to be launched when the .launch file is ran. The .launch files follow a specific syntax, a form of XML. A launch file always begins with a <launch> tag and ends with a </launch> tag. Declarations of nodes to be launched are enclosed within a <node> tag for example:

<node pkg="lab1" type="talker.py" name="talker"/>

pkg refers to the package we are launching a node from. type refers to the exact node file we wish to launch. The name parameter allows us to name the node; you can change the name if you wish. Under lab1/launch, you should find the launch file that launches the two nodes: lab1.launch. Inspect lab1.launch file and see how it refers to the talker and listener scripts.

Then in a terminal window roslaunch the launch file:

roslaunch lab1 lab1.launch

OR use the absolute/relative path of the launch file:

roslaunch path/to/lab1.launch

(Note that we did not run roscore this time. The command 'roslaunch' handles that for us.)

You are now running the talker and listener.

Useful ROS commands

ROS provides many command line tools to inspect the state of the ROS environment. For example, while your launch file is still running, in a new terminal window run:

rosnode list

This will list the names of all the nodes running currently in the ROS environment.

You can also get information about a specific node. For example, run:

rosnode info talker

Notice that, ROS gives you information about the type of this node, but also about the topics it is publishing to.

Try running rosnode info for another running node and see what information you get.

Similar to rosnode, there is also rostopic to get information about topics. Run:

rostopic list

This will list all the active topics in your ROS environment.

Again, you can get information about a specific topic. For example, run:

rostopic info /chatter

Notice that ROS gives you information about the type of the topic and the nodes publishing to and listening to this topic.

Try running rostopic info on other topics you have seen in the rostopic list output.

Using rostopic echo you can also listen to a particular topic. Execute:

rostopic echo /chatter

One final useful command is roscd. This is like regular cd in Linux, but allows you to navigate between ros packages without knowing the full path. For example, go to any random directory on your system, and then execute:

roscd lab1

Notice that this takes you to your package's directory.

Exercises

1. Create a copy of talker.py under lab1/src and save it as a new file numeric_talker.py. Edit this file to include an additional publisher publishing to a new /numeric_chatter topic with message type Int8 (an 8-bit integer type). Your talker should keep a counter starting from 0, increment it every iteration of the while loop (resetting back to zero at 127, which is the max value for a signed 8-bit integer), and publish the value of the counter to the /numeric_chatter topic. You will need to import the new message type to be able to use it. (Go to http://wiki.ros.org/std_msgs to see all the standard message types defined in ROS.) Do not remove the String publisher to the /chatter topic! A node can publish to multiple topics.

2. Create a copy of listener.py under lab1/src and save it as numeric_listener.py. Edit this file to include an additional subscriber to the /numeric_chatter topic and print the received number in its callback function. Do not remove the subscriber to the /chatter topic! A node can listen to multiple topics.

Submission instructions: When you think you have completed this worksheet, add, commit, and push your new files (numeric_talker.py, numeric_listener.py, and any other related files you might have created) and changes to the git repository with the correct git message “Lab1 completed.”. **You must do this before the deadline**. Your commit with this specific message is what informs us that you have submitted your files and we can then check your files. Specifically, you can use the commands below to commit and push your code to git with the correct message:

cd $HOME/catkin_ws/src/lab1/src/

git add <list-of-files-you-want-to-push-to-git>

git commit -m “Lab1 completed.”

git push