

# Introduction to TurtleBot

EECS 376/476 Lab1

due: 2/23/18

## INTRODUCTION

This initial lab is based on an EECS 275 lab designed by Prof. Greg Lee; much of the text here is re-used with permission.

The robot hardware used for the first part of this course is a version of the Turtlebot 2 that is produced by Autonomous. The sensors on this platform include a microphone, bumper sensors, and an ASUS Xtion Pro 3D depth camera (similar functionality to the Microsoft Kinect). This particular TurtleBot uses an NVIDIA Jetson TK1 quad-core ARM board with 192 GPU CUDA core.

TurtleBot makes extensive use of ROS. This laboratory assignment provides an introduction to using the platform and demonstrates the methods to control the robot from a secondary computer over the network. *Please note:* This type of operation is subject to difficulty. Debugging and understanding the difficulties is an express purpose of the exercises below.

## EXERCISES

There are multiple robots available for use by the course. Groups should attempt to use the same platform for this and subsequent laboratories. The robots have similar names, e.g.: deeplearning02w, deeplearning03w and deeplearning04w. These are the names associated with the wifi interfaces, removing the “w” should be the wired interfaces were the robot to have a wired connection (sometimes the “w” hostname works for wired connections, so be flexible and try other names if connecting is difficult). Labels identifying the robots are located on the top of the robots.

## Turn on the Robot

Set the master power switch for the robot to the “on” position. The NVIDIA Jetson TK1 board must be powered up manually after the master power switch.

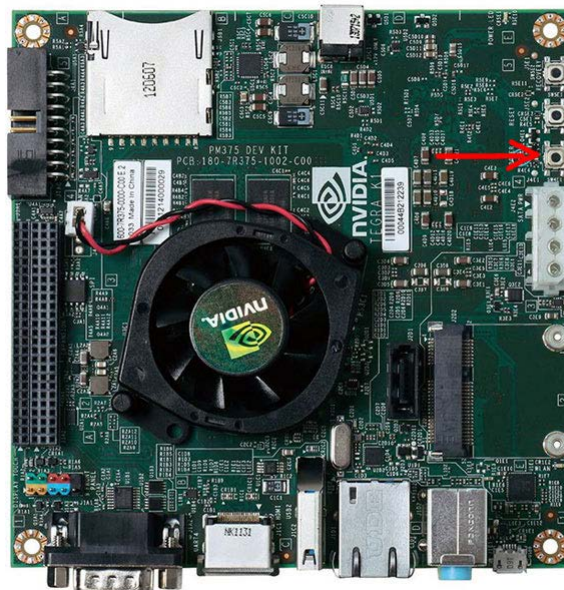


Figure 1: The power button for the NVIDIA Jetson TK1 board must be pressed after the main power has been switched on. A green LED near the switch will illuminate when the board is turned on.

## Finding the Robot on the Network

The robots are connected to the CWRU network through a wireless access point in the Glennan Laboratory that is dedicated to robot operation. (You should not connect your laptop to that wireless access point.) This access point connects the robots to the wired campus network, not CaseGuest or CaseWireless. The connection between the campus wired and wireless networks has security features that are incompatible with ROS. It is, therefore, required that the desktops in the laboratory be used when connecting to the robots as laptops connected to the campus wireless networks will work poorly. (Note, you may establish connections, but the robots may operate intermittently.)

Because the robots are still using the CWRU campus network resources, it should be possible to connect to them using URIs instead of IP addresses. This is beneficial as the robots have not been assigned static IP addresses and it would be difficult to determine their IP addresses when they are turned on. The drawback, however, is that the campus DNS system has some delay. If the campus DNS does not update quick enough, it is possible to use Google's DNS to find the IP addresses, but if it is necessary to use Google's DNS, it will be necessary to use IP addresses instead of URIs.

Run the following command to determine if the campus DNS has propagated the name of the robot through the network.

```
host deeplearning0{2|3|4}w
```

The command will indicate the host is not found if the DNS has not propagated the address through the network. If the command does not return the IP address of the robot after a few minutes, it is possible to use Google's DNS. The IP address for Google's DNS is 8.8.8.8. To use Google's DNS, use the following command.

```
host deeplearning0{2|3|4}w 8.8.8.8
```

After the address has been propagated, the command will return the IP address of the robot. Verify that the robot is reachable by pinging it. If it was necessary to use Google's DNS before, it will be again.

```
ping deeplearning0{2|3|4}w [8.8.8.8]
```

This command will return statistics about the route between control computer and robot, if the robot is reachable.

Note: logging directly into the robot (e.g., via `ssh`) is heavily discouraged. No part of this laboratory should require any actions be performed locally on the robots.

Alternately you could attempt to ping the robots by using the actual IP address returned by the host command above, where the X's are replaced with relevant IP address information.

```
ping XXX.XXX.XXX.XXX
```

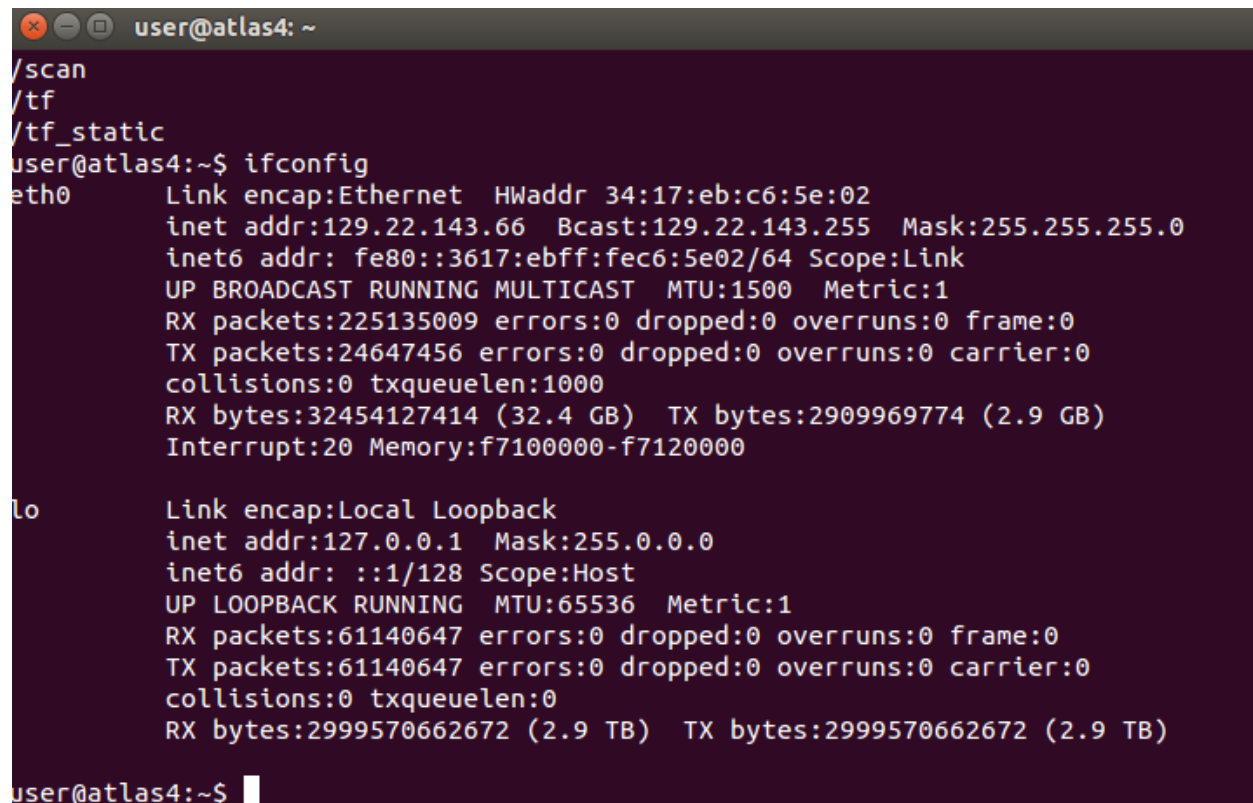
## Connecting to the Robot

The next few steps are done manually to understand the process, but a script will be used in the future to automate the process.

ROS is designed to exist on the network. It defaults to local operation by default. To leverage the network distributed capabilities of ROS, the computer with the ROS Master (robot) must be configured to use the proper network interface, not the "localhost" interface. This is part of the bootup scripts on the robots and is already complete. Any other computers that will connect to the ROS Master must be configured to look to the robot for the ROS Master instead of locally. This configuration is applied after the ROS setup.bash script has been run, but before running anything. (Note: If you have taken to manually running `roscore` instead of allowing `roslaunch` to manage this automatically, you must not

run `roscore` manually, the ROS Master is located on the robot.) If the campus DNS never managed to propagate the hostname, you will need to use the IP address instead of the hostname below.

First issue the `ifconfig` command note your machines `eth0` or `eth1` inet addr as shown in Figure 2:

A terminal window titled 'user@atlas4: ~' showing the output of the 'ifconfig' command. The output is color-coded: green for interface names, red for status, and yellow for statistics. The 'eth0' interface is shown with its Ethernet link, IP address 129.22.143.66, and various statistics. The 'lo' interface is shown as a local loopback with IP address 127.0.0.1.

```
user@atlas4: ~  
/scan  
/tf  
/tf_static  
user@atlas4:~$ ifconfig  
eth0      Link encap:Ethernet  HWaddr 34:17:eb:c6:5e:02  
          inet addr:129.22.143.66  Bcast:129.22.143.255  Mask:255.255.255.0  
          inet6 addr: fe80::3617:ebff:fec6:5e02/64 Scope:Link  
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1  
          RX packets:225135009 errors:0 dropped:0 overruns:0 frame:0  
          TX packets:24647456 errors:0 dropped:0 overruns:0 carrier:0  
          collisions:0 txqueuelen:1000  
          RX bytes:32454127414 (32.4 GB)  TX bytes:2909969774 (2.9 GB)  
          Interrupt:20 Memory:f7100000-f7120000  
  
lo        Link encap:Local Loopback  
          inet addr:127.0.0.1  Mask:255.0.0.0  
          inet6 addr: ::1/128 Scope:Host  
          UP LOOPBACK RUNNING  MTU:65536  Metric:1  
          RX packets:61140647 errors:0 dropped:0 overruns:0 frame:0  
          TX packets:61140647 errors:0 dropped:0 overruns:0 carrier:0  
          collisions:0 txqueuelen:0  
          RX bytes:2999570662672 (2.9 TB)  TX bytes:2999570662672 (2.9 TB)  
  
user@atlas4:~$
```

Figure 2: Detecting the ETH address for ROS\_IP configuration

```
# Setup ROS  
source /opt/ros/indigo/setup.bash  
# Where is the ROS Master  
export ROS_MASTER_URI=http://deeplearning0{2|3|4}w.eecs.cwru.edu:11311  
# export ROS_MASTER_URI=http://<IP Address of Robot if DNS is not up>:11311  
# identify your local ROS_IP  
export ROS_IP=<ETH0 INET ADDR (See Figure 2)>
```

To verify that you are connected to the robots, list the available ROS topics.

```
rostopic list
```

The list of topics will be fairly long and include `/cmd_vel` if everything is working. If there is no entry `“/cmd_vel”` it is possible that the list of ROS topics is from the local machine and not the robot. (There should be not ROS Master [`roscore`] running locally.)

## Drive the Robot: teleop

ROS contains many useful command line and GUI tools to inspect a ROS system. Executables such as `roscall`, `rostopic` and `rosparam` allow users to retrieve information about ROS nodes currently connected to the system, view the messages that are being passed and between which nodes, and inspect the parameter server. The executable `rqt_gui` is a GUI interface to the ROS system that has a number of plugins that extend from low level listing and reading of messages, and writing specific messages from and to the ROS system, to higher level tools that show the hierarchy of nodes and messages, allow robots

to be driven and to view the data from sensors. To start the `rqt_gui` executable, use the following command:

```
roslaunch rqt_gui rqt_gui &
```

This interface contains a tool that can drive the robot around. To use it, add the Steer Robot tool from the Plugins->Robot Tools menu. The default address for driving robots is `/cmd_vel`.

An alternative teleoperation interface is “teleop\_twist\_keyboard” (see: [http://wiki.ros.org/teleop\\_twist\\_keyboard](http://wiki.ros.org/teleop_twist_keyboard)). You can run this with:

```
roslaunch teleop_twist_keyboard teleop_twist_keyboard.py
```

and `cmd_vel` commands will be published based on keyboard keys.

## Drive the Robot: program control

Modify your open-loop commander program from PS1. Change the output topic from `/robot0/cmd_vel` to `/cmd_vel`.

Modify the program to command the robot to move in a 1mx1m square. Observe and report on how accurately the robot travels in a square. Make a movie of your robot motion, and post it on YouTube.

Be sure to reduce your speed from the default for the `stdr` program! **Do the math on what you need to move in a 1 x 1 m square but as an FYI it should be less than 0.4 m / s! 1 m/s command is fast in real life!**

## Report:

Submit one report per group, together with:

- a pointer to your code on github
- a link to your YouTube video
- A concise report on your results with numerical observations on accuracy of the robot attempting to drive around a square.