ECEN430 ROS Tutorial Pt1

Nick Thompson

July 2018

Chapter 1

ROS and Ubuntu Introduction

1.1 Command Line Refresher

ROS uses the Linux command line for the majority of programming and execution tasks. Before starting with ROS ensure that you are up to speed on how to navigate the Linux command line efficiently. Table 1.1 lists common commands that you will with the command line. I wouldn't throw this out as it will be useful later.

Table 1.1: Common Linux Command Line tools

Syntax	**Useage**
pwd	Print Working directory (where you currently are)
ls	List files in directory
cd foldername	Change directory
mkdir foldername	Make directory
gedit filename	Edit a file with a GUI, if the file doesn't exist it will be created
nano filename	Edit a file with command line, if the file doesn't exist it will be created
mv /path/oldfilename /path/newfilename	Moves a file
cp /path/oldfilename /path/newfilename	Copies a File
rm filename	Removes a file
rm -r foldername	Removes a folder
sudo	Gets the admin privileges
apt-get install pkgname	Installs a new package, usually has to be used with sudo privileges
cat filename	Prints the file to command line

Table 1.2 shows some of the more exotic commands that will help when debugging ROS issues.

Table 1.2: My caption

Syntax	**Useage**
lsusb	List connected USB devices
dmesg — grep tty	List USB devices with more detail
ifconfig	Display IP address of network interfaces

1.2 ROS Introduction

ROS or robot operating system is middle-ware that sits between high-level control and the sensor/actuators. ROS was initially developed at Stanford University, but has become a worldwide standard for robotic development. The ROS system consists of nodes, each node has a compute function to gather, manipulate or display data. Data is transferred between nodes on topics. A node can subscribe or publish data to/from any topic. Each topic has a message type, a message defines how the data is packetised. At a low level ROS uses the TCP/IP stack for transferring data, this allows easy expansion over networked systems.

1.2.1 Catkin Work-space

ROS uses the C-Make build system to compile all c++ code. To ensure that the compiler can scale to cover all the packages the code is stored in a work-space. The work-space is usually located in the home dicetory `~/catkin_ws/`. The devel and build folders contain the executable and shell scripts needed to execute the code. The catkin_ws has file structure as shown below.

```
catkin_ws/
    src/
        package1/
        package2/
        package3/
    /build
    /devel
    /install
```

All ROS packages can be built in the work-space with the `catkin_make` command. Single packages can be made with `catkin_make *package_name*`

1.2.2 ROS Master

No ROS nodes can be launched without the ROS master. The ROS Master provides naming and registration services to the rest of the nodes in the ROS system. It tracks publishers and subscribers to topics as well as services. The role of the Master is to enable individual ROS nodes to locate one another. Once these nodes have located each other they communicate with each other peer-to-peer. An example registration process is shown below in Fig 1.1.

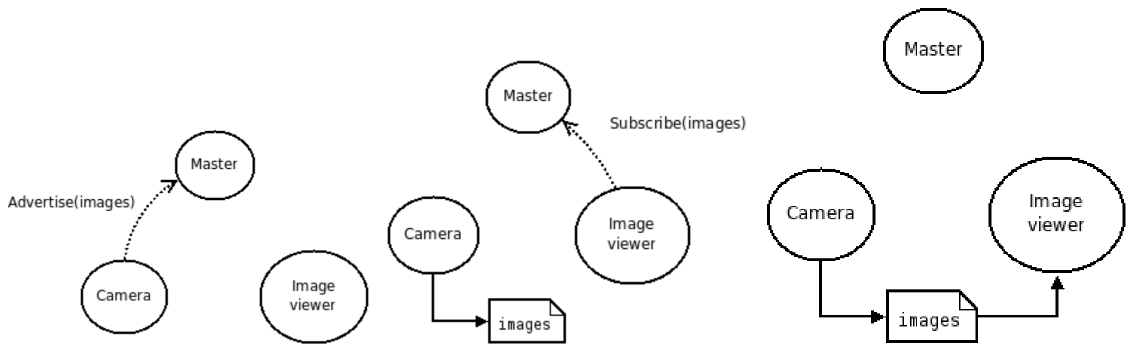


Figure 1.1: The Process of registering a Publisher and Subscriber with the ROS Master

The ROS master is usualy started with the the `roscore` command. When using launch files the ROS master is started with the `roslaunch` command.
Images and text from: `http://wiki.ros.org/Master`

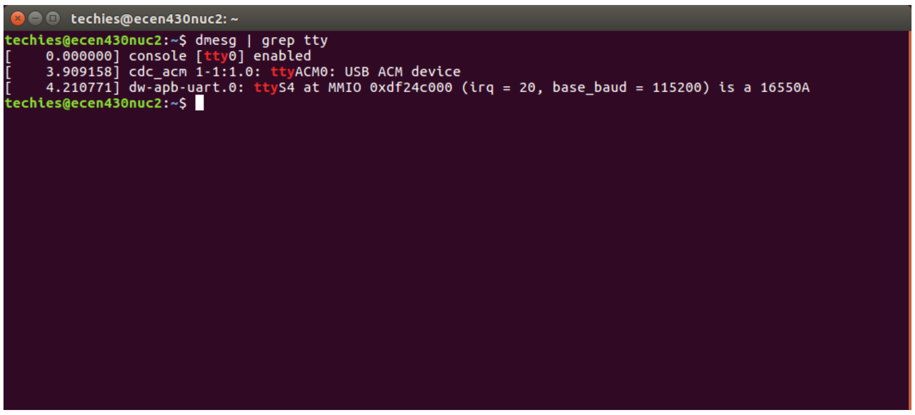
Chapter 2

LIDAR

LiDAR or Light Detection and Ranging is a technique used for determining the range to objects. The LiDARs you have today are the URG-04LX-UG01 and cost around $1500 NZD. Now for the joy of ROS, there is already a library available for these LiDARs. You can download and install it using the command:

```
$ sudo apt-get install ros-kinentic-urg-node
```

Now plug in your LiDAR to the USB port on your NUC, the spindle should spin up and the LED start flashing. Use the `$ dmesg | grep tty` command to identify what serial port the LiDAR has been assigned. By default it should be on `/dev/ttyACM0`, Figure 2.1 shows the output when the LiDAR is correctly connected.



```
techies@ecen430nuc2: ~
techies@ecen430nuc2:~$ dmesg | grep tty
[    0.000000] console [tty0] enabled
[    3.909158] cdc_acm 1-1:1.0: ttyACM0: USB ACM device
[    4.210771] dw-apb-uart.0: ttyS4 at MMIO 0xdf24c000 (irq = 20, base_baud = 115200) is a 16550A
techies@ecen430nuc2:~$
```

Figure 2.1: The dmesg output with the LiDAR shown on port /dev/ttyACM0

Using terminal start the ROS master:

```
$ roscore
```

Open a new terminal tab using `CTRL + SHIFT + T`. Using this terminal tab start the urg_node with:

```
$ rosrun urg_node urg_node _serial_port:=/dev/ttyACM0
```

Where the serial port is set to the serial port of the LiDAR. We can now varify that the LiDAR is sending data using the rostopic command line tool. Fist check that the the urg_node is publishing data on the `/scan`. In a new terminal tab, use the command:

```
$ rostopic list
```

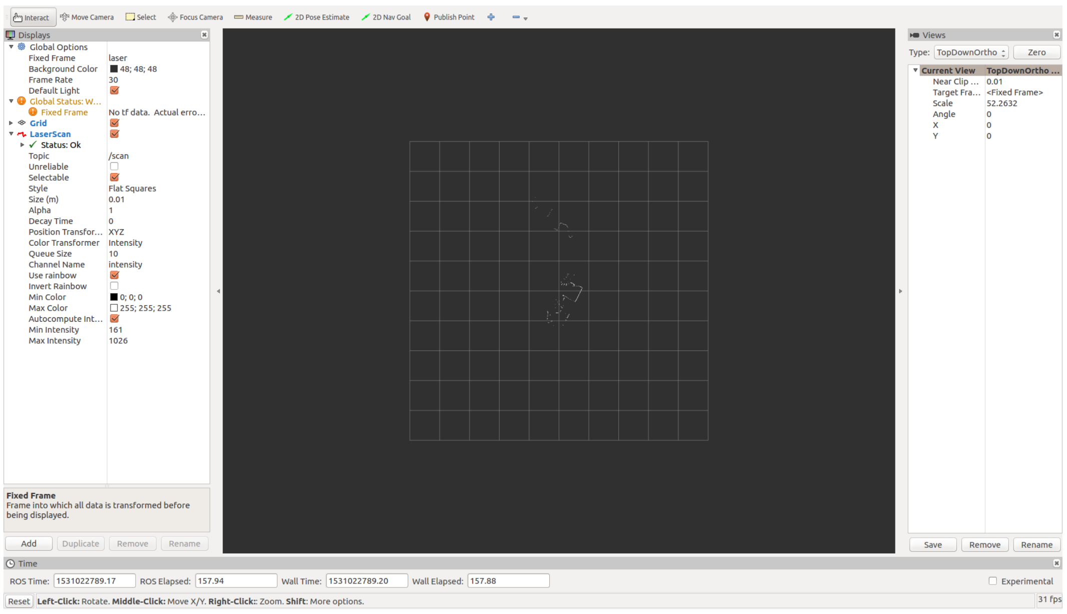
If the `/scan` topic shows up in the list, use rosoptic to check the publish frequency of the topic with:

```
$ rostopic hz /scan
```

If the LiDAR is running correctly it should be publishing data at 10 Hz. We can now visualize the data using rviz, launch it with:

```
$ rviz
```

Using the panel on the left of RVIZ add the LiDAR data as shown in Figure 2.2.



Global Options	
Fixed Frame	laser
Background Color	■ 48; 48; 48
Frame Rate	30
Default Light	✓
Global Status: W…	
Fixed Frame	No tf data. Actual erro…
Grid	✓
LaserScan	✓
Status: Ok	
Topic	/scan
Unreliable	☐
Selectable	✓
Style	Flat Squares
Size (m)	0.01
Alpha	1
Decay Time	0
Position Transfor…	XYZ
Color Transformer	Intensity
Queue Size	10
Channel Name	intensity
Use rainbow	✓
Invert Rainbow	☐
Min Color	■ 0; 0; 0
Max Color	□ 255; 255; 255
Autocompute Int…	✓
Min Intensity	161
Max Intensity	1026

Fixed Frame
Frame into which all data is transformed before being displayed.

Type: TopDownOrtho

Current View	TopDownOrtho …
Near Clip …	0.01
Target Fra…	<Fixed Frame>
Scale	52.2632
Angle	0
X	0
Y	0

ROS Time: 1531022789.17 ROS Elapsed: 157.94 Wall Time: 1531022789.20 Wall Elapsed: 157.88 ☐ Experimental

Left-Click: Rotate. **Middle-Click:** Move X/Y. **Right-Click:**: Zoom. **Shift**: More options. 31 fps

Figure 2.2: Caption

once you are done, stop all running programs

Chapter 3

Mapping

ROS includes a large number of mapping packages, one of the essayist to use is Hector Mapping as it only uses input from a LiDAR. To use Hector Mapping we have to tell ROS the spacial relation between the laser and the robot base, this is known as a TF or transform. As the laser will not be moving relative to the base of the robot we can use a static_tf_publisher.
Rather than publishing the TF from a command line tool we will use a launch file. A launch file is an easy way of launching a number of different nodes from one command line tab. To create a launch file we need to first create a ROS package. Using command line navigate to the src directory of the catkin_ws using the command:

```
$ cd ~/catkin_ws/src
```

Once here we can create the ROS package with the command:

```
$ roscreate-pkg myFirstPackage std_msgs roscpp rospy
```

cd into the newly created myFirstPackage directory and create a new directory called launch, then cd into that:

```
$ cd myFirstPackage
$ mkdir launch
$ cd launch
```

Your path should now be: `~/catkin_ws/src/myFirstPackage/launch/` . Now create a new file called hector_mapping.launch using your favourite editor. I like gedit, but you can use any one you please:

```
$ gedit hector_mapping.launch
```

Copy and paste the code below into the launch file.

```
<launch>
  <!--Sensor setup-->
  <node pkg="urg_node" name="urg_node" type="urg_node" output="screen">
    <param name="serial_port"  value="/dev/ttyACM0"/>
  </node>

  <!--Transform setup-->
  <node pkg="tf" type="static_transform_publisher" name="base_frame_2_laser_link" args="
    0 0 0 0 0 0 /base_link /laser 100"/>
  <node pkg="tf" type="static_transform_publisher" name="
    base_link_base_frame_broadcaster" args="0 0 0 0 0 0 /base_frame /base_link 100" />
  <node pkg="tf" type="static_transform_publisher" name="odom_base_frame_broadcaster"
    args="0 0 0 0 0 0 /odom /base_frame 100" />

  <!--Hector Mapping setup-->
    <node pkg="hector_mapping" type="hector_mapping" name="hector_mapping" output="
    screen"/>

</launch>
```

Listing 3.1: mapping_config.launch

In this launch files we launch the URG node for the LiDAR data, setup static transforms between the laser, odometry and base frames. Then we launch hector mapping. Before running this launch file we need to install hector mapping with:

```
$ sudo apt-get install ros-kinetic-hector-mapping
```

now run the launch file with:

```
$ roslaunch myFirstPackage hector_mapping.launch
```

If it all worked, you should have mapping working, now in another terminal run rviz and visualize the mapping output. If everything is running correctly you should have your first ROS publisher, subscriber system. These can be easily visualized using the RQTgraph tool. In a new terminal tab run:

```
$ rqtgraph
```

RQT graph is a great way of creating images for reports with minimal effort.

Chapter 4

Real Sense

Following the guide, install the real sense libs and launch the real-sense node. The Cameras we are using are the R200 cameras.
`http://wiki.ros.org/librealsense`
`http://wiki.ros.org/realsense_camera`
Now use Rviz to visualize the depth, and camera images.