Map Building And Autonomous Navigation Using The Ros Navigation Stack And The Turtlebot

Technical Report for the European Masters in Robotics (EMARO) program at École Centrale de Nantes

Submitted by
Matthew Stein
Roger Williams University
mstein@rwu.edu
Visiting scholar in the EMARO program
June 28, 2017

Abstract
This report contains instructions and technical details for accomplishing the two related goals of building a map of an unknown environment and navigating in that environment using the constructed map. Both tasks use the ROS navigation stack as the underlying technology, with either mapping or navigation overlaid. This report describes implementation on the Yujin Turtlebot equipped with Hokuyo Scanning Laser Range Finder (LiDAR) and without the original equipment Xbox Kinect sensor (sensor was disconnected). In addition to presenting implementation procedures, this report contains some commentary on system performance.

Index des illustrations

Introduction
The experimental setup employs two computers communicating by local wireless. The ROS software for control is distributed between the two computers shown in Illustration 1. Using the nomenclature consistent with the ROS Turtlebot tutorial: (http://wiki.ros.org/turtlebot_navigation/Tutorials/Build A Map With SLAM) the desktop computer will be referred to as “On Your Workstation” and the on-board ASUS notebook as “On the

Turtlebot". The ASUS notebook connects via USB cable to the Kobuki base and to the Hokuyo LiDAR. The network cable from the rear of desktop computer is unplugged and connected to the NETGEAR router, while an additional cable (found in the router box) connects the router to the network receptacle of the workstation.

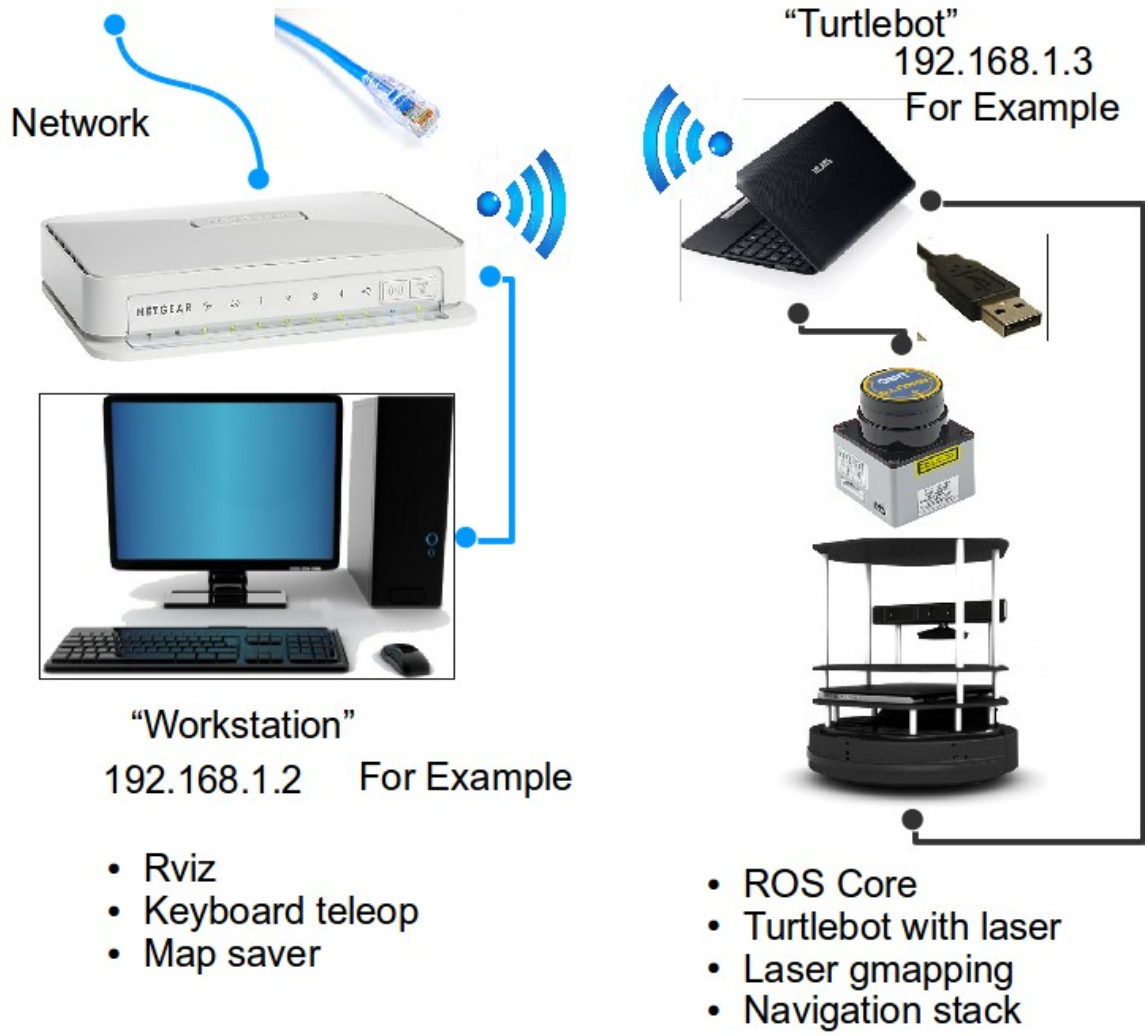


Illustration 1: Experimental Setup Pictorial Description.

Part I Creating a Map with LiDAR

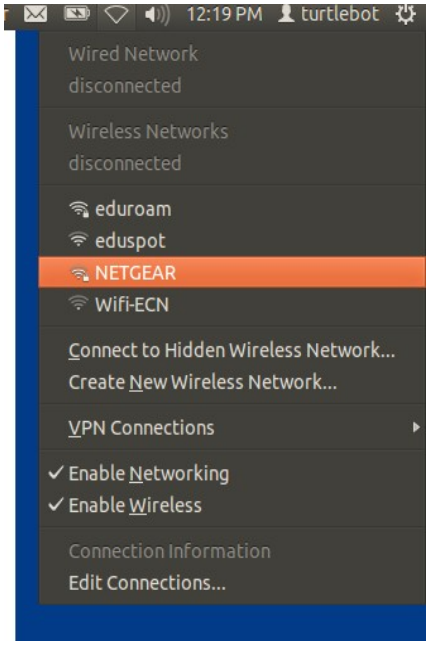
Step 1 Establish Wireless Network Connection

No additional configuration is required for the desktop computer. On startup the desktop computer detects its wired network connection through the router and establishes Internet connection as normal. It is sometimes a little slow to do this, but after a few minutes it works normally without any other user action.

The ASUS notebook must be manually connected to the wireless hub. With the screen of the ASUS notebook open, log into the notebook with:
username: turtlebot
password: turtlebot

The default network name for the wireless hub established by the NETGEAR router is "NETGEAR". This should appear in the list of available networks menu as shown in Illustration 2.

Illustration 2: ASUS Notebook connecting to the wireless network: NETGEAR.



Connecting the first time will require a **network passkey: netgear;ecn**

To test if the wireless connection is working, open up Firefox (or any browser) to localhost, in other

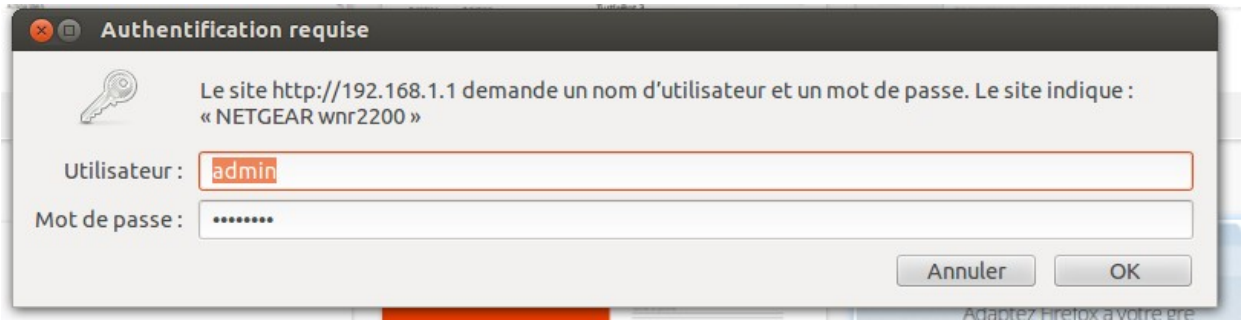


Illustration 3: Authentication dialig for NETGEAR router.

words the address: http://192.168.1.1 this may bring up the authentication dialog shown in Illustration 3. The username and password are the factory default:

username: admin
password: password

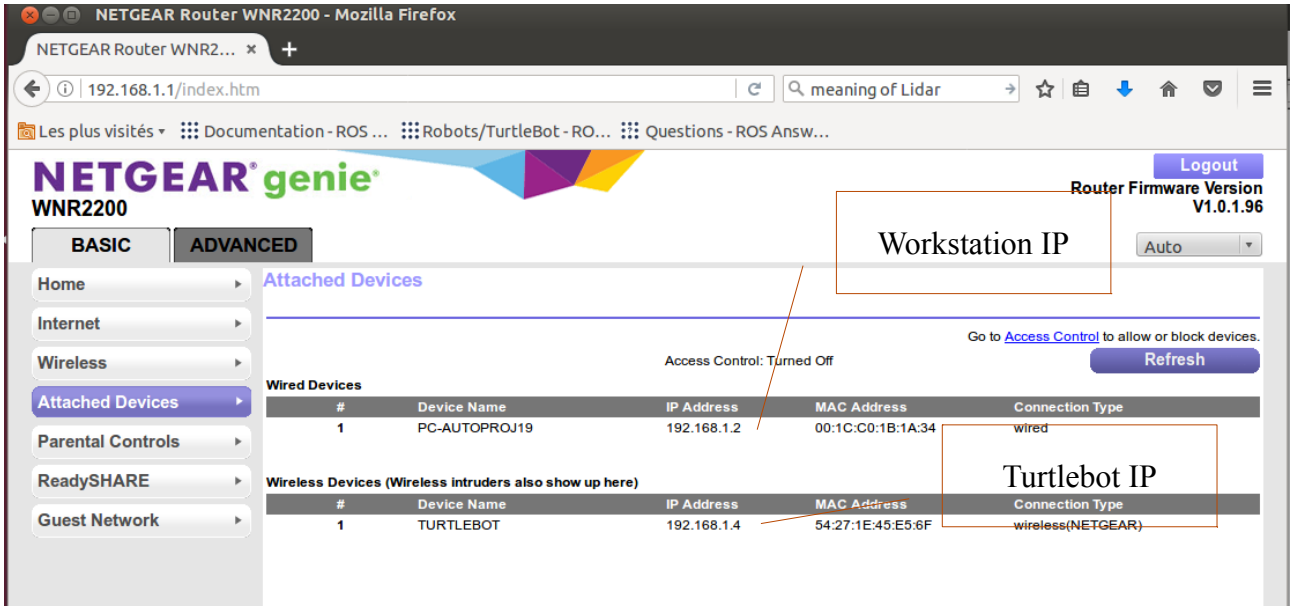


Illustration 4: Router control panel in browser showing connected devices and TURTLEBOT_IP.

The router control panel shown in Illustration 4 shows there are two devices connected. Take note of the IP address of the device named TURTLEBOT. This address is subject to change depending on boot sequence, etc. In all locations in this document where the string *TURTLEBOT_IP* appears, replace this string with the IP address shown in this panel. In all locations in this document where the string *WORKSTATION_IP* appears, replace this string with the IP address shown in this panel.

If both the name of the desktop computer and the TURTLEBOT appear, the network is established, you may close the ASUS notebook and proceed to Step 2. Because the workstations are shared laboratory computers, no attempt was made to create names associated with these IP addresses in a hosts database. These addresses do not change often during experimentation and changes are accomodated with straightforward edits to .bashrc files.

Step 2 Set ROS networking parameters

On the turtlebot, define the following environment variables, for example:
ROS_HOSTNAME="192.168.1.4"

It may be convenient to edit the ~/.bashrc file and place these lines at the bottom:

```
export ROS_MASTER_URI=http://localhost:11311
export ROS_HOSTNAME=TURTLEBOT_IP
```

Turtlebot

On the workstation, also edit the ~/.bashrc file and place the following lines at the bottom:

```
export ROS_MASTER_URI="http://TURTLEBOT_IP :11311"
export ROS_HOSTNAME="WORKSTATION_IP"
export LIBGL_ALWAYS_SOFTWARE=1
```

Workstation

Note that the last command is included to avoid poor performance of rviz on the laboratory desktop. This option may not be necessary on a different computer, but without it the rviz window showed dropouts and noise resembling black snow.

Step 3 Bring Up Turtlebot

Open two terminals on your workstation and connect to the turtlebot using the following command:

```
$ ssh -l turtlebot TURTLEBOT_IP
for example :
ssh -l turtlebot 192.168.1.4
```
Workstation

when requested, the password is also **turtlebot.** In one window, bring up the core ROS nodes of the turtlebot with:

```
$ roslaunch turtlebot_bringup minimal_laser.launch
```
Turtlebot

If all goes well the screen will resemble Illustration 5 and the turtlebot will chirp pleasantly. Note that the file minimal_laser.launch is a modified version of minimal.launch that does not initialize the Kinect sensor and instead initializes the LiDAR. Source of the minimal_laser.launch is included in the appendix A.

Step 4 Bring Up Turtlebot gmapping

Place the turtlebot in the location and orientation that you would like to correspond to the origin of the map. In the other terminal, bring up gmapping (see steps 9-11 to bring up navigation). Enter the command:

```
$  roslaunch turtlebot_navigation gmapping_laser.launch
```
Turtlebot

Illustration 6 shows the turtlebot window after successful launching of gmapping_laser.launch. The file gmapping_laser.launch is a modification of gmapping.launch with the initiatilaization of the Kinect sensor removed. A full listing is in Appendix B. Note if either of these launch files fails corrective action must be taken. If either launch file fails with the message:
Unable to contact my own server at [http://192.168.1.3:41035/].
This usually means that the network is not configured properly.
The network environment variables are improperly set. If the laser driver fails it may not be fully plugged in, look for an illuminated green LED on top of the unit.

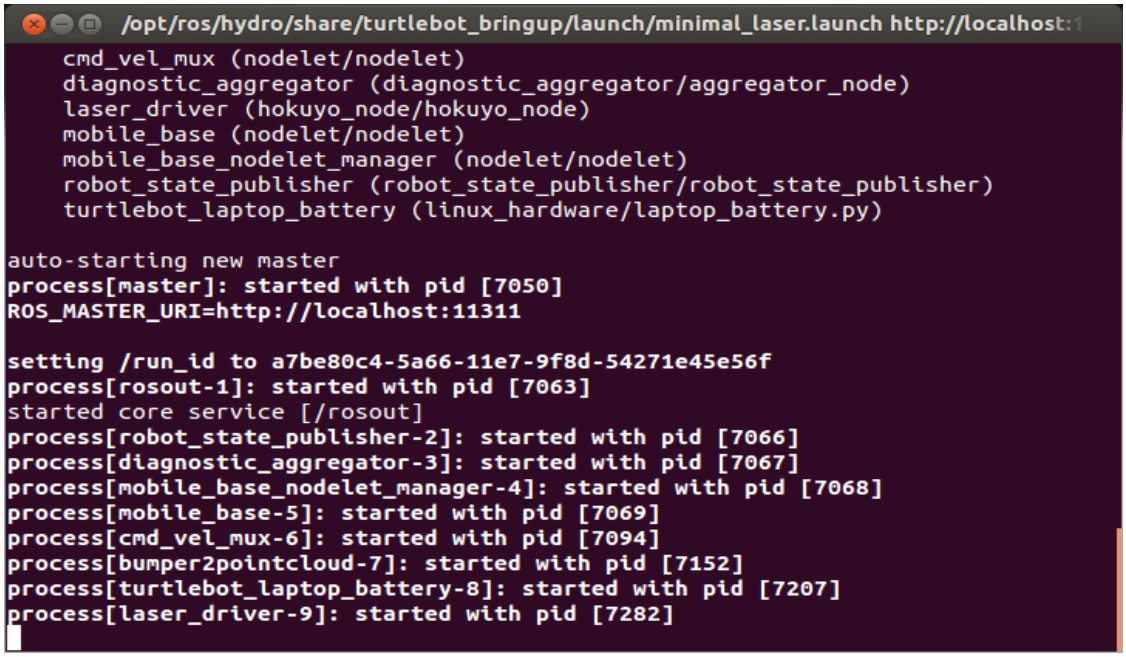


Illustration 5: Turtlebot terminal window showing successful launch of minimal_laser.launch

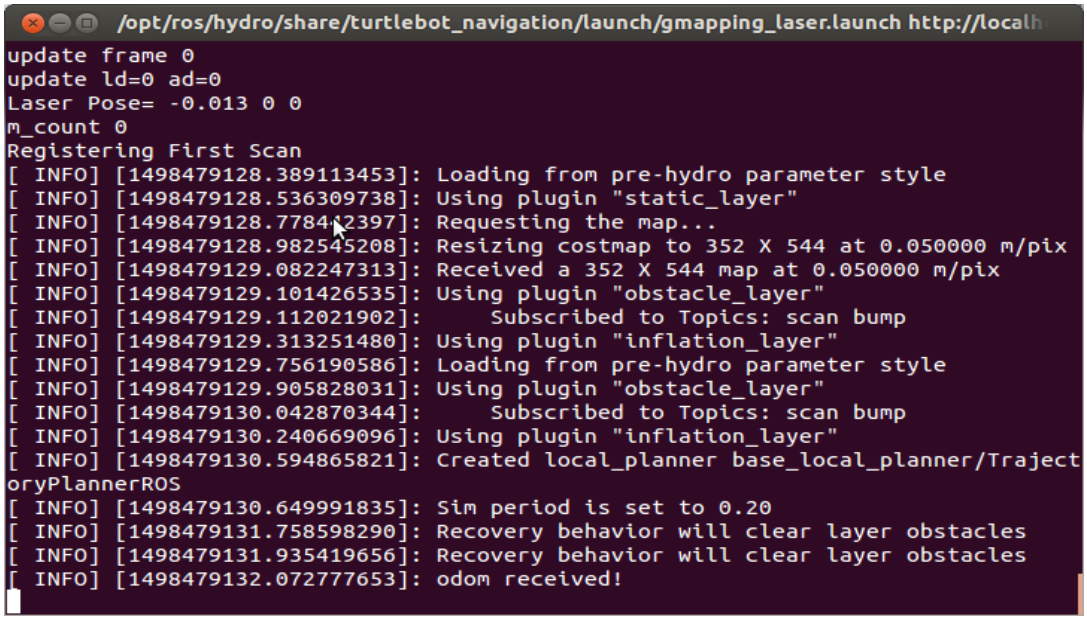


Illustration 6: Turtlebot terminal showing successful launch of gmapping_laser.launch

Step 5 Bring Up Rviz on the Workstation

Bring up rviz to view the map creation process. Any rviz will do, but the launch file of the turtlebot distribution as shown here is pre-configured to show topics of interest:

```
$ roslaunch turtlebot_rviz_launchers view_navigation.launch
```

Workstation

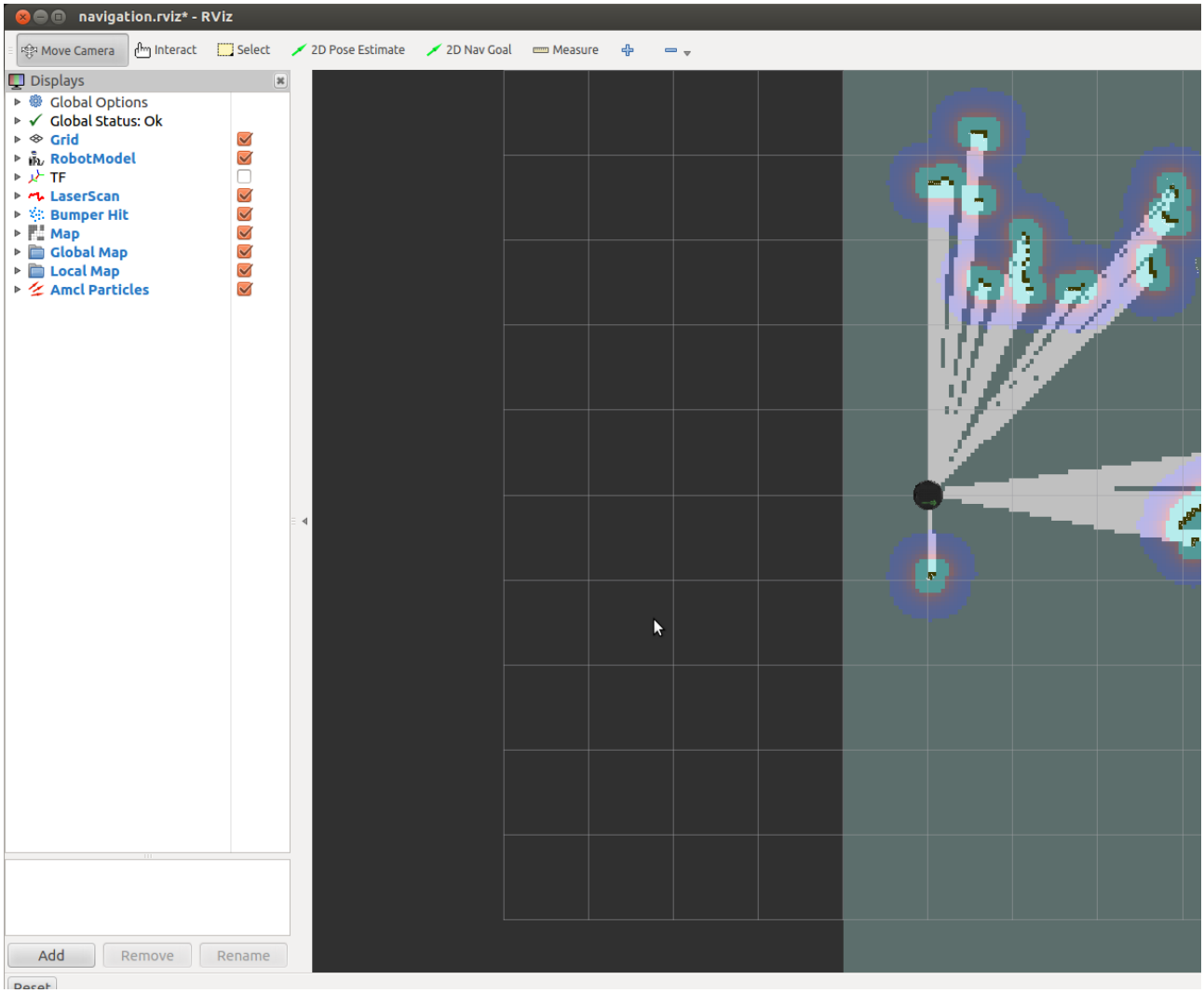


Illustration 7: Rviz window showing turtlebot in center and first scans on the map.

Illustration 7 shows the rviz screen with the turtlebot model in the center of the map and scans to the detected surfaces. If a scan sucessfully returns a distance reading, the cells up to that distance are marked clear. If no reliable distance is returned the cells remain unknown.

Step 6 Bring Up Turtlebot Teleoperation

The fun part is to now drive the turtlebot around and watch the map being constructed. Any teleoperation can be used, but I found default keyboard teleoperation was entirely adequate for this purpose. In another terminal on the workstation (a total of 5 terminals will be required) issue the command:

```
$ roslaunch turtlebot_teleop keyboard_teleop.launch –screen
```

Workstation

(note the double dash before the screen argument). This produces the screen shown in Illustration 8. As the instructional text in the window indicates, the turtlebot may now be teleoperated with the keyboard. Use the listed keys to drive the turtlebot and observe the map being formed, as shown in Illustration 9. Note that the turtlebot may be navigated to previously visited areas to improve the map, as was done to produce Illustration 10.

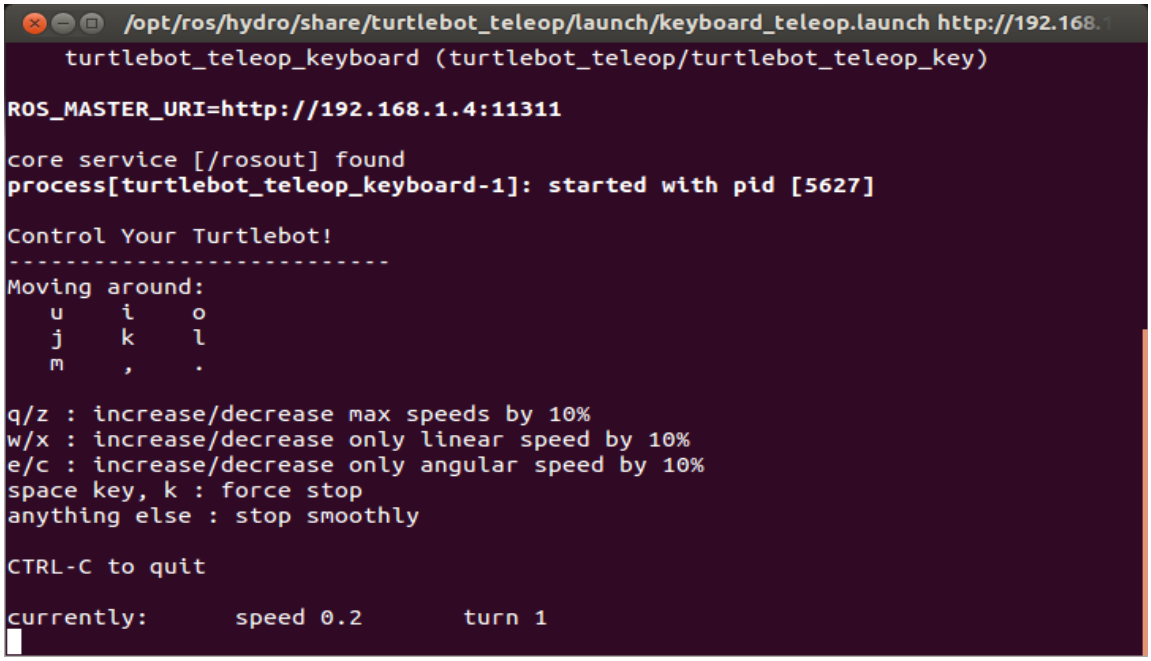


Illustration 8: Workstation window showing successful launch of keyboard_teleop.launch.

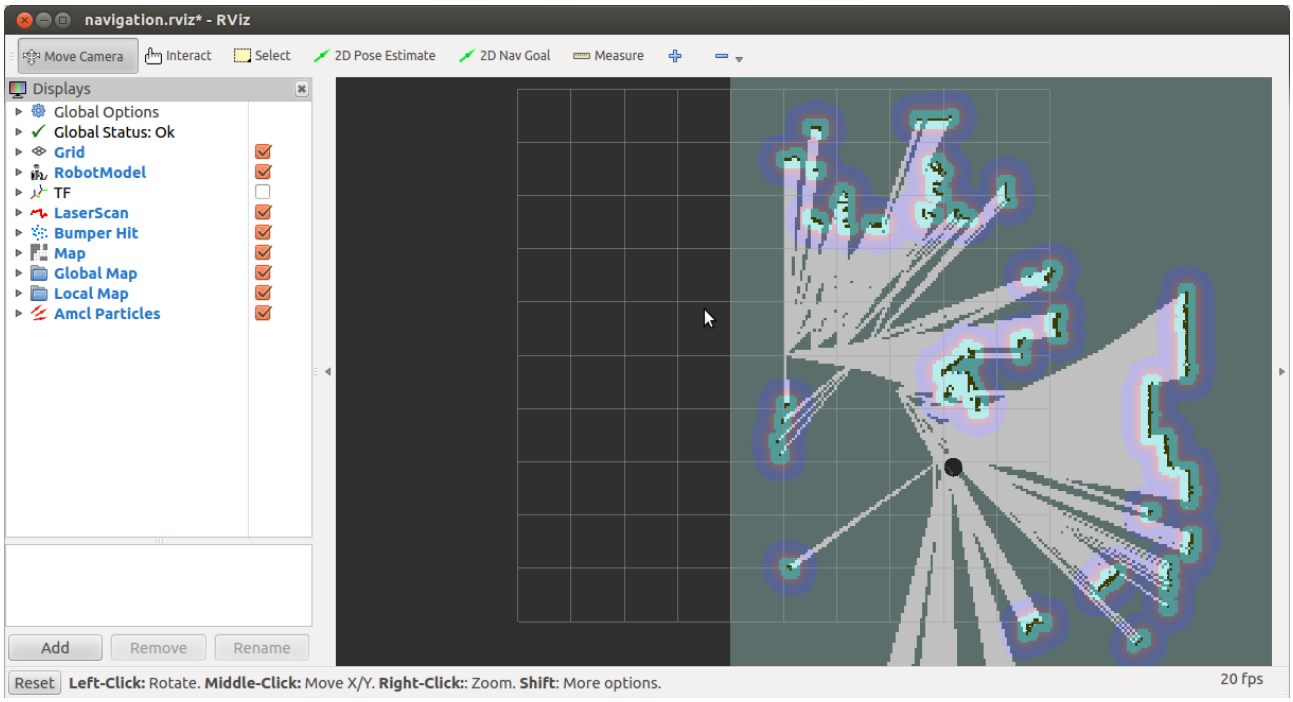


Illustration 9: Rviz window after turtlebot has begun to move. Note some clear cells were initially missed.

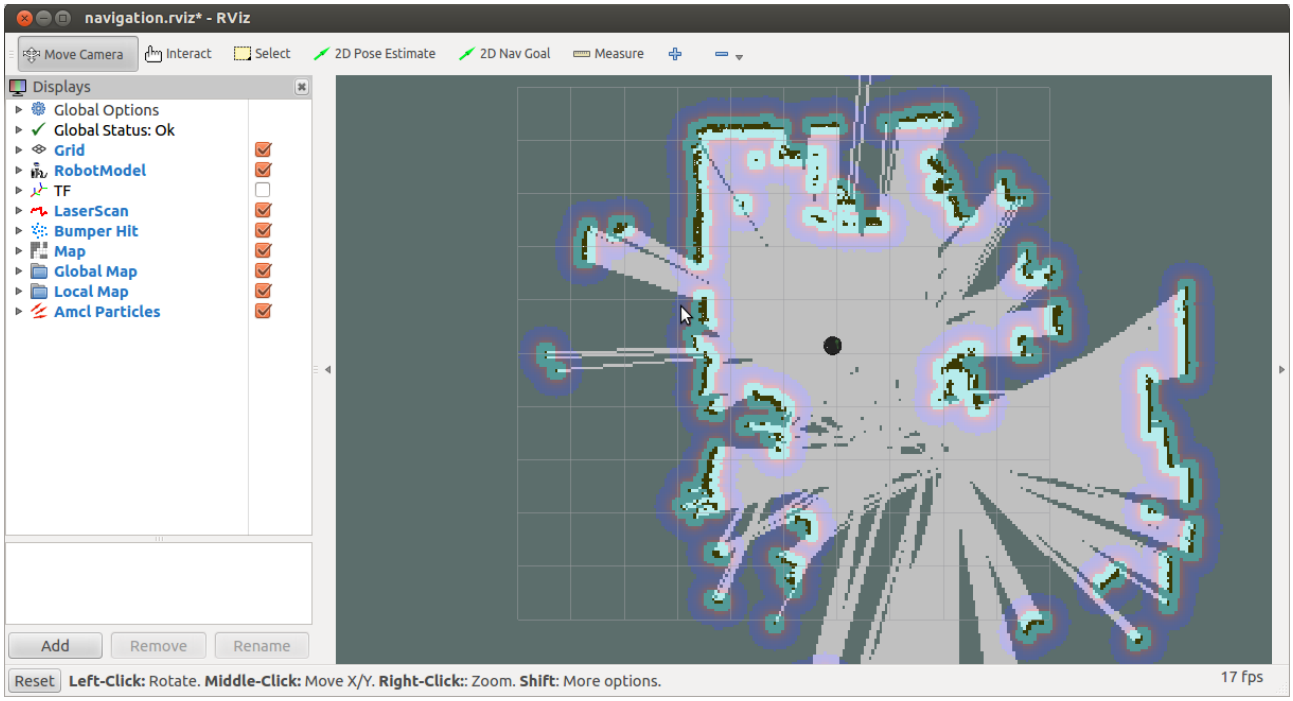


Illustration 10: Rviz window after turtlebot has been driven to return to previously covered areas.

Step 7 Save The Map

You may notice that the gmapping process has been reporting progress the whole time, as shown in Illustration 11. Gmapping will continue until the process is aborted in the terminal window. At any time the map may be saved and re-saved without interfering in the gmapping process, with following command on the workstation:

```
$ rosrun map_server map_saver -f /tmp/my_map
```

Workstation

```
[ INFO] [1498481082.378918189]: Waiting for the map
[ INFO] [1498481082.894432755]: Received a 576 X 544 map @ 0.050 m/pix
[ INFO] [1498481082.894599610]: Writing map occupancy data to /tmp/my_map.pgm
[ INFO] [1498481082.914735557]: Writing map occupancy data to /tmp/my_map.yaml
[ INFO] [1498481082.914982458]: Done
```

where “my_map” can be any name you specify. Note that two files are generated, a my_map.pgm file and a my_map.yaml file. The my_map.pgm file contains the bitmap graphics and this file can be directly edited by any image editing software, for example Gnome Paint. Black indicates occupancy, gray clear and green uncertain. The my_map.yaml is a small file containing information about the image but also has the hard-coded name of the image file. Therefore, if you change the name of the map file be certain to edit the yaml file to make the corresponding name change and then also change the name of the yaml file itself.

Step 8 (optional) Crop The Map

By default the map contains an abundance of blank space around the area of detected surfaces, as seen on the right image of Illustration 13. Well hidden in the “scripts” folder of the map_server git (Illustration 12) is an undocumented utility to crop the map, removing the blank spaces around the map. Illustration 13 shows the map after and before cropping. Issue the command below to create the files cropped.pgm and cropped.yaml.

```
$ python crop_map.py my_map.yaml
```

Workstation

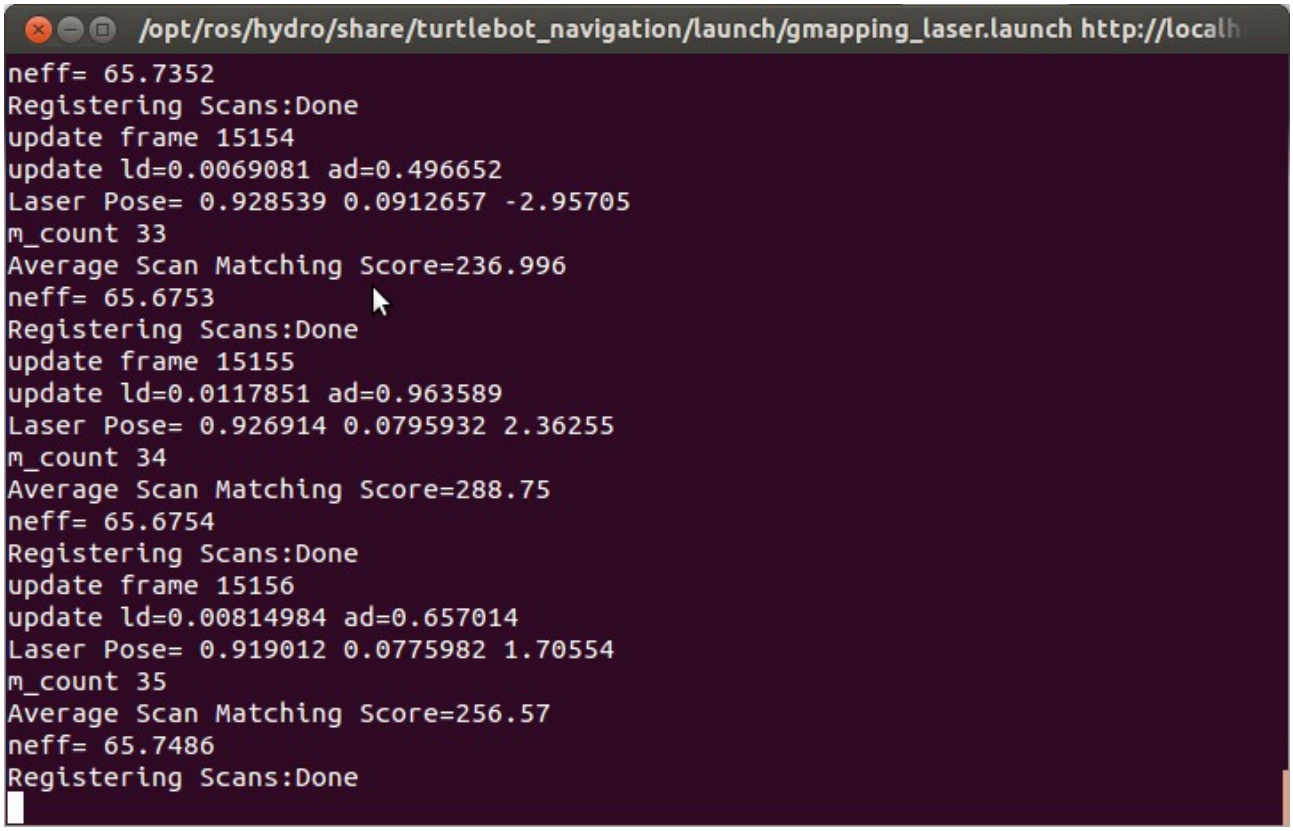


Illustration 11: Turtlebot terminal window showing output of gmapping node.

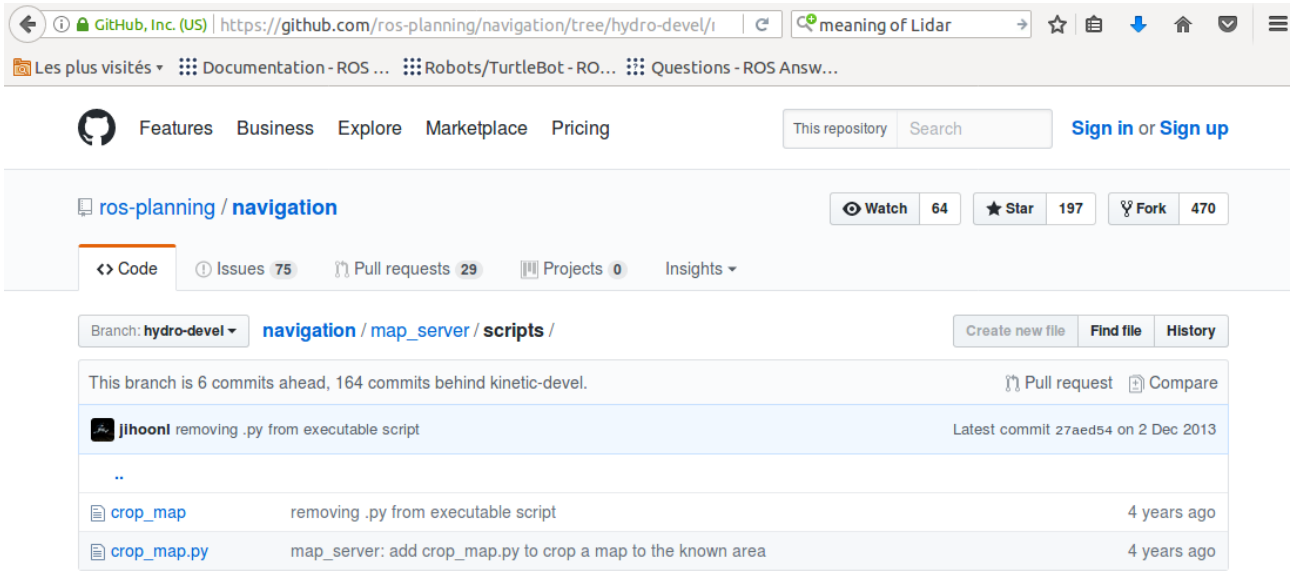


Illustration 12: Location of the undocumented but useful utility to crop blank areas off the map.

Part II Navigation of a known map with the ROS navigation stack

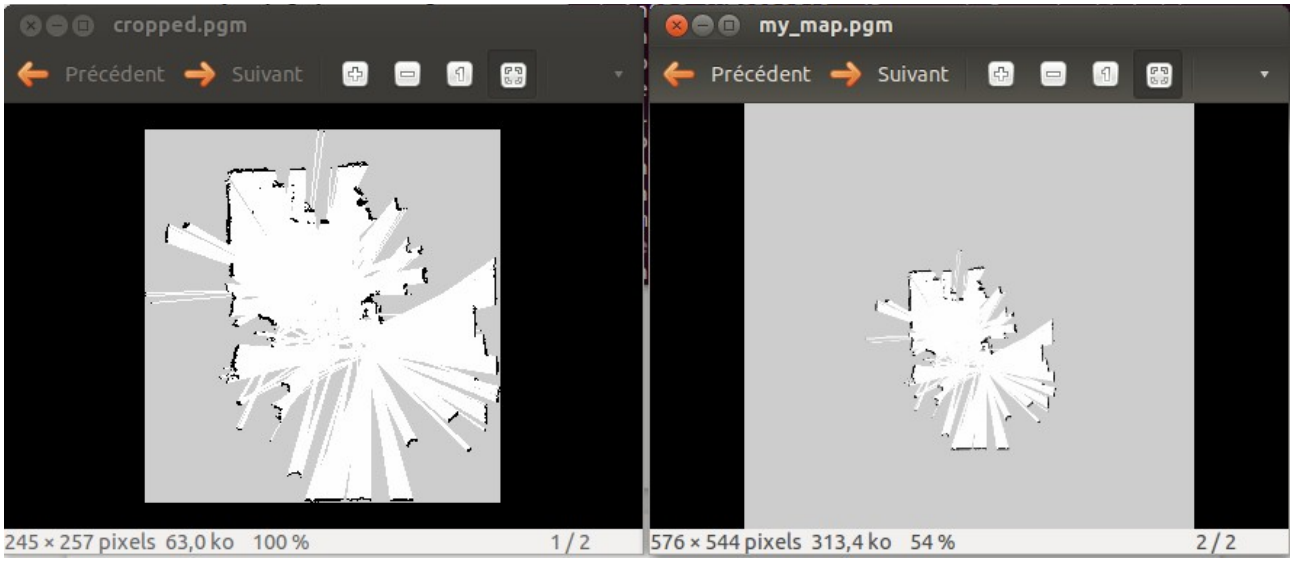


Illustration 13: Map before and after processing by the crop_map.py utility.

This section presumes you have created and stored a map in Part I. Note that the map shown in Illustration 14 (Niveau 1 of Building D on the École Centrale de Nantes campus) has been manually edited in two ways. First, thick black lines were added along exterior walls where laser light passes through transparent windows and did not register the walls as solid. The navigation stack treated these cells as clear and, in seeking the shortest path, created trajectories that passed through the interior courtyard. Also note that an obstruction was artificially added near the passageway between the first and second rooms; appearing as a sold black square. This kluge, highlighted in red, was added because the turtlebot repeatedly encountered chair legs in that location that are invisibe to the LiDAR sensor.

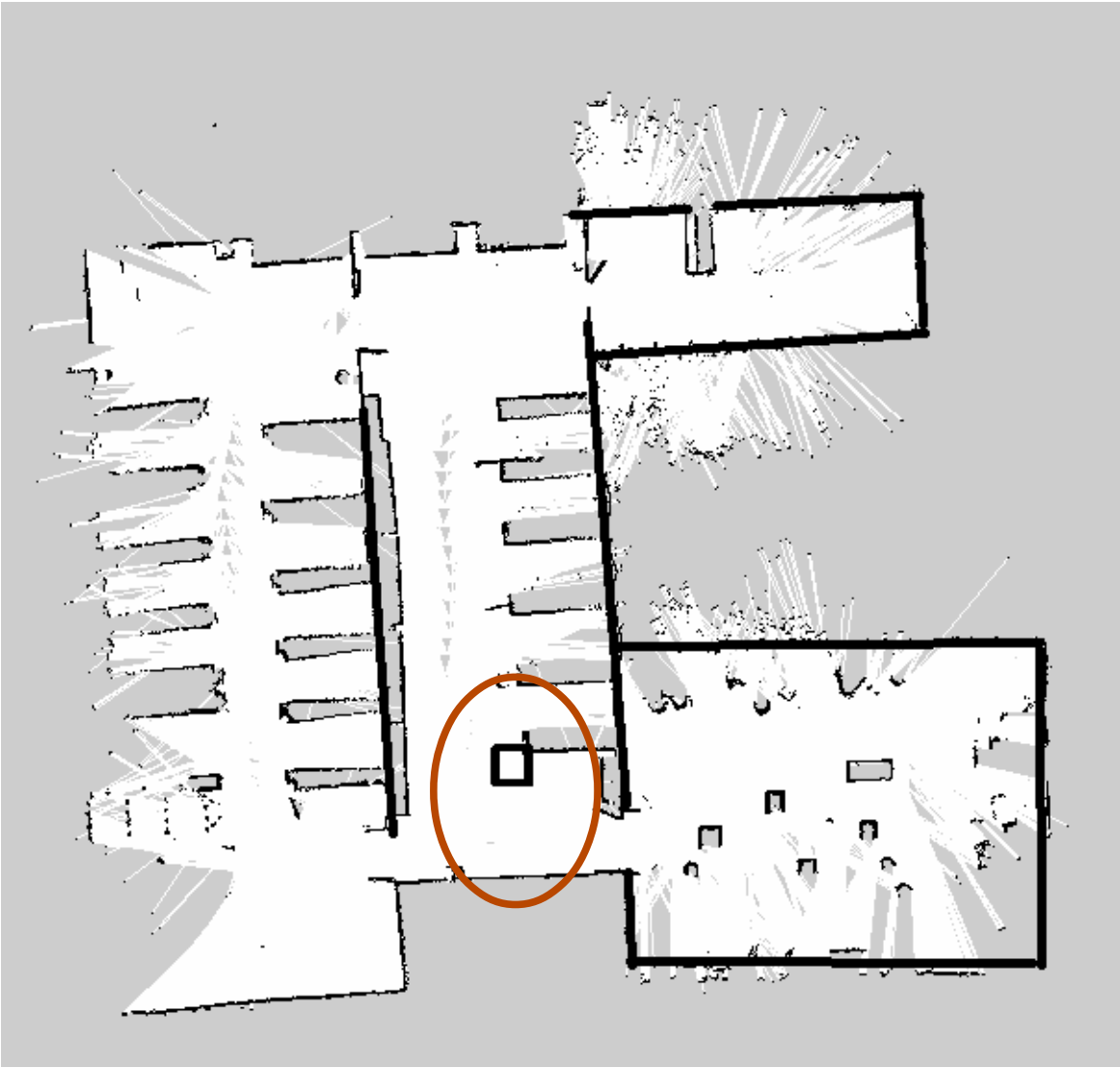


Illustration 14: A map of three rooms and a hallway in building D of ECN. Note that some walls have been manually thickened and note the artificial object circled in red.

Step 9 Copy The Map To The Turtlebot

The navigation stack executes on the ASUS notebook loading the fixed map on initiation of the map_server node. It is therefore necessary to transfer the map and corresponding yaml file from the workstation to the turtlebot. Create a new folder on the turtlebot to locate these files. In this example, I have created the folder turtle_nav on the turtlebot as ~/turtle_nav.

I saved the hand-modified map into the file **labs.png (**Note that the map_server can read many image formats and is not limited to .pgm type files). Copy the map files to the turtlebot using the **rcp** command on the workstation and enter password when prompted. Do the same for the yaml file.

```
PC-AUTOPROJ19:~/Documents/turtlebot$ rcp labs.png turtlebot@TURTLEBOT_IP:labs.png
turtlebot@TURTLEBOT_IP's password:
```

Workstation

The contents of the yaml file are shown here, note that the map file name is explicitly listed, as highlighted in red.

File labs.yaml

```
free_thresh: 0.196
image: labs.png
negate: 0
occupied_thresh: 0.65
origin: [-30.55, -6.6000000000000005, 0.0]
resolution: 0.05
```
Turtlebot

The launch file turtle_nav.launch is shown here. Note that I have entered the absolute file path in the arguments section of the map_server node, highlighted in red below. It is an absolute path because I lost patience trying to figure out why the relative path would not work.

File turtlebot_nav.launch

```
<launch>

  <!-- Map server-->
  <arg name="map_file" default="/home/turtlebot/turtle_nav/labs.yaml"/>
  <node name="map_server" pkg="map_server" type="map_server" args="$(arg map_file)" />

  <arg name="initial_pose_x" default="0.0"/> <!-- Use 17.0 for willow's map in simulation -->
  <arg name="initial_pose_y" default="0.0"/> <!-- Use 17.0 for willow's map in simulation -->
  <arg name="initial_pose_a" default="0.0"/>

  <include file="$(find turtlebot_navigation)/launch/includes/amcl.launch.xml">
    <arg name="initial_pose_x" value="$(arg initial_pose_x)"/>
    <arg name="initial_pose_y" value="$(arg initial_pose_y)"/>
    <arg name="initial_pose_a" value="$(arg initial_pose_a)"/>
  </include>

  <include file="$(find turtlebot_navigation)/launch/includes/move_base.launch.xml"/>

</launch>
```
Turtlebot

Step 10 Start The Navigation Stack

Terminate the gmapping process by typing ^C in the terminal window or manually stopping the nodes with the rosnode kill command. Also terminate the teleoperation node on the workstation as this may interfere with move_base.

Note that the launch file above also launches AMCL and move_base, both highly complex software structures with the default set of parameters. These parameters may be viewed in the include files indicated. Also see the ROS navigation WIKI for a pretty good description of the numerous parameters defined. Launch the navigation stack by launching the above file:

```
$ roslaunch turtle_nav.launch
```
Turtlebot

Note that if the core ROS nodes launched in Step 3 have stopped this will generate an error message. If so, I have found that killing all nodes and then restarting with Step 3 followed by launching turtle_nav.launch usually works.

Step 11 Bring Up Rviz Again

Although repeating Step 5 here works perfectly well, I have found that displaying a few more topics is helpful for navigation. Illustration 15 shows an rviz session that is executed by running **rviz** alone

```
$rviz
```

Workstation

and then loading a configuration file that was previously saved in a local folder.

Navigation begins by selecting the 2D Pose Estimate button and placing the green arrow at the position and orientation of the turtlebot on the map. In a second or two you will see the cloud of points from AMCL move to the location followed by the turtlebot model jumping to that location. You may do this any number of times before creating a navigation goal. Select the 2D Nav Goal button and click on a target location and orientation to start the navigation. This process may be repeated as long as batteries hold. The turtlebot's global path plan and local path plan will be visible on the map as the turtlebot navigates.

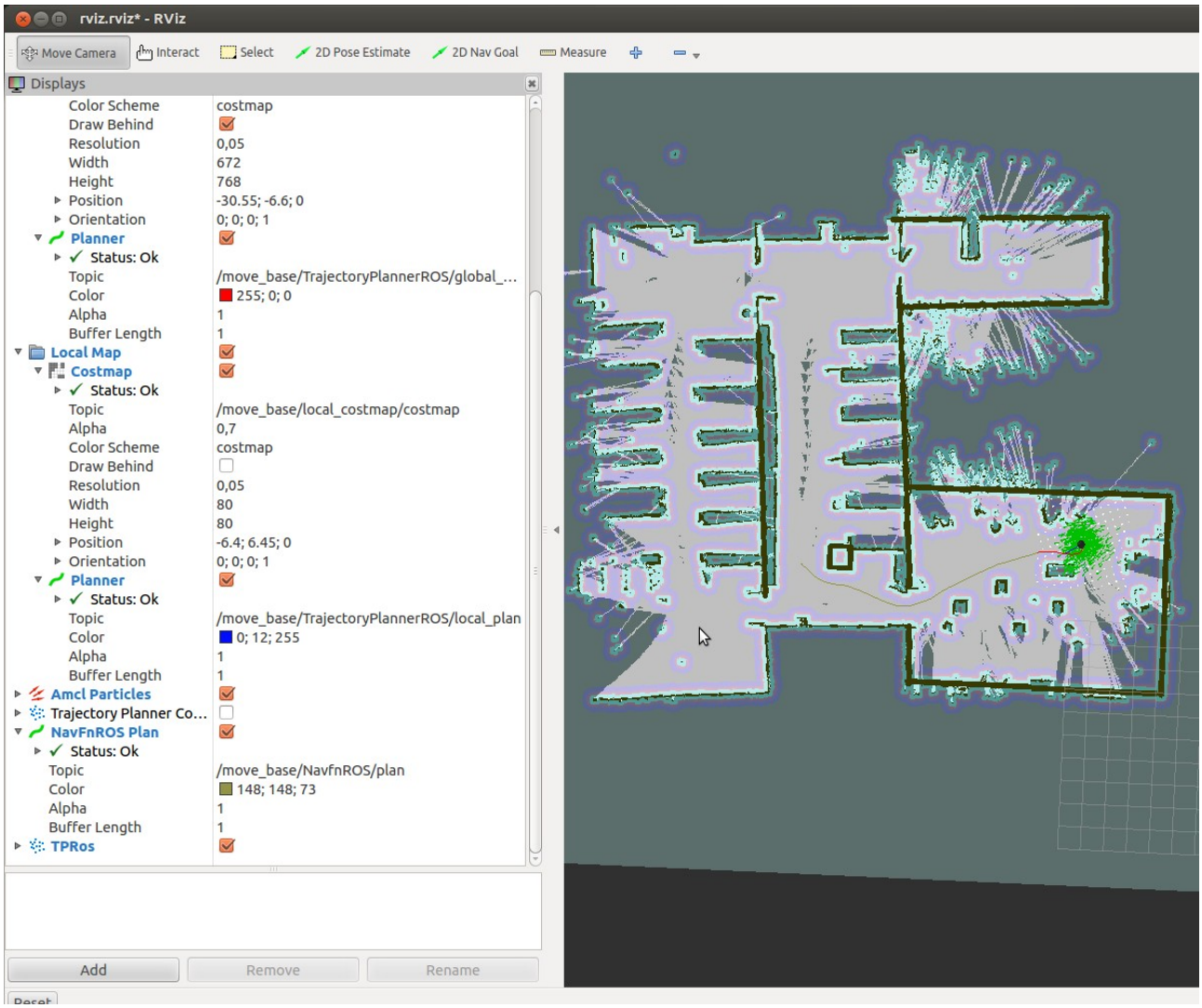


Illustration 15: Rviz window running the navigation stack. Initial turtlebot location was set with 2D Initial Pose and navigation goal was set using 2D Nav Goal.

Observations

I made the following observations while directing the turtlebot to navigate around the niveau. These are included here only for reference and for potential assistance in understanding navigation.

- The laser scans are visible as flickering white dots in rviz, but these scans are used only by localization and obstacle detection. Thus the turtlebot will not avoid new or moving objects that it can clearly detect but are not on the map.
- Thin objects such as the legs of chairs are rarely if ever seen with enough persistence to register on the map. Thus the turtlebot often collides with these if you do not take steps to either remove the chair or add it to the map.
- The turtlebot will respond to activation of its front bumpers. Activation appears as a red dot and an occupancy appears in the local cost map. The turtlebot attempts some form of spinning maneuver on bumper activation, but as this is an ad hoc behavior it sometimes works wonderfully and other times fails just as spectacularly.
- There seems to be no issues with battery life, as I have sent the turtlebot from room to room for hours without recharging. I have encountered no limitation due to the range of the NETGEAR router.
- I made an informal comparison of the sensors by mapping the same area using the Kinect sensor (the turtlebot's as-shipped configuration). Subjectively, the map made by the Kinect was less crisply defined than the map made by the LiDAR. There was more distortion of the map, visible, for example, as oblique angles between the rooms that are at right angles. As would be expected, the smaller sweep angle of the Kinect required more in-place rotations of the turtlebot to scan the entire room.

Conclusions

The primary conclusion of this study is that the capabilities promised in the ROS tutorial are for the most part a reality on the turtlebot. The turtlebot navigation package available in the ROS distrbution required very little customization and works well. The few modifications required included:

- Physically adding the LiDAR sensor and disconnecting the Kinect.
- Modifying launch files to remove initialization of the Kinect and instead initialize the LiDAR.
- Re-direct ROS topics so that the LiDAR provides the /scan topic.

with these accomplished the gmapping and navigation worked admirably with the default parameters.

Appendices

Appendix A File minimal_laser.launch

Placed in turtlebot folder /opt/ros/hydro/share/turtlebot_bringup/launch

```
<launch>
  <arg name="base"       default="$(optenv TURTLEBOT_BASE kobuki)"/>  <!-- create, rhoomba
-->
  <arg name="battery"    default="$(optenv TURTLEBOT_BATTERY
/sys/class/power_supply/BAT0)"/>  <!-- /proc/acpi/battery/BAT0 in 2.6 or earlier kernels-->
  <arg name="stacks"     default="$(optenv TURTLEBOT_STACKS hexagons)"/>  <!-- circles,
hexagons -->
  <arg name="3d_sensor"  default="$(optenv TURTLEBOT_3D_SENSOR kinect)"/>  <!-- kinect,
asus_xtion_pro -->
  <arg name="simulation" default="$(optenv TURTLEBOT_SIMULATION false)"/>

  <param name="/use_sim_time" value="$(arg simulation)"/>

  <include file="$(find turtlebot_bringup)/launch/includes/robot.launch.xml">
    <arg name="base" value="$(arg base)" />
    <arg name="stacks" value="$(arg stacks)" />
    <arg name="3d_sensor" value="$(arg 3d_sensor)" />
  </include>
  <include file="$(find turtlebot_bringup)/launch/includes/mobile_base.launch.xml">
    <arg name="base" value="$(arg base)" />
  </include>
  <include file="$(find turtlebot_bringup)/launch/includes/netbook.launch.xml">
    <arg name="battery" value="$(arg battery)" />
  </include>
  <node name="laser_driver" pkg="hokuyo_node" type="hokuyo_node">
   <param name="frame_id" value="base_laser_link" />
  </node>
</launch>
```

Appendix B File gmapping_laser.launch

Placed in turtlebot folder /opt/ros/hydro/share/turtlebot_navigation/launch

```
<launch>
<!-- Removed by Matthew Stein June 2017
  <include file="$(find turtlebot_bringup)/launch/3dsensor.launch">
    <arg name="rgb_processing" value="false" />
    <arg name="depth_registration" value="false" />
    <arg name="depth_processing" value="false" />
 -->

    <!-- We must specify an absolute topic name because if not it will be prefixed by "$(arg
camera)".
         Probably is a bug in the nodelet manager: https://github.com/ros/nodelet_core/issues/7
    <arg name="scan_topic" value="/scan" />
  </include>
-->
```

```
  <include file="$(find turtlebot_navigation)/launch/includes/gmapping.launch.xml"/>
  <include file="$(find turtlebot_navigation)/launch/includes/move_base.launch.xml"/>
</launch>
```