Lab #2 and ROS Tools

CS/EE/ME 134

05/02/18

ROS Debugging Tools

1. rosrun rqt_tf_tree rqt_tf_tree
   - Do necessary components exist, and are they connected properly?
2. rqt_graph
   - Are all nodes communicating properly across the right topics?
3. rostopic echo … (or rostopic info ...)
4. rviz → add in the frames/topics you would like to read from)
   - If you want to see laser scans, add the “laser” frame to your rviz and display the /scan topic associated with it
5. Read Wiki/APIs for ROS nodes you are using!!
   - Make sure parameters are defined as you want them
   - Some nodes require certain transformations in the *tf* tree to exist

Lab #2 – Part 1

1. Bring up the Turtlebot. As you saw in the mobile_base.launch.xml file, this has been modified to launch the move_base node.

   $ roslaunch turtlebot_bringup minimal.launch

2. Run the laser scanner. In ROS Kinetic, the command is slightly different as shown below. Be sure to set permissions for the laser scanner as you did in the previous lab.

   $ sudo chmod a+rw /dev/ttyACM0

   $ rosrun hokuyo_node hokuyo_node

3. Setup the transforms for the laser frame using tf_setup.py. You can find the python script here.

4. Run the amcl node in order to localize the turtlebot within the generated map. The amcl node uses odometry and laser scans to localize the turtlebot.

   $ rosrun amcl amcl _use_map_topic:=true _base_frame_id:=base_footprint

5. Run the slam_gmapping node in order to build the map of the environment.

   $ rosrun gmapping slam_gmapping _xmin:=-10 _xmax:=10 _ymin:=-10 _ymax:= 10

   _map_update_interval:=5

Turtlebot_Bringup minimal.launch

```
<launch>
  <!-- Turtlebot -->
  <arg name="base"              default="$(env TURTLEBOT_BASE)"         doc="mobile base type [create, roomba]"/>
  <arg name="battery"           default="$(env TURTLEBOT_BATTERY)"      doc="kernel provided locatio for battery info, use /proc/acpi/batte
  <arg name="stacks"            default="$(env TURTLEBOT_STACKS)"       doc="stack type displayed in visualisation/simulation [circles, hex
  <arg name="3d_sensor"         default="$(env TURTLEBOT_3D_SENSOR)"    doc="3d sensor types [kinect, asux_xtion_pro]"/>
  <arg name="simulation"        default="$(env TURTLEBOT_SIMULATION)"   doc="set flags to indicate this turtle is run in simulation mode."/
  <arg name="serialport"        default="$(env TURTLEBOT_SERIAL_PORT)"  doc="used by create to configure the port it is connected on [/dev/

  <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)" />
    <arg name="serialport" value="$(arg serialport)" />
  </include>
  <include unless="$(eval arg('battery') == 'None')" file="$(find turtlebot_bringup)/launch/includes/netbook.launch.xml">
    <arg name="battery" value="$(arg battery)" />
  </include>

</launch>
```

mobile_base.launch.xml

```
<!--
  The mobile platform base.

  Selector for the base.
 -->
<launch>
  <!-- mobile base nodelet manager -->
  <node pkg="nodelet" type="nodelet" name="mobile_base_nodelet_manager" args="manager"/>

  <!-- mobile base -->
  <arg name="base"/>
  <arg name="serialport"/>
  <include file="$(find turtlebot_bringup)/launch/includes/$(arg base)/mobile_base.launch.xml">
    <arg name="serialport" value="$(arg serialport)"/>
    <arg name="manager" value="mobile_base_nodelet_manager"/>
  </include>

  <!-- velocity commands multiplexer -->
  <node pkg="nodelet" type="nodelet" name="cmd_vel_mux" args="load yocs_cmd_vel_mux/CmdVelMuxNodelet mobile_base_nodelet_manager">
    <param name="yaml_cfg_file" value="$(find turtlebot_bringup)/param/mux.yaml"/>
    <remap from="cmd_vel_mux/output" to="mobile_base/commands/velocity"/>
  </node>

  <!-- added by Richard 04/19/18 -->
  <node pkg="move_base" type="move_base" respawn="false" name="move_base" output="screen">
    <rosparam file="$(find turtlebot_navigation)/param/global_planner_params.yaml" command="load" />
    <rosparam file="$(find turtlebot_navigation)/param/global_costmap_params.yaml" command="load" />
    <rosparam file="$(find turtlebot_navigation)/param/local_costmap_params.yaml" command="load" />
    <remap from="cmd_vel" to="cmd_vel_mux/input/teleop"/>
  </node>

</launch>
```

Lab #2 – Part 1

1. Bring up the Turtlebot. As you saw in the mobile_base.launch.xml file, this has been modified to launch the move_base node.

   $ roslaunch turtlebot_bringup minimal.launch

2. Run the laser scanner. In ROS Kinetic, the command is slightly different as shown below. Be sure to set permissions for the laser scanner as you did in the previous lab.

   $ sudo chmod a+rw /dev/ttyACM0

   $ rosrun hokuyo_node hokuyo_node

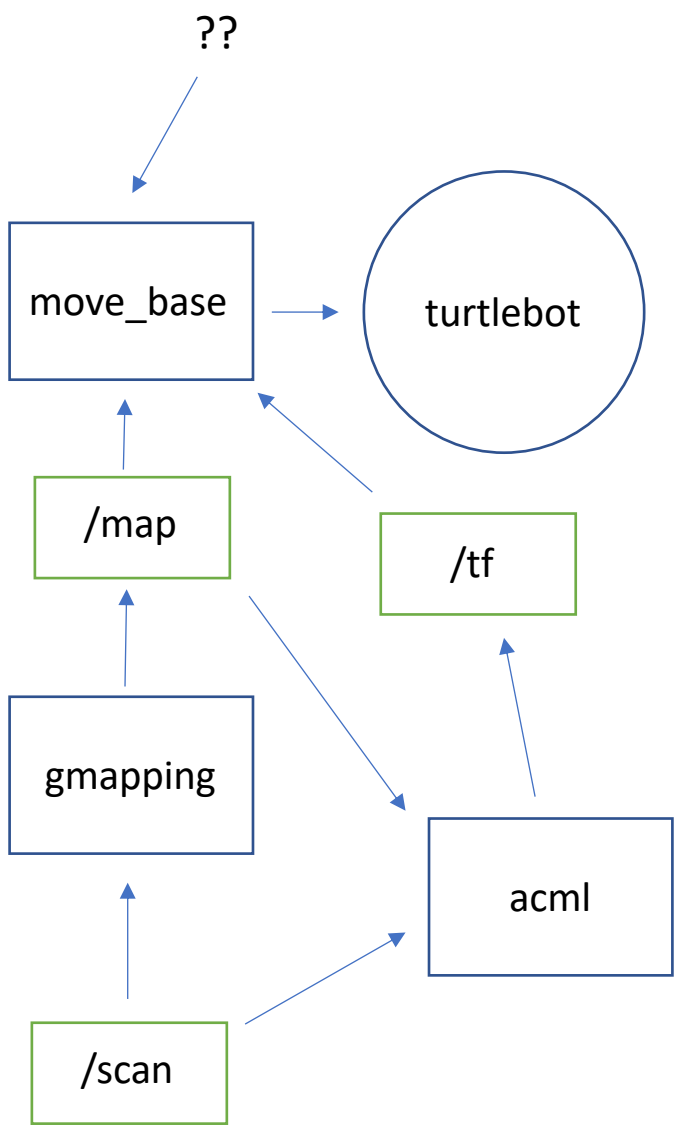
3. Setup the transforms for the laser frame using tf_setup.py. You can find the python script here.

4. Run the amcl node in order to localize the turtlebot within the generated map. The amcl node uses odometry and laser scans to localize the turtlebot.

   $ rosrun amcl amcl _use_map_topic:=true _base_frame_id:=base_footprint

5. Run the slam_gmapping node in order to build the map of the environment.

   $ rosrun gmapping slam_gmapping _xmin:=-10 _xmax:=10 _ymin:=-10 _ymax:= 10 _map_update_interval:=5

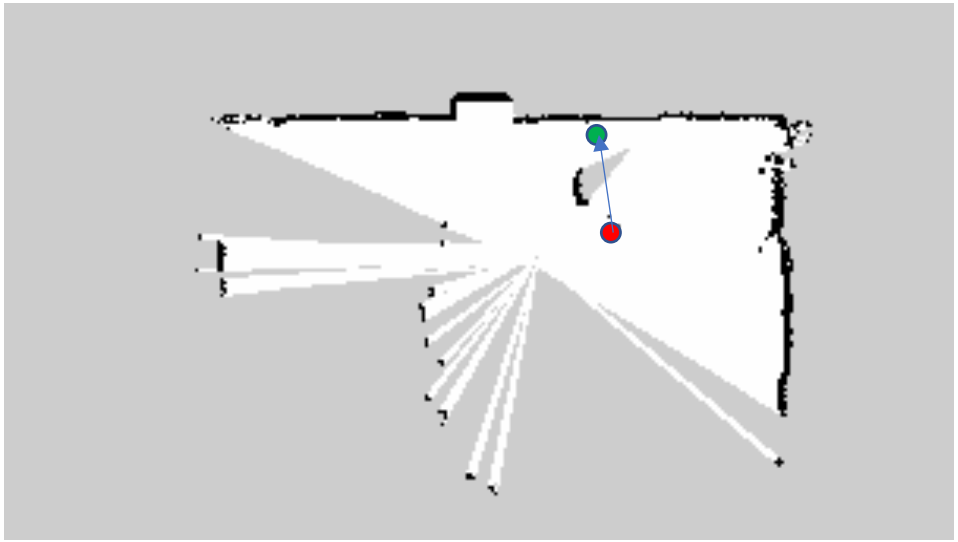


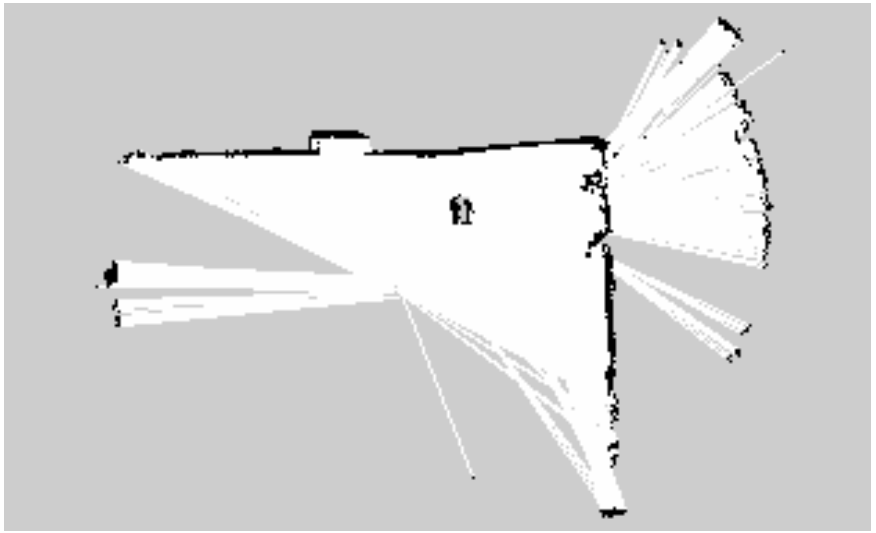
Lab #2 – Mapping out the Arena

5. Run the slam_gmapping node in order to build the map of the environment.

   $ rosrun gmapping slam_gmapping _xmin:=-10 _xmax:=10 _ymin:=-10 _ymax:= 10

   _map_update_interval:=5





For this lab, we will be mapping the arena first, and then traversing it

Lab #2 – Part 3

1. Create a folder for your group, and navigate to that folder. Start recording a bag file from that folder.

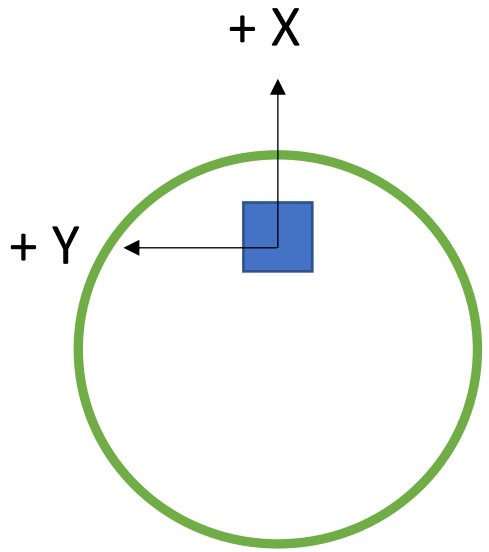
   $ rosbag record -a

2. Run the node you wrote for the pre-lab for sending a goal message to the navigation stack. If you were unable to complete that portion of the prelab, we will have a node at the lab for you to run. Initially avoid any obstacles in your first attempt.

   $ python *your_code.py*

3. Change the commanded coordinates to move around any obstacles in the arena, and build up a map of the arena.
4. Send a command that sends the turtlebot through one of the obstacles.
5. Stop recording for your bag file, and check that the expected contents are there.

   $ rosbag info *your_bag_file*



Lab slots are 30 minutes long, please come prepared and ready to run the Turtlebot using your messages!

Lab #2 – Part 4

1. Print out the rqt_graph that resulted. Are there any issues with how the nodes/topics are connected? If so, comment on what those issues are and how you would change them (be specific).

2. Print out the transform tree that resulted. Are there any issues with transform tree? If so, comment on what those issues are and how you would change them (be specific).

3. After sending your message to the goal, did the Turtlebot execute the expected plan? If not, print out both the transform tree and rqt_graph to show that the messages were being properly sent/received.

4. If you encountered any other issues during the lab, comment on what those issues were and why they arose. The goal here is to convince us that you understand the underlying processes enough to debug the issue. If you did not encounter any issues in the lab, you may skip this problem and get full credit.

5. If your turtlebot experience any collisions with obstacles during the lab, give the reasons why (be specific).

6. What is the default topic that the move_base node publishes to in order to drive the Turtlebot? What topic did it actually publish to in order to drive the Turtlebot in the lab (hint: look into the updated move_base.launch.xml file).