<https://web.fs.uni-lj.si/lakos/rosin/Documentation/getting_started/>

[Warning: the following packages need to be installed on the PC:](http://wiki.ros.org/rplidar)

ros-melodic-rosserial-server

ros-melodic-mouse-teleop

ros-melodic-teleop-twist-keyboard

ros-melodic-slam-gmapping

ros-melodic-imu-tools

ros-melodic-robot-localization

To install all the above packages, run:

sudo apt-get update

sudo apt-get install ros-melodic-rosserial-server ros-melodic-mouse-teleop ros-melodic-teleop-twist-keyboard ros-melodic-slam-gmapping ros-melodic-imu-tools ros-melodic-robot-localization

<http://wiki.ros.org/rplidar>

Steps to setup USB ad so on….

<https://github.com/Slamtec/rplidar_ros>

From: <https://github.com/husarion/rosbot_description> – DAE file available

AND

<https://husarion.com/tutorials/ros-tutorials/6-slam-navigation/>

<!-- We want the sensor functionality to be available in the ROS ecosystem, so you should install a ROS package that provides a node for the sensor you are trying to integrate. -->

https://husarion.com/tutorials/ros-tutorials/6-slam-navigation/

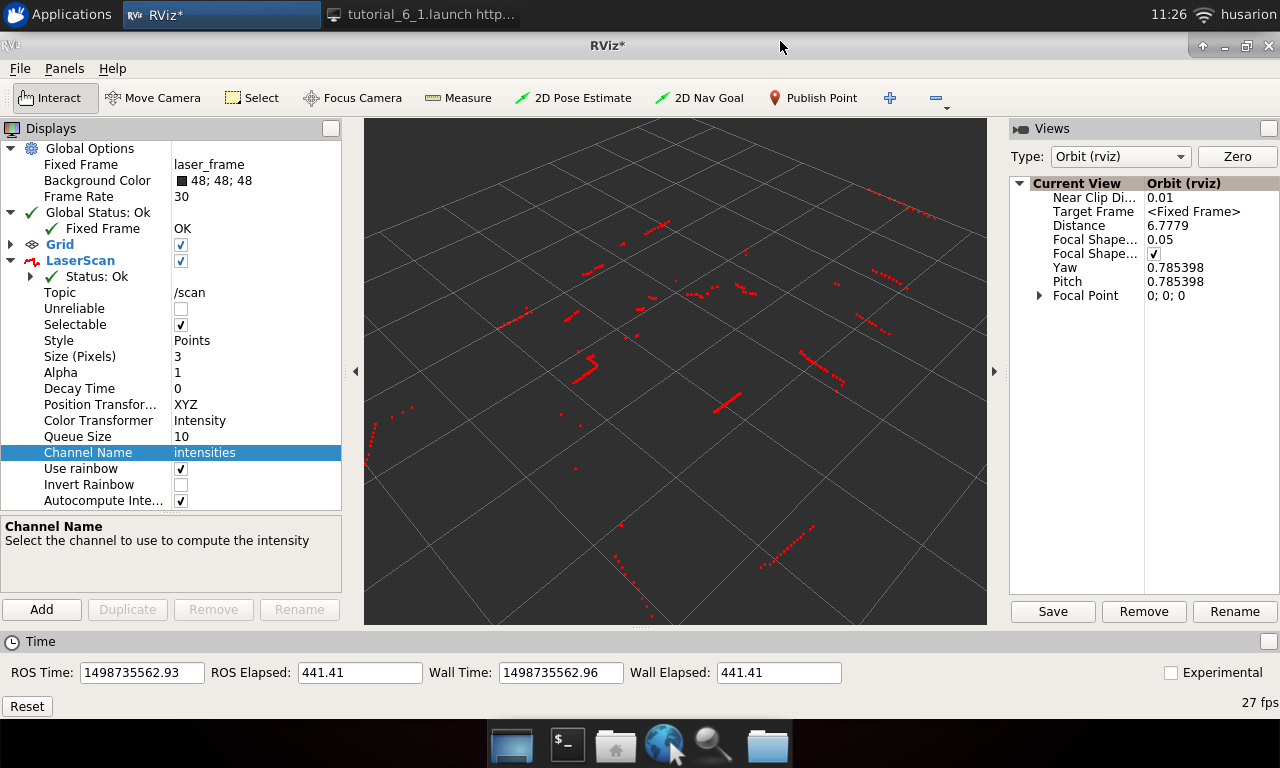
o test it you can run only this one node:

rosrun rplidar\_ros rplidarNode

For Gazebo you do not need any additional nodes, just start simulator and laser scans will be already published to appropriate topic.

In case there are no scans showing, there may be a problem with laser scanner plugin for Gazebo. Some GPUs, mainly the integrated ones have problems with proper rendering of laser scanner. To solve it, you will have to change the used plugin to CPU based. Go to file rosbot.gazebo located in rosbot\_description/src/rosbot\_description/urdf, comment out section succeeding <!-- RpLidar using GPU --> line with <!-- to begin comment and --> to end comment and uncomment section succeeding <!-- RpLidar using CPU --> line by deleting <!-- and -->.

You should have /scan topic in your system. You can examine it with rostopic info but better do not try to echo it, it is possible but you will get lots of output that is hard to read. Better method for checking the /scan topic is to use rviz. Run rviz and click Add from object manipulation buttons, in new window select By topic and from the list select /scan. In visualized items list find position Fixed Frame and change it to laser. To improve visibility of scanned shape, you may need to adjust one of visualized object options, set value of Style to Points. You should see many points which resemble shape of obstacles surrounding your robot.



Shut down rplidarNode and run it again, but with some other nodes:

CORE2 bridge node - roslaunch rosbot\_ekf rosserial\_bridge.launch

Rosbot PRO - roslaunch rosbot\_ekf rosserial\_bridge.launch serial\_port:=/dev/ttyS4 serial\_baudrate:=460800

rplidarNode - driver for rpLidar laser scanner

drive\_controller\_node - publisher that you just created

Or instead of these three, start Gazebo:

roslaunch rosbot\_gazebo maze\_world.launch

You will also need:

static\_transform\_publisher - tf publisher for transformation of laser scanner relative to robot

teleop\_twist\_keyboard - keyboard controller

rviz - visualization tool

You can use below launch file:

<launch>

<arg name="use\_rosbot" default="true"/>

<arg name="use\_gazebo" default="false"/>

<include if="$(arg use\_gazebo)" file="$(find rosbot\_gazebo)/launch/maze\_world.launch"/>

<include if="$(arg use\_gazebo)" file="$(find rosbot\_gazebo)/launch/rosbot.launch"/>

<include file="$(find rplidar\_ros)/launch/rplidar.launch"/> <!-- Rosbot 2.0-->

<!-- <include file="$(find rplidar\_ros)/launch/rplidar\_a3.launch"/> --> <!-- Rosbot PRO-->

<node if="$(arg use\_rosbot)" pkg="tutorial\_pkg" type="drive\_controller\_node" name="drive\_controller"/>

<!-- ROSbot 2.0 -->

<include if="$(arg rosbot)" file="$(find rosbot\_ekf)/launch/rosserial\_bridge.launch" />

<!-- ROSbot 2.0 PRO -->

<!--

<include file="$(find rosbot\_ekf)/launch/rosserial\_bridge.launch">

<arg name="serial\_port" value="/dev/ttyS4"/>

<arg name="serial\_baudrate" value="460800"/>

</include>

-->

<node if="$(arg use\_rosbot)" pkg="tf" type="static\_transform\_publisher" name="laser\_broadcaster" args="0 0 0 3.14 0 0 base\_link laser 100" />

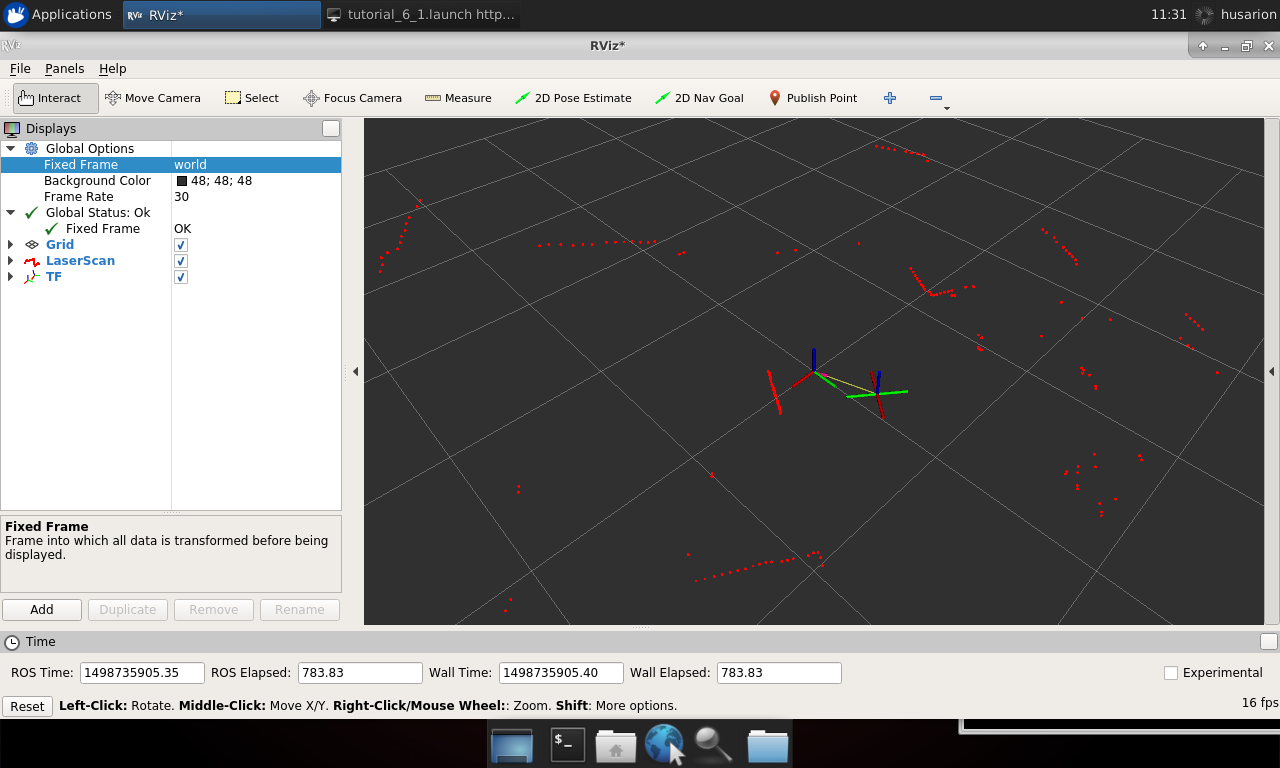
<node pkg="teleop\_twist\_keyboard" type="teleop\_twist\_keyboard.py" name="teleop\_twist\_keyboard" output="screen"/>

<node pkg="rviz" type="rviz" name="rviz"/>

</launch>

In rviz add Tf and /scan. This time set Fixed Frame to odom.

Try to move around your robot, you should see as laser scans change its shape accordingly to obstacles passed by robot.



Navigation and map building

For building a map and localizing robot relative to it, we will use slam\_gmapping node from gmapping package.

This node subscribes /tf topic to obtain robot pose relative to starting point and laser scanner pose relative to robot and also subscribe /scan topic to obtain laser scanner messages. Node publishes to /map topic with message type nav\_msgs/OccupancyGrid, this contain the actual map data.

We need to set few parameters:

base\_frame - name of frame related with robot, in our case it will be base\_link

odom\_frame - name of frame related with starting point, in our case it will be odom

delta - map resolution expressed as size of every pixel in meters

You can use below launch file:

<launch>

<arg name="use\_rosbot" default="true"/>

<arg name="rosbot\_pro" default="false"/>

<arg name="use\_gazebo" default="false"/>

<include if="$(arg use\_gazebo)" file="$(find rosbot\_gazebo)/launch/maze\_world.launch"/>

<include if="$(arg use\_gazebo)" file="$(find rosbot\_gazebo)/launch/rosbot.launch"/>

<include if="$(arg use\_rosbot)" file="$(find rplidar\_ros)/launch/rplidar.launch"/>

<include if="$(arg rosbot\_pro)" file="$(find rplidar\_ros)/launch/rplidar\_a3.launch"/>

<node if="$(arg use\_rosbot)" pkg="tutorial\_pkg" type="drive\_controller\_node" name="drive\_controller"/>

<include if="$(arg use\_rosbot)" file="$(find rosbot\_ekf)/launch/rosserial\_bridge.launch"/>

<include if="$(arg rosbot\_pro)" file="$(find rosbot\_ekf)/launch/rosserial\_bridge.launch">

<arg name="serial\_port" value="/dev/ttyS4"/>

<arg name="serial\_baudrate" value="460800"/>

</include>

<node pkg="tf" type="static\_transform\_publisher" name="laser\_broadcaster" args="0 0 0 3.14 0 0 base\_link laser 100" />

<node pkg="teleop\_twist\_keyboard" type="teleop\_twist\_keyboard.py" name="teleop\_twist\_keyboard" output="screen"/>

<node pkg="rviz" type="rviz" name="rviz"/>

<node pkg="gmapping" type="slam\_gmapping" name="gmapping">

<param name="base\_frame" value="base\_link"/>

<param name="odom\_frame" value="odom" />

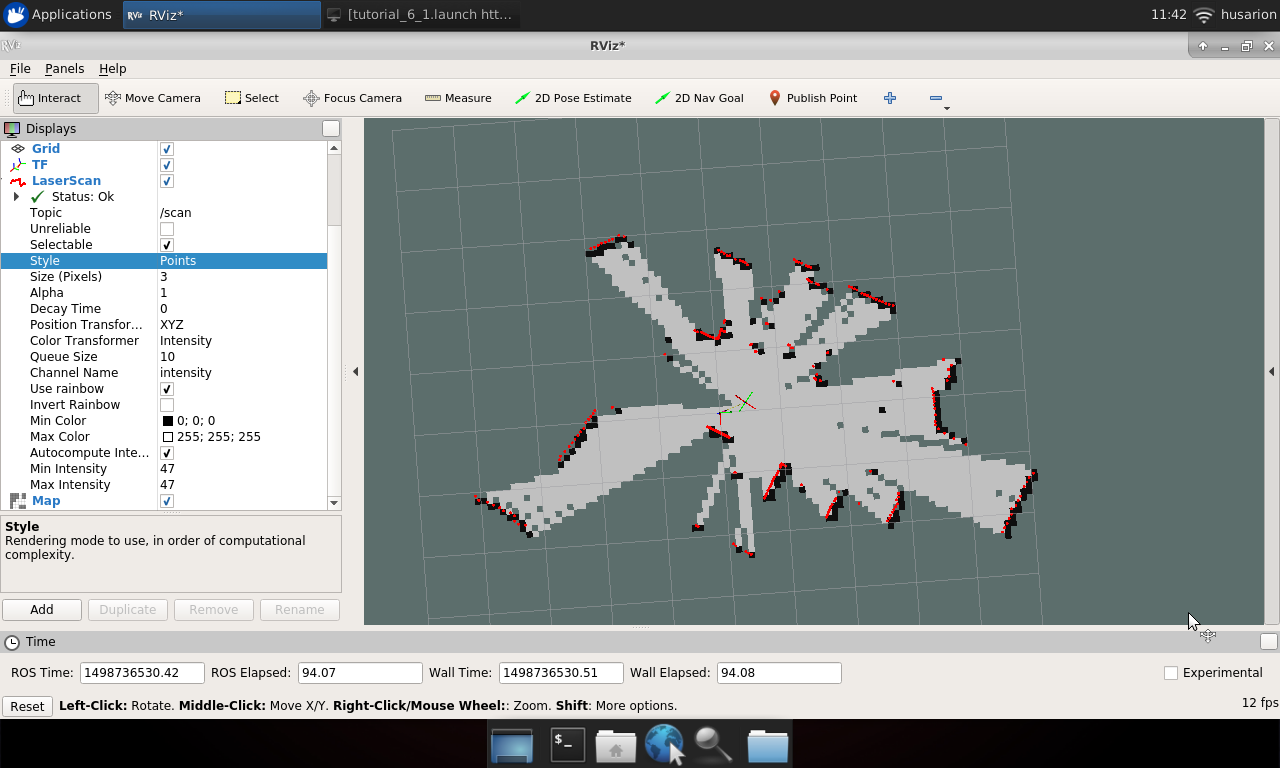
<param name="delta" value="0.1" />

</node>

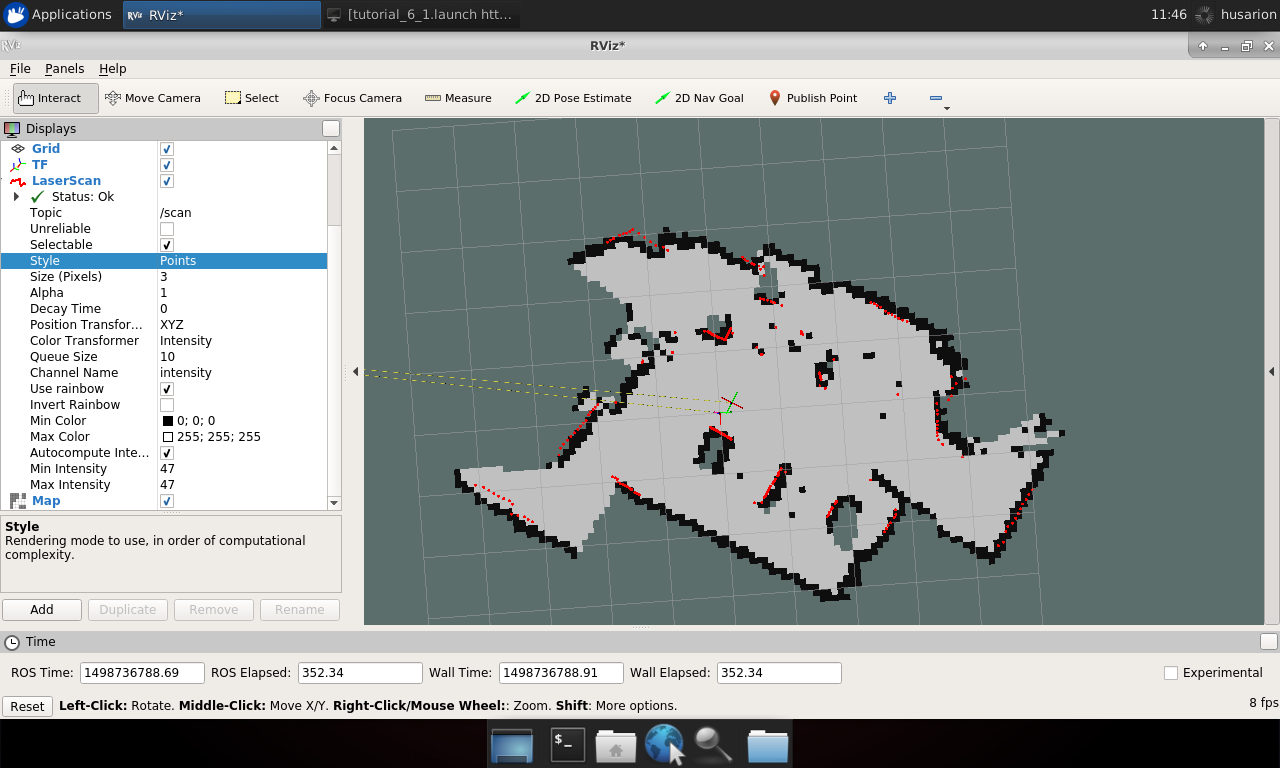
</launch>

In rviz add Tf and /scan, again open object adding window, select By topic and from the list select /map.

At the beginning there could be no map, you may need to wait few second until it is generated. Starting state should be similar to the one on picture:



Now drive your robot around and observe as new parts of map are added, continue until all places are explored. Final map should look like below:



Summary

After completing this tutorial you should be able to publish transformations between various frames, connect laser scanner to the system, set up slam\_gmapping node to perform SLAM task and visualize map, robot position and all related frames.

From: <https://docs.leorover.tech/integrations/lidar-sensor> URDF example

Mounting and wiring the sensor

When mounting the sensor, you should be particularly careful not to obstruct the field of view by other parts of the Rover.

We developed 3D printable models of parts that allow mounting the aforementioned sensors to the mounting plate located at the top of the robot. The files are listed here:

3D-printed parts

/documentation/3d-printed-parts

The sensor can be connected to the robot's main computer via the mounting plate USB socket.

As for the power supply, you will have to provide additional connection from robot's battery to the sensor through a 5V DC converter (if the sensor is not powered via the USB socket).

The mounted sensor should look similar to this:

RPLIDAR



Integrating the sensor with the system

The first thing you can do is to make sure your device has the correct permissions and is available at the fixed path on your system. To do this, you can add the following rule to theudev service:

RPLIDAR

/etc/udev/rules.d/lidar.rules

KERNEL=="ttyUSB\*", ATTRS{idVendor}=="10c4", ATTRS{idProduct}=="ea60", MODE="0666", GROUP="dialout", SYMLINK+="lidar"

Paste these lines to /etc/udev/rules.d/lidar.rules file and reload udev rules by typing:

sudo udevadm control --reload-rules && sudo udevadm trigger

Your device should now be available at the /dev/lidar path.

We want the sensor functionality to be available in the ROS ecosystem, so you should install a ROS package that provides a node for the sensor you are trying to integrate.

RPLIDAR

sudo apt install ros-melodic-rplidar-ros

Now, create a launch file that would start the node with a fitting configuration.

RPLIDAR

/etc/ros/laser.launch

<launch>

<node name="rplidar\_node" pkg="rplidar\_ros" type="rplidarNode" output="screen">

<param name="serial\_port" value="/dev/lidar"/>

<param name="frame\_id" value="laser\_frame"/>

</node>

</launch>

Include your launch file in the robot.launch file, so that your node will start at boot.

/etc/ros/robot.launch

<include file="/etc/ros/laser.launch"/>

Your robot should be aware of where the sensor is located and what space it occupies. You can ensure it does that by creating a URDF model of the sensor.

RPLIDAR

/etc/ros/urdf/laser.urdf.xacro

<?xml version="1.0"?>

<robot>

​

<link name="rplidar\_link">

<visual>

<origin xyz="0 0 0.003"/>

<geometry>

<box size="0.079 0.086 0.006"/>

</geometry>

<material name="support">

<color rgba="0.5 0.5 0.5 1.0"/>

</material>

</visual>

<visual>

<origin xyz="0 0 0.023"/>

<geometry>

<cylinder radius="0.038" length="0.04"/>

</geometry>

<material name="lidar">

<color rgba="1.0 0.0 0.0 0.7"/>

</material>

</visual>

<collision>

<origin xyz="0 0 0.003"/>

<geometry>

<box size="0.079 0.086 0.006"/>

</geometry>

</collision>

<collision>

<origin xyz="0 0 0.023"/>

<geometry>

<cylinder radius="0.038" length="0.04"/>

</geometry>

</collision>

</link>

​

<joint name="rplidar\_joint" type="fixed">

<origin xyz="0.0775 0 0"/>

<parent link="base\_link"/>

<child link="rplidar\_link"/>

</joint>

​

<link name="laser\_frame"/>

​

<joint name="laser\_joint" type="fixed">

<origin xyz="0 0 0.0368" rpy="0 0 ${pi}"/>

<parent link="rplidar\_link"/>

<child link="laser\_frame"/>

</joint>

​

</robot>

and including it in the description that is uploaded at boot.

/etc/ros/urdf/robot.urdf.xacro

<xacro:include filename="/etc/ros/urdf/laser.urdf.xacro"/>

You can experiment with the URDF file and create a more representative model of the sensor by adding more visual and collision tags or by including meshes in STL or COLLADA format.

The last step is to either reboot the robot or restart the leo service.

sudo systemctl restart leo

Reading and visualizing the data

The robot should now publish the LaserScan messages on the /scan topic. You can check the raw data that it sends by typing:

rostopic echo /scan

If you have ROS installed on your computer, you can get a more graphical representation of the data with RViz. If you don't have ROS, you can follow this guide:

Install ROS on your computer

/development-tutorials/ros-development/install-ros-on-your-computer

Before starting RViz, make sure you completed the Connecting other computer to ROS network section of ROS Development tutorial:

ROS Development

/development-tutorials/ros-development

Now, open RViz by typing rviz in the terminal, or, if you have the leo\_viz package installed, type:

roslaunch leo\_viz rviz.launch

This will start RViz with visualization of the current robot model.

On the Displays panel click Add -> By topic and search for the /scan topic. Choose the LaserScan display and click Ok. You should now be able to see the data from the sensor visualized as points in 3D space.

To put the points into the camera image, you can also add theCamera display (be sure to check compressed as the Transport Hint). Here's an example end result: