Lessons for robot tele-control, mapping and navigation in ROS

These lessons are designed for high school students and college students. They cover material for working with the Robot Operating System - ROS.

All modern electronic and mechatronic devices are controlled by computer programs - software. Robots are complex machines that usually require more special and complex software solutions. Therefore, the robot operating system-ROS was created. Its goal is to provide ready-made software libraries and solutions for easy robot programming.

- **What is ROS?**

The Robot Operating System - ROS is designed to control robots, drives, machines and more. The main components in ROS are packages, nodes, topics, services and messages. ROS is a meta-operating system and is installed on other operating systems such as Linux (recommended) or Windows.
Software in ROS is organized in packages. A package might contain ROS nodes, a ROS-independent library, a dataset, configuration files, a third-party piece of software, or anything else that logically constitutes a useful module. The goal of these packages it to provide this useful functionality in an easy-to-consume manner so that software can be easily reused. In general, ROS packages follow a "Goldilocks" principle: enough functionality to be useful, but not too much that the package is heavyweight and difficult to use from other software.
A node is a process that performs computation. Nodes are combined together into a graph and communicate with one another using streaming topics, RPC services, and the Parameter Server. These nodes are meant to operate at a fine-grained scale; a robot control system will usually comprise many nodes. For example, one node controls a laser range-finder, one Node controls the robot's wheel motors, one node performs localization, one node performs path planning, one node provides a graphical view of the system, and so on.
ROS services are designed to provide two-way communication between nodes, based on the principle of request / response. They are used to automate some processes during robot control.
Nodes communicate with each other by publishing messages to topics.
A message is a simple data structure, comprising typed fields. Standard primitive types (integer, floating point, boolean, etc.) are supported, as are arrays of primitive types. Messages can include arbitrarily nested structures and arrays (much like C structs). Nodes can also exchange a request and response message as part of a ROS service call. These request and response messages are defined in srv files.

- **Ubuntu terminal and ROS commands.**

Because Ubuntu Linux is the recommended operating system for working with ROS, we use it in these tutorials. To work with ROS we must use the Linux terminal shown in

Figure 1. The terminal in Linux is a basic tool for accessing, processing files, working with devices, installing / uninstalling programs, running scripts and applications, and more. In the terminal through special commands, we can start programs in ROS, to check the active nodes, topics, services and to visualize messages with data from different topics.

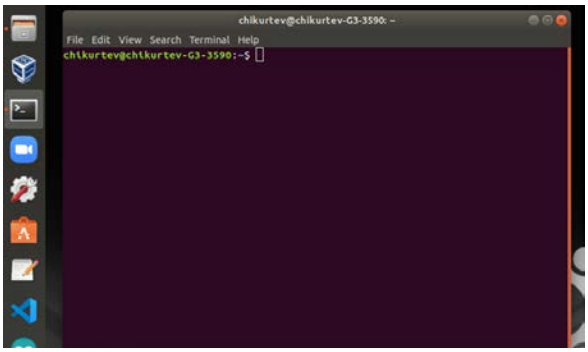

Figure 1. Linux terminal.

Some of the most used commands are:

rosrun package_name node_name – start a ROS node

roslaunch package_name launch_file_name – start a launch file

rosnode list – list all active nodes

rosnode info /node_name – show information about a node

rostopic list – list all active topics

rostopic info /topic_name – show information about a topic

rostopic echo /topic_name – show the data transmitted in a topic

rostopic pub /topic name massege_type „messege data“ - publish a message in a topic

rosservice list – list all available services

rosservice call /service_name param – call a service and parameter (some services don’t require parameters)

If we want to run a node from a given package in ROS, as an example we can use the following command:

rosrun turtlesim turtlesim_node

We can see that the name of the used package is **turtlesim** and the name of the started node is **turtlesim_node**. In this way, we can run any node, located in any package. However, this method runs only a single node and it is not suitable and comfortable when we have to work with several nodes.

ROS provides a method for running multiple node by using files of type ‘launch’. When we want to run that king of files, we have to use the ‘roslaunch’ command. An example for usage of this command is:

roslaunch turtlebot3_gazebo turtlebot3_empty_world.launch

Here, like the way we run node, first is the name of the package, where is located the file, then is the name of the file and its extension '.launch'. This file type is often used because it allows you to run multiple programs and set specific parameters for each program at the same time.

In addition to commands for starting nodes, commands for providing information are also very useful. These commands help the developer to navigate the full picture and check which programs are running, which topics are active and additional information about them. For example, the **rostopic list** command shows all started topics. **rostopic info /cmd_vel** command will show all the information about the topic like: nodes, which are publishing the topic, subscribed nodes, massage type and others. By this method, we can gather information about every active topic and to track its connections. This method is used also for nodes and services.
The third type of commands is used for direct interaction between user and ROS nodes. These methods give the ability to visualize the transmitted data into the topics or to send data in topic, and to call a service. An example for publishing data in topic is as follows:
rostopic pub -r 10 /cmd_vel geometry_msgs/Twist '{linear: {x: 0.1, y: 0.0, z: 0.0}, angular: {x: 0.0,y: 0.0,z: 0.0}}'
In this example, we are publishing a message in the topic **cmd_vel**, message type is geometry_msgs/Twist and message parameters are two vectors linear and angular. Each vector consists data for robot movement. Other important parameter for publishing data in topic is **-r 10**. This parameter activates the Rate mode – continuous publishing if the message with given frequency (the digit sets the frequency). If we do not use that parameter, then the active mode is Latching mode (the default mode). In this mode the message is published once and then the process stops.

We have already been acquainted with the main components for working with ROS and we can move on to work with robots and some of the popular ROS packages.

- **ROS tools and applications**

As part of the ROS, it is often necessary to use some specialized tools and applications. The most commonly used ones are:

- **Gazebo**: application for robot simulation.
  This application specializes in simulating robots and creating simulated environments. Through it, we can test and experiment with virtual robot models those we want to control. We can also create our own virtual robot.
- **Rviz**: real-time data visualization tool.
  Rviz is a very useful tool that visualizes data from all topics in ROS. With this tool, we can monitor what data is received from different sensors, compare and analyze them. Here is also visualized the model of the robot we use, the map made if any, the trajectory traveled and others.

- Tool for visualization of ROS nodes, topics and services and the connections between them - **rqt_graph**.
  With the help of this tool, we can easily navigate in the current state of all active nodes, topics and more. The tool shows the relationships between the nodes, and gives a clear idea of the principle of operation of a control system. This tool can detect irregularities, improve algorithms and track the operation of a system.

- **Tele-Control of Mobile Robot - Control of a simulated mobile robot via keyboard and/or joystick.**

Tele-control is a method of controlling a robot directly by an operator. This type of control is usually used for initial tests of a robot, for learning and memorizing movements, and for mapping. The process of tele-control is expressed in direct commands to the robot. For example, we send a command to a mobile robot to move forward at a certain speed and this continues until we issue a stop command. In this way, simulated robots can be controlled in a simulation environment.
In ROS, it is accepted to use a standard topic called **cmd_vel** to drive a mobile robot. In order to control a mobile robot, we need to have a program that publishes data in this topic. There are various ways to tele-control a robot: through keyboard commands and through joystick commands.
The program **teleop_twist_keyboard**, provides teleoperation functions by using the keyboard. This program reads the keys on the keyboard and sends data to the **cmd_vel** topic according to preset parameters.
The joystick control program works in the same way. It is named **teleop_twist_joy**. This program reads the data from the joystick buttons and converts it into robot control data.
In this tutorial, we will use the simulation of the robot turtlebot3, in a similar way the simulations of the robots husky and turtlebot2 can be used.
The robot simulation in a simulation environment must be started initially. This is done by executing the following command in the terminal:

```
export TURTLEBOT3_MODEL=burger
roslaunch turtlebot3_gazebo turtlebot3_world.launch
```

After executing the given command, the simulated world of the robot shown in Figure 2 is loaded. Gazebo application and the simulated models of the robot and the world around it can be seen in the figure.

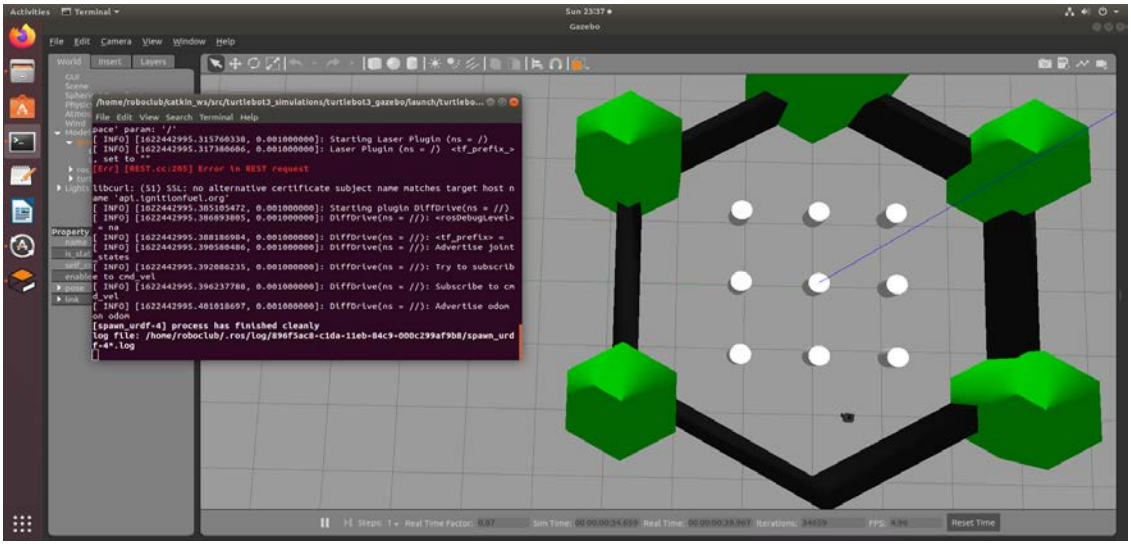

Figure 2. Gazebo application and the simulated models of the robot and the world around it.

When we have a robot running, we can now start one of the tele-control programs. In this tutorial we will look at remote control with a keyboard, because it does not require a joystick. To run the keyboard control program in a separate terminal, you must run the following command:

```
export TURTLEBOT3_MODEL=burger
roslaunch turtlebot3_teleop turtlebot3_teleop_key.launch
```

When the command is executed, instructions for working with the keyboard keys are displayed in the terminal (see Figure 3). As you can see in the figure, the keys used are: w, a, s, d, x.

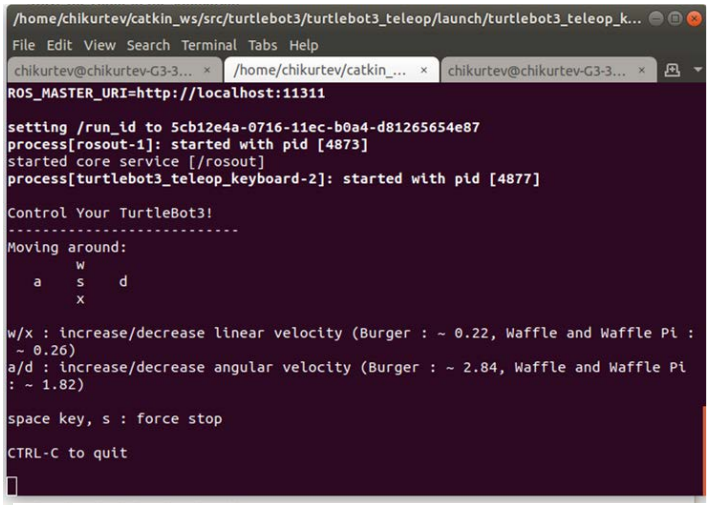

Figure 3. Terminal for tele-control f Turtlebot3.

From the explanations in the terminal, it is clear that one keystroke „w x, a, d” increases or decreases the speed of the robot. Therefore, once we start pressing the specified keys, the program sends commands to the robot and it starts moving. Therefore, we can now control the robot and move it wherever we want. Robot movement can be seen in the Gazebo simulation.

- **SLAM Map Building by tele controlling the simulated robot in the simulated environment.**

As mentioned above, tele-control can be used for mapping. Mapping is a method of creating a map of a room or environment, using various software tools and algorithms. In mobile robots, mapping serves as the basis of the Navigation and Localization systems. By circling the space and using sensors to measure distance (laser, infrared, ultrasonic, etc.), the robot can automatically create a map.

In this tutorial, we will learn about the process of creating a map using ROS packages for localization and mapping. As in the tutorial above, we will use the turtlebot3 robot simulation. After we have executed the commands and steps from the previous lesson (the simulation of turtlebot3 is loaded and the tele-control program is started), now the following command should be started in a new terminal:

```
export TURTLEBOT3_MODEL=burger
roslaunch turtlebot3_slam turtlebot3_slam.launch slam_methods:=gmapping
```

This command opens a window of the Rviz tool, where the data from all topics are displayed, including the created map. Now we have to use the tele-control method to go around the whole space around the robot until a complete map with good details is formed, as in Figure 4.

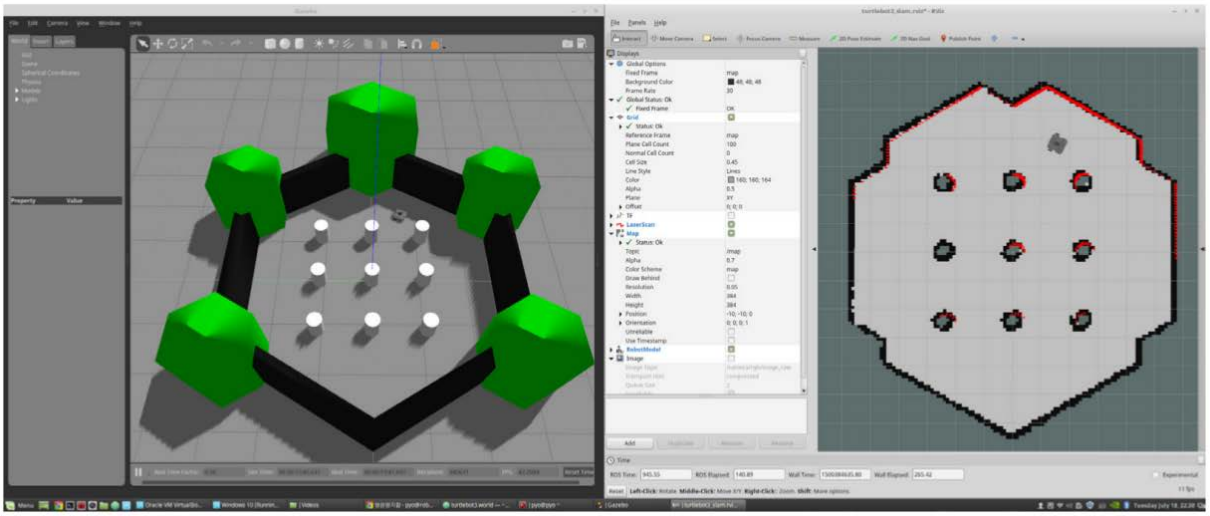

Figure 4. Gazebo and Rviz, created map.

When we have a well-made map, the last step of the mapping process remains - saving the map. To perform this step we need to open a new terminal and execute the following command:

```
rosrun map_server map_saver -f ~/map
```

This command saves the data of the map in two files in the “HOME” directory. We can easily specify the location where to save these files, as well as the file names. This is done by entering the desired location after the ‘~’ sign, and the words after the last slash are taken as the name of the map.

Thus, once we have a map, we can use it to control the robot by setting a desired destination and using the ROS navigation package.

- **Control the simulated robot to perform Autonomous Navigation of a Known Map to a user selected point on the map**

Autonomous navigation is the process of moving a mobile robot independently from one place to another. During movement, the system detects the presence of obstacles, calculates the current location of the robot and determines the shortest path to the set destination. In this tutorial, we use the ROS package for autonomous navigation.

Now, just like in the mapping lesson, we have to start the simulation of the turtlebot3 robot, and it is not necessary to start the tele-control program (the navigation system will control the robot itself). The next step is to start the navigation system programs with the following command in a new terminal:

```
export TURTLEBOT3_MODEL=burger
roslaunch turtlebot3_navigation
turtlebot3_navigation.launch map_file:=$HOME/map.yaml
```

This command, in addition to starting the navigation packs, tells the navigation system which map to use. In our example, we saved a map named ‘map’ located in the ‘HOME’ directory, and now we tell the navigation system to use that map.

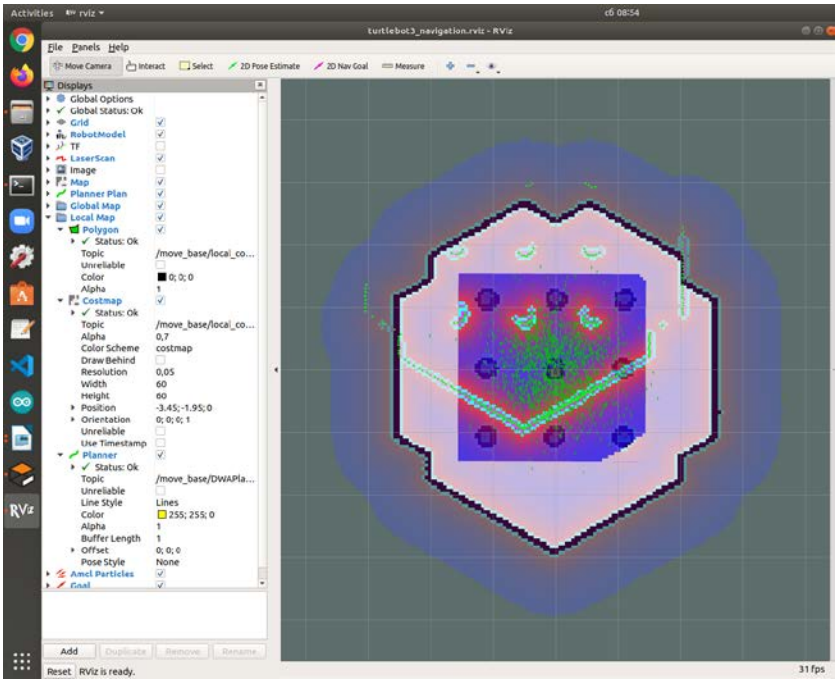

Figure 5. Starting the navigation packages.

After executing the above command, a window opens in Rviz, where the specified map, the location of the robot and the data from the sensors are displayed (Figure 5). As can be seen from the figure, the data from the Lidar (in red) do not match the outline of the map, but there is a shift. This is due to the difference between the initial position of the robot and the saved one when creating the map.

If we want normal operation of the navigation system, we have to set the true initial localization of the robot in the map. This is done by using the '2D Pose Estimate' button:

1. Click the 2D Pose Estimate button in the RViz menu (figure 6).
2. Click on the map where the actual robot is located and drag the large green arrow toward the direction where the robot is facing.
3. Repeat step 1 and 2 until the LDS sensor data is overlayed on the saved map.

When we successfully set the correct location of the robot, then the outlines of the sensors and those on the map must match exactly. Figure 7 shows an example with a correctly determined initial location. However, spots can be seen on the map that have been marked from the previous location of the robot.

Figure 5. Rviz button menu.

To remove old data fingerprints we need to run the following command in a new terminal:

```
rosservice call /move_base/clear_costmaps
```

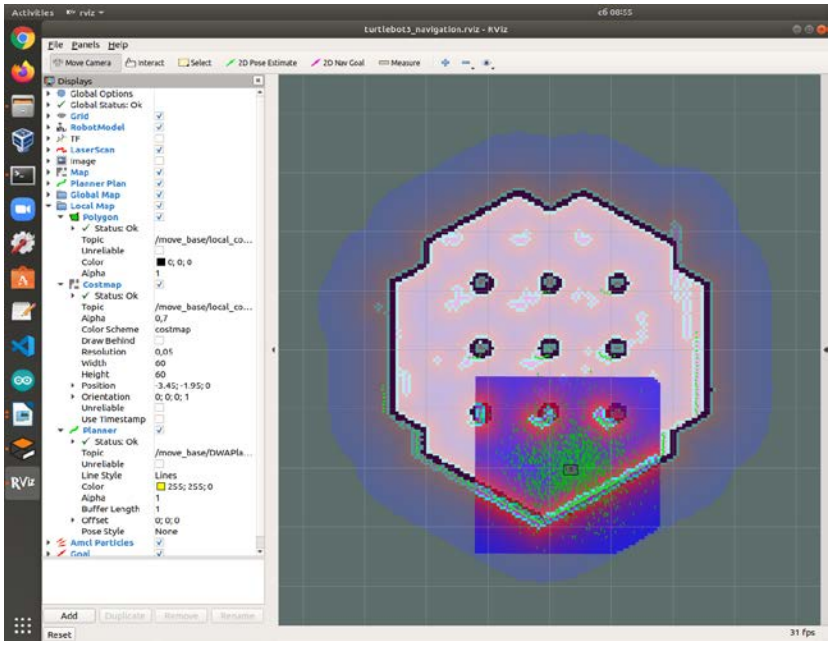

Figure 7. Successfully set initial position.

This command clears all old data from the map and after executing it, we can start setting desired positions where the robot should go. Then set the desired destination to reach.

- Set Navigation Goal

1. Click the 2D Nav Goal button in the RViz menu (figure 8).

Figure 8. Rviz button menu.

2. Click on the map to set the destination of the robot and drag the green arrow toward the direction where the robot will be facing (figure 9).

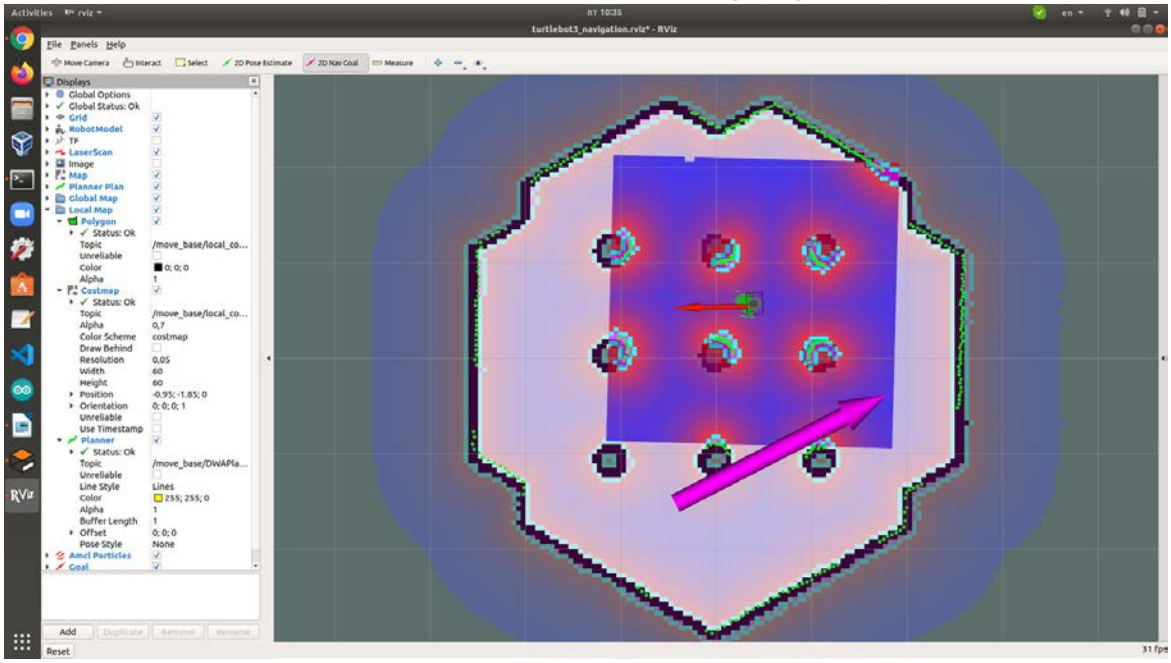

Figure 9. Setting goal for the navigation.

Wait until the robot reaches the desired destination and orientation. When the robot successfully completes the task in the terminal where the navigation node is started, the message "Goal reached" is displayed. After writing this message we can set the next destination and orientation. On the map we can see the location and orientation of the robot, the generated path to the target and the trajectory of the robot.

An example for Turtlebot3 Navigation is available in: https://www.youtube.com/watch?v=VYlMywwYALU&ab_channel=ROBOTISOpenSourceTeam