Chapter 10

Mobile Robots

10.1. Robot Supported by ROS

Robots supported by ROS can be found on the Wiki page (http://robots.ros.org/). About 180 robots are developed based on ROS. Some of these include custom robots that are publicly released by developers, and it is a noticeable list considering a single system supports such diverse robots. The most well-known robots among these are the PR2 developed by Willow Garage and TurtleBot. Both robots are ROS standard platforms that Willow Garage or Open Robotics (formerly OSRF) took part in development. In this chapter, we will focus on TurtleBot which was developed as an entry level platform.

10.2. TurtleBot3 Series

TurtleBot is a ROS standard platform robot. Turtle is derived from the Turtle robot, which was driven by the educational computer programming language 'Logo'[1] in 1967. In addition, the turtlesim node, which first appears in the basic tutorial of ROS, is a program that mimics the command system of the Logo turtle[2] program. It is also used to create the Turtle icon as a symbol of ROS as shown in Figure 10-1. The nine dots used in the ROS logo derived from the back shell of the turtle. TurtleBot, which originated from the Turtle of Logo, is designed to easily teach people who are new to ROS through TurtleBot as well as to teach computer programming language using Logo. Since then TurtleBot has become the standard platform of ROS, which is the most popular platform among developers and students.

Figure 10-1 Symbols of each ROS version

There are 3 versions of the TurtleBot series[3] (see Figure 10-2). TurtleBot1 was developed by Tully (Platform Manager at Open Robotics) and Melonee (CEO of Fetch Robotics) from Willow Garage on top of the iRobot's Roomba-based research robot, Create, for ROS deployment. It was developed in 2010[4] and has been on sale since 2011. In 2012, TurtleBot2 was developed by Yujin Robot based on the research robot, iClebo Kobuki. In 2017, TurtleBot3 was developed with features to supplement the lacking functions of its predecessors, and the demands of users. The TurtleBot3 adopts ROBOTIS smart actuator 'Dynamixel' for driving.

1 http://el.media.mit.edu/logo-foundation/index.html
2 http://el.media.mit.edu/logo-foundation/what_is_logo/logo_primer.html
3 http://www.turtlebot.com/about
4 http://spectrum.ieee.org/automaton/robotics/diy/interview-turtlebot-inventors-tell-us-everything-about-the-robot

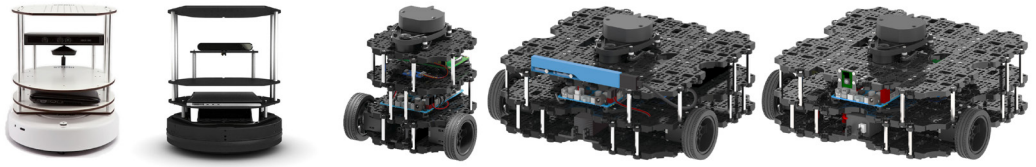

FIGURE 10-2 From left TurtleBot1, TurtleBot2, TurtleBot3 (last three models)

TurtleBot3 is a small, affordable, programmable, ROS-based mobile robot for use in education, research, hobby, and product prototyping. The goal of TurtleBot3 is to dramatically reduce the size of the platform and lower the price without having to sacrifice its functionality and quality, while at the same time offering expandability. The TurtleBot3 can be customized into various ways depending on how you reconstruct the mechanical parts and use optional parts such as the computer and sensor. In addition, TurtleBot3 is evolved with cost-effective and small-sized SBC that is suitable for robust embedded system, 360 degree distance sensor and 3D printing technology.

10.3. TurtleBot3 Hardware

There are three official TurtleBot3[5] models, TurtleBot3 Burger, Waffle and Waffle Pi as shown in Figure 10-3, and this book will mainly discuss based on TurtleBot3 Burger unless otherwise mentioned. In addition, TurtleBot3 supports various structures and hardware such as Monster, Tank, Carrier and these variations are named as TurtleBot3 + [Suffix]. The basic components of TurtleBot3 are actuators, an SBC for operating ROS, a sensor for SLAM and navigation, restructurable mechanism, an OpenCR embedded board used as a sub-controller, sprocket wheels that can be used with tire and caterpillar, and a 3 cell lithium-poly battery. TurtleBot3 Waffle is different from Burger in terms of platform shape which can conveniently mount components, use of higher torque actuators, high-performance SBC with Intel processor, RealSense Depth Camera for 3D recognition from Intel. TurtleBot3 Waffle Pi is the same shape as the Waffle model, but this model is used the Raspberry Pi as the Burger model, and the Raspberry Pi Camera to make it more affordable.

5 http://turtlebot3.robotis.com

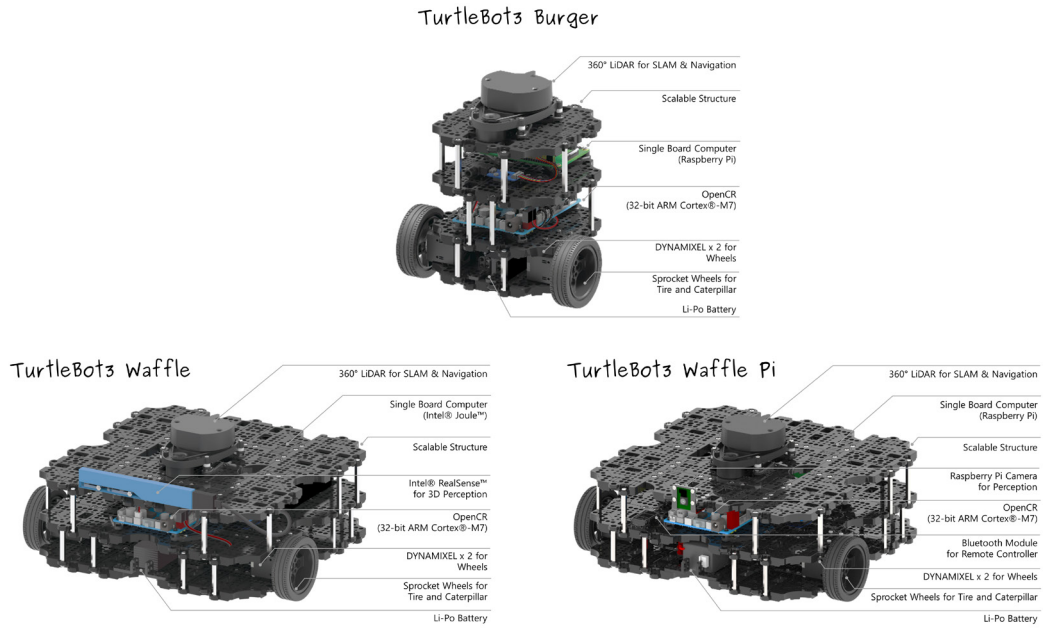

FIGURE 10-3 Hardware configuration of TurtleBot3

The 3D CAD design files of TurtleBot3 are available through cloud-based 3D CAD tool 'Onshape', and allows all design team members to access the shared design files using smartphones, tablets as well as PC, regardless of operating system. Not only you can check each component of TurtleBot3 using a web browser, but also you can download parts to your repository and make your own parts by modifying the design. After downloading the STL file, parts can be printed using a 3D printer. The files for each model can be found in the Open Source item provided as an appendix at the TurtleBot3 official Wiki[6].

6 http://turtlebot3.robotis.com

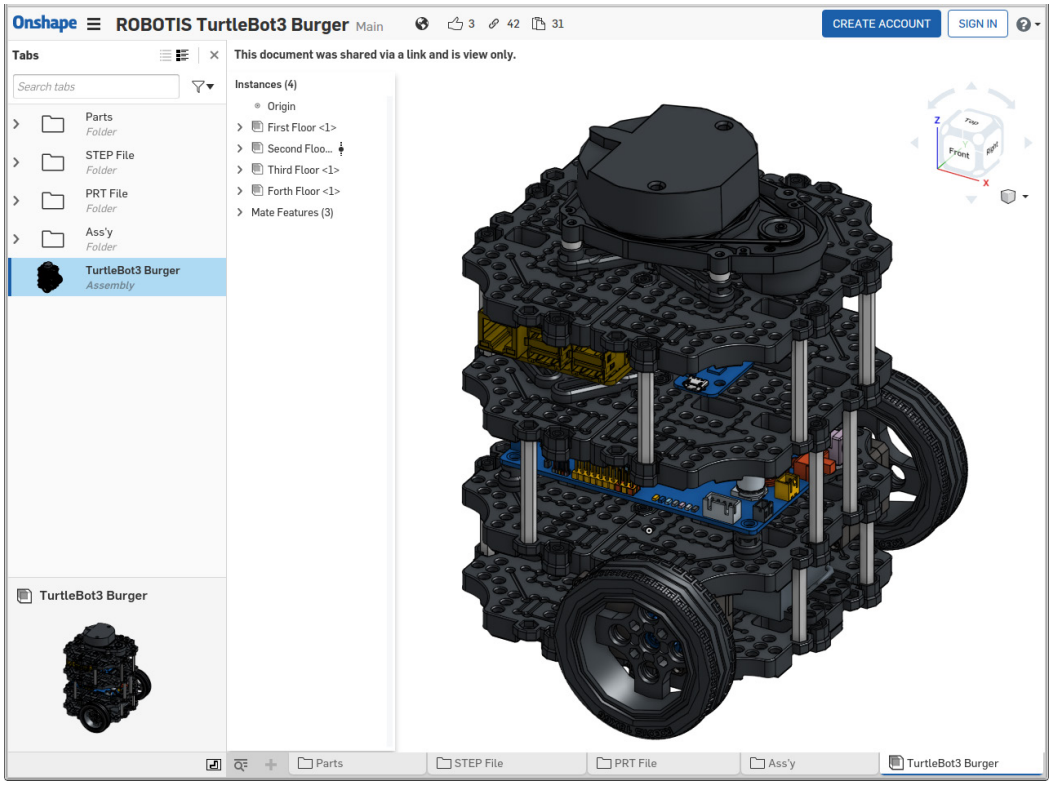

FIGURE 10-4 Open source hardware of TurtleBot3

Official TurtleBot3 Wiki

The aforementioned hardware details of TurtleBot3 and basic contents described in this chapter can also be found in the official TurtleBot3 wiki in following link. To learn ROS using TurtleBot3, refer to the link below.

http://turtlebot3.robotis.com

Open Source Hardware for TurtleBot3

The hardware design files and software of the TurtleBot3 are open to the public. If you need the hardware files for each TurtleBot3 model and for OpenCR used as the sub-controller of the TurtleBot3, refer to the list and links below. Unless otherwise stated, all hardware are open source and comply with the Hardware Statement of Principles and Definition v1.0 license.

OpenCR: https://github.com/ROBOTIS-GIT/OpenCR-Hardware

TurtleBot3 Burger:	http://www.robotis.com/service/download.php?no=676
TurtleBot3 Waffle:	http://www.robotis.com/service/download.php?no=677
TurtleBot3 Waffle Pi:	http://www.robotis.com/service/download.php?no=678
TurtleBot3 Friends OpenManipulator Chain:	http://www.robotis.com/service/download.php?no=679
TurtleBot3 Friends Segway:	http://www.robotis.com/service/download.php?no=680
TurtleBot3 Friends Conveyor:	http://www.robotis.com/service/download.php?no=681
TurtleBot3 Friends Monster:	http://www.robotis.com/service/download.php?no=682
TurtleBot3 Friends Tank:	http://www.robotis.com/service/download.php?no=683
TurtleBot3 Friends Omni:	http://www.robotis.com/service/download.php?no=684
TurtleBot3 Friends Mecanum:	http://www.robotis.com/service/download.php?no=685
TurtleBot3 Friends Bike:	http://www.robotis.com/service/download.php?no=686
TurtleBot3 Friends Road Train:	http://www.robotis.com/service/download.php?no=687
TurtleBot3 Friends Real TurtleBot:	http://www.robotis.com/service/download.php?no=688
TurtleBot3 Friends Carrier:	http://www.robotis.com/service/download.php?no=689

10.4. TurtleBot3 Software

The TurtleBot3 software consists of firmware (FW) of OpenCR board used as a sub-controller and 4 ROS packages. The firmware of TurtleBot3 is also called as 'turtlebot3_core' in the sense that it is the core of TurtleBot3 which was already described in Chapter 9 Embedded System. It uses OpenCR as a sub-controller to estimate the location of the robot by calculating the encoder value of Dynamixel, which is the driving motor of TurtleBot3 or to control the velocity according to the command published by the upper-level software. In addition, acceleration and angular velocity are obtained from 3-axis acceleration and 3-axis gyro sensor mounted on OpenCR to estimate the direction of the robot, and the battery state is also measured and transmitted via topics.

TurtleBot3's ROS package includes 4 packages which are 'turtlebot3', 'turtlebot3_msgs', 'turtlebot3_simulations', and 'turtlebot3_applications'. The 'turtleBot3' package contains TurtleBot3's robot model, SLAM and navigation package, remote control package, and bringup package. The 'turtlebot3_msgs' package contains message files used in turtlebot3, 'turtlebot3_simulations' contains packages related to simulation, and 'turtlebot3_applications' package contains applications.

Open source software for TurtleBot3

The software of TurtleBot3 is disclosed to public as an open source. The OpenCR bootloader used as a sub-controller of TurtleBot3, the firmware for developing with Arduino IDE, and the firmware for controlling TurtleBot3, as well as the ROS packages (turtlebot3, turtlebot3_msgs, turtlebot3_simulations, turtlebot3_applications) are also available as an open source. Licenses for open source software vary for each source but basically comply with Apache license 2.0, and some software uses 3-Clause BSD License and GPLv3.

https://github.com/ROBOTIS-GIT/OpenCR

https://github.com/ROBOTIS-GIT/turtlebot3

https://github.com/ROBOTIS-GIT/turtlebot3_msgs

https://github.com/ROBOTIS-GIT/turtlebot3_simulations

https://github.com/ROBOTIS-GIT/turtlebot3_applications

10.5. TurtleBot3 Development Environment

The development environment of TurtleBot3 can be divided into Remote PC that performs remote control, SLAM, Navigation package, and TurtleBot PC that controls the robot components and collects sensor information as shown in Figure 10-5. Both PCs are very similar in terms of its development environment, but the packages they use are configured differently depending on the performance and purpose of the PC. The basic development environment for both PC requires Linux (here Ubuntu 16.04 and compatible Linux mint and Ubuntu MATE) as the base operating system, and ROS (Kinetic Kame). Refer to Chapter 3 for configuring ROS Development Environment in detail. Also, refer to the following website for setting up PC, TurtleBot and OpenCR.

- http://turtlebot3.robotis.com

If you have Linux and ROS installed, you can install the software associated with TurtleBot3. All of these installation methods are described in the above-mentioned TurtleBot3 wiki, but here we summarize them briefly and explain just the installation method. Let's install dependent packages and TurtleBot3 packages on the operating PC (this PC will be referred as the Remote PC) that controls the TurtleBot3. However, we have excluded the 'turtlebot3_applications' package which contains various examples not mentioned in this book.

Installation command for Dependent Packages (Remote PC)

```
$ sudo apt-get install ros-kinetic-joy ros-kinetic-teleop-twist-joy ros-kinetic-teleop-twist-keyboard ros-kinetic-laser-proc ros-kinetic-rgbd-launch ros-kinetic-depthimage-to-laserscan ros-kinetic-rosserial-arduino ros-kinetic-rosserial-python ros-kinetic-rosserial-server ros-kinetic-rosserial-client ros-kinetic-rosserial-msgs ros-kinetic-amcl ros-kinetic-map-server ros-kinetic-move-base ros-kinetic-urdf ros-kinetic-xacro ros-kinetic-compressed-image-transport ros-kinetic-rqt-image-view ros-kinetic-gmapping ros-kinetic-navigation
```

TurtleBot3 Package Installtion (Remote PC)

```
$ cd ~/catkin_ws/src/
$ git clone https://github.com/ROBOTIS-GIT/turtlebot3.git
$ git clone https://github.com/ROBOTIS-GIT/turtlebot3_msgs.git
$ git clone https://github.com/ROBOTIS-GIT/turtlebot3_simulations.git
$ cd ~/catkin_ws && catkin_make
```

Next, install TurtleBot3 packages and dependent packages such as sensor package in the SBC of TurtleBot3.

Installation command for Dependent Package (TurtleBot3)

```
$ sudo apt-get install ros-kinetic-joy ros-kinetic-teleop-twist-joy ros-kinetic-teleop-twist-keyboard ros-kinetic-laser-proc ros-kinetic-rgbd-launch ros-kinetic-depthimage-to-laserscan ros-kinetic-rosserial-arduino ros-kinetic-rosserial-python ros-kinetic-rosserial-server ros-kinetic-rosserial-client ros-kinetic-rosserial-msgs ros-kinetic-amcl ros-kinetic-map-server ros-kinetic-move-base ros-kinetic-urdf ros-kinetic-xacro ros-kinetic-compressed-image-transport ros-kinetic-rqt-image-view ros-kinetic-gmapping ros-kinetic-navigation
```

TurtleBot3 Package Installtion (TurtleBot3)

```
$ cd ~/catkin_ws/src
$ git clone https://github.com/ROBOTIS-GIT/turtlebot3.git
$ git clone https://github.com/ROBOTIS-GIT/turtlebot3_msgs.git
$ git clone https://github.com/ROBOTIS-GIT/hls_lfcd_lds_driver.git
$ cd ~/catkin_ws && catkin_make
```

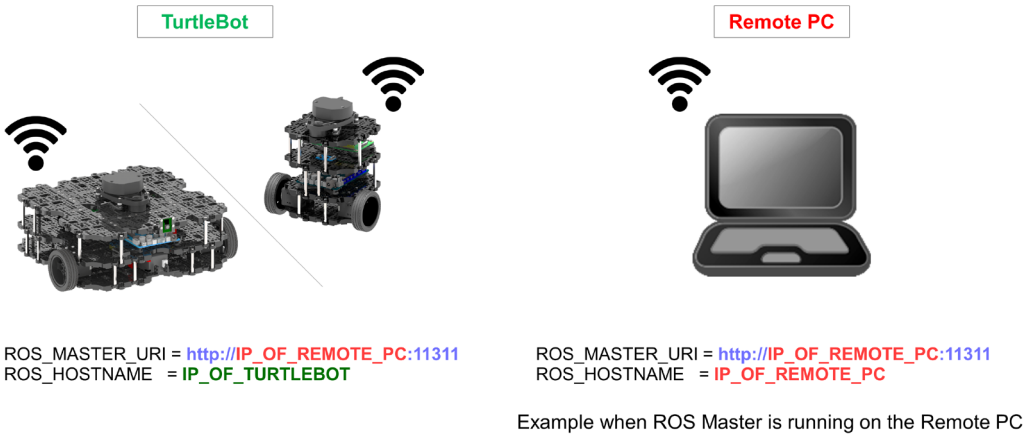

FIGURE 10-5 Setting Remote Control for TurtleBot3

Once all software are installed, it is important to configure the network environment as shown in Figure 10-5. For details on how to change the settings of ROS_HOSTNAME and ROS_MASTER_URI, refer to Section 3.2 and Section 8.3. TurtleBot3 uses desktop PC or laptop as a Remote PC, which acts as a master to run the roscore and takes process demanding controls such as SLAM and Navigation. The SBC in TurtleBot3 is responsible for operating robot components and sensor data collection. The following is an example of remote control setting when ROS Master is running on the remote PC.

Verify the IP address of the Remote PC

In the terminal window, use 'ifconfig' command to check the IP address of the remote PC (for example, 192.168.7.100).

ROS_HOSTNAME and ROS_MASTER_URI settings on the Remote PC

Modify the ROS_HOSTNAME and ROS_MASTER_URI settings in the '~/.bashrc' file as follows:

```
export ROS_HOSTNAME=192.168.7.100
export ROS_MASTER_URI=http://${ROS_HOSTNAME}:11311
```

Verify the IP address of TurtleBot3

Check the IP address of TurtleBot3 using 'ifconfig' command in the terminal window. Let's say the IP address of TurtleBot3 is 192.168.7.200. As a precaution, the TurtleBot must be in the same network area as the Remote PC.

ROS_HOSTNAME and ROS_MASTER_URI settings of the TurtleBot3 SBC

Modify the ROS_HOSTNAME and ROS_MASTER_URI settings in the '~/.bashrc' file as follows

```
export ROS_HOSTNAME=192.168.7.200
export ROS_MASTER_URI=http://192.168.7.100:11311
```

Now we have completed the development environment for TurtleBot3. In the next section, let's control TurtleBot3 with various ROS packages starting with remote control.

10.6. TurtleBot3 Remote Control

Let's take a look at remote control of TurtleBot3. Nearly all devices that can be connected to a PC, such as keyboard, bluetooth controller RC100, PS3 joystick, XBOX 360 joystick, Wii Remote, Nunchuk, Android app, LEAP Motion can be used to control TurtleBot3. For more information, refer to the teleoperation section at 'http://turtlebot3.robotis.com/'. In this section, we will use the most commonly used keyboard and the PS3 joystick, which is often used for robot control.

10.6.1. Controlling TurtleBot3

Run roscore (Remote PC)

On the remote PC, use the following command to run roscore. The roscore should be executed only once.

```
$ roscore
```

Execute the file turtlebot3_robot.launch [TurtleBot]

In TurtleBot, run the launch file 'turtlebot3_robot.launch' as follows. This launch file executes the 'turtlebot3_core' which is in charge of communication with OpenCR, the controller of the TurtleBot3, and 'hls_lfcd_lds_driver' node which drives LDS which is a 360 degree distance sensor.

```
$ roslaunch turtlebot3_bringup turtlebot3_robot.launch --screen
```

--screen option

Roslaunch can execute multiple nodes at the same time. However, messages from each node are not displayed by default. If necessary, you can use the '–screen' option to see all of the hidden messages during operation. If you are using roslaunch, it is recommended to append this option.

Execute the file turtlebot3_teleop_key.launch [Remote PC]

Execute the 'turtlebot3_teleop_key.launch' file from the Remote PC.

```
$ roslaunch turtlebot3_teleop turtlebot3_teleop_key.launch --screen
```

When you run this launch file, the 'turtlebot3_teleop_keyboard' node is executed and the following message appears in the terminal window. This node receives 'w', 'a', 'd', 'x' key inputs and transmits the translational and rotational speed to the robot in m/sec and rad/sec respectively. The 'spacebar' and 's' key will reset the translational and rotational speed to '0' to stop the movement of TurtleBot3.

```
Control Your TurtleBot3!
---------------------------
Moving around:
        w
   a    s    d
        x

w/x : increase/decrease linear velocity
a/d : increase/decrease angular velocity
space key, s : force stop

CTRL-C to quit
```

If you want to use the PS3 joystick instead of the keyboard to teleoperate the TurtleBot3, install the dependent package for joystick on the remote PC as follows. Then run the 'teleop.launch' file in the 'teleop_twist_joy' package to control the robot with the PS3 joystick. The PS3 joystick must be connected to the remote PC via Bluetooth.

```
$ sudo apt-get install ros-kinetic-joy ros-kinetic-joystick-drivers ros-kinetic-teleop-twist-joy
$ roslaunch teleop_twist_joy teleop.launch --screen
```

10.6.2. Visualization of TurtleBot3

Let's visualize the status of the robot on RViz. Before executing RViz, the 3D model has to be set as Burger. Use the below command to designate the 3D model of TurtleBot3 Burger. If you are running TurtleBot3 Waffle or Waffle Pi, you should set the TURTLEBOT3_MODEL parameter as 'waffle' or 'waffle_pi' instead of 'burger'. Then execute the 'turtlebot3_model.launch' file and RViz will be loaded.

```
$ export TURTLEBOT3_MODEL=burger
$ roslaunch turtlebot3_bringup turtlebot3_remote.launch
$ rosrun rviz rviz -d `rospack find turtlebot3_description`/rviz/model.rviz
```

When RViz is executed, the 3D model of TurtleBot3 Burger will be displayed at the origin of RGB coordinates along with the tf of each joint of the robot as shown in Figure 10-6. Also, the distance data from 360 degree LDS sensor can be displayed around the robot as red dots.

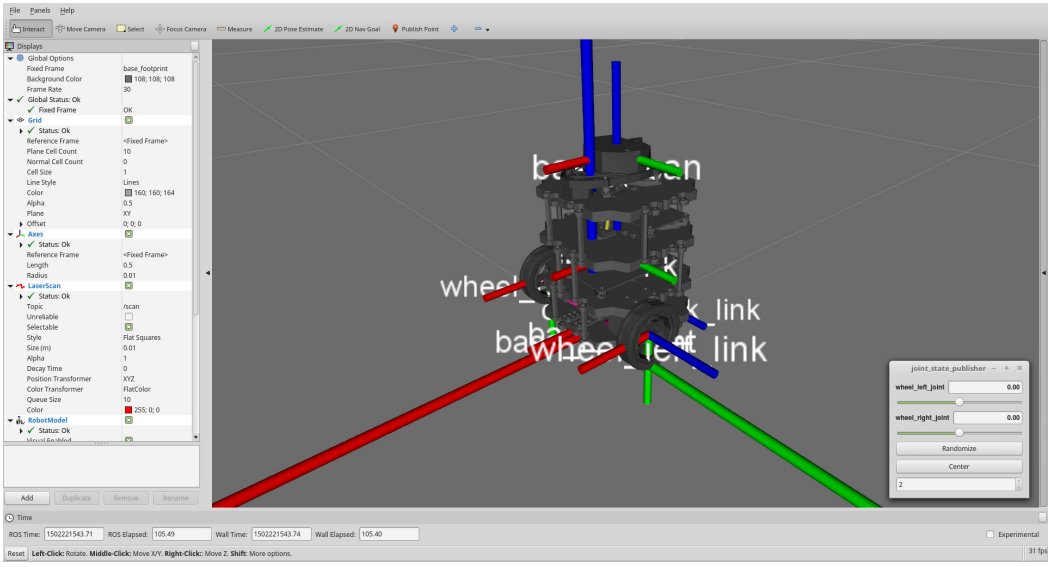

FIGURE 10-6 Visualization of TurtleBot3

In order to operate TurtleBot3, some works has to be done from local TurtleBot3 SBC, which will be a nuisance for the user to work back and forth on two computers. To solve this problem, remotely accessing the TurtleBot3 SBC from the remote PC using SSH is recommended. This allows you to execute commands on the TurtleBot3 SBC from the remote PC. Here's an example of how to remotely connect to the TurtleBot3 SBC from the remote PC. For more information, see the notes on SSH.

```
$ ssh turtlebot@192.168.7.200
```

SSH(Secure Shell)

SSH refers to the application or the protocol that allows you to log in to another computer in the network and to run commands on a remote system and copy files to another system. It is often used when connecting to remote computer and sends a command from a terminal window in Linux. To do this, the ssh application should be installed as follows.

```
$ sudo apt-get install ssh
```

To connect to a remote computer (in this case TurtleBot3), use the following command to connect in the terminal window. Once connection is established to the PC, commands can be entered just like using a local computer.

```
$ ssh username @ ip address of the remote PC
```

In case of Raspberry Pi (TurtleBot3 Burger and Waffle Pi), since the SSH server of Ubuntu MATE 16.04.x and Raspbian is disabled by default. If you want to enable SSH, please refer to the documents below.

```
https://www.raspberrypi.org/documentation/remote-access/ssh/
https://ubuntu-mate.org/raspberry-pi/
```

10.7. TurtleBot3 Topic

If you run roscore only on the remote PC and do not run any other nodes, 'rostopic list' command will return '/rosout' and '/rosout_agg'. Let's run TurtleBot3 by running the 'turtlebot3_robot.launch' file in the terminal window of TurtleBot3 SBC as we did for the remote control of TurtleBot above. When TurtleBot3 robot launch file is executed, the 'turtlebot3_core' node and the 'turtlebot3_lds' node will be running, and messages that are published from each node such as joint status, actuators, and the IMU can be received as topics.

```
$ roslaunch turtlebot3_bringup turtlebot3_robot.launch --screen
```

For example, the following 'rostopic list' command can verify that various topics are being published or subscribed:

```
$ rostopic list
/cmd_vel
/cmd_vel_rc100
/diagnostics
/imu
/joint_states
```

```
/odom
/rosout
/rosout_agg
/rpms
/scan
/sensor_state
/tf
```

In addition, launch the 'turtlebot3_teleop_key.launch' on the remote PC as you did for TurtleBot3 remote control:

```
$ roslaunch turtlebot3_teleop turtlebot3_teleop_key.launch --screen
```

To get more details on node and topic, run 'rqt_graph' as shown in the following example. You can then check out the topics published from and subscribed by the TurtleBot3 as shown in Figure 10-7.

```
$ rqt_graph
```

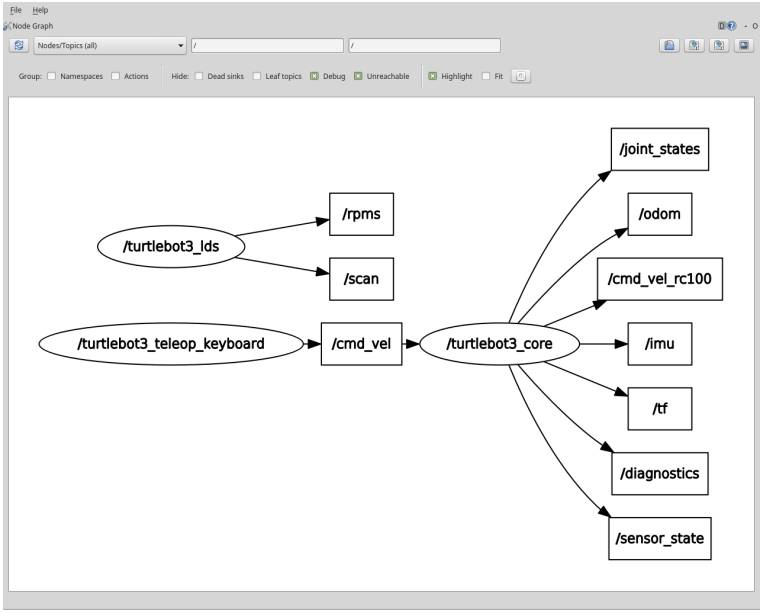

FIGURE 10-7 TurtleBot3 node and topic

10.7.1. Subscribed Topic

The topics mentioned above can be divided into subscribing topics received by TurtleBot3 and publishing topics transmitted from TurtleBot3. Among them, the subscribing topics are shown in the below table. You do not need to know all the subscribing topics, but it would be a good practice to learn how to use them. Above all things, let's take a look at 'cmd_vel'. This is a useful topic for controlling the robot, and the user can control the forward, backward, left and right rotation of the robot with this topic.

Topic Name	Format	Function
motor_power	std_msgs/Bool	Dynamixel Torque On/Off
reset	std_msgs/Empty	Reset Odometry and IMU Data
sound	turtlebot3_msgs/Sound	Output Beep Sound
cmd_vel	geometry_msgs/Twist	Control the translational and rotational speed of the robot. unit in m/s, rad/s (actual robot control)

* Topics used in TurtleBot3 may change depending on the purpose.

TABLE 10-1 Subscribed Topics of the TurtleBot3

10.7.2. Controlling a Robot using Subscribe Topic

For the previously mentioned subscribe topics, the robot receives and processes topics that the user has published. It is difficult to test every topic in this book, so let's just use a few subscribe topics. The following example stops the motor with 'rostopic pub' command in the terminal window.

```
$ rostopic pub /motor_power std_msgs/Bool "data: 0"
```

Next, let's control the velocity of the TurtlebBot3. The x and y used here are the translational speeds, and the unit is the ROS standard m/s. And z is the rotational speed in rad/s. When the value of x is 0.02 as shown in the following example, the TurtleBot3 advances at 0.02 m/s in the positive x-axis direction.

```
$ rostopic pub /cmd_vel geometry_msgs/Twist "linear:
  x: 0.02
  y: 0.0
  z: 0.0
angular:
  x: 0.0
```

```
  y: 0.0
  z: 0.0"
```

When the z value is given as 1.0 as shown in the following example, the TurtleBot3 will rotate counterclockwise at 1.0 rad/s.

```
$ rostopic pub /cmd_vel geometry_msgs/Twist "linear:
  x: 0.0
  y: 0.0
  z: 0.0
angular:
  x: 0.0
  y: 0.0
  z: 1.0"
```

10.7.3. Published Topic

Topics that the TurtleBot3 publishes are diagnostics, debugging, sensors related topics such as 'joint_states', 'sensor_state', 'odom', 'version_info' and 'tf'.

You do not need to know all the published topics, but it would be a good practice to learn how to use them. Especially, 'odom' for odometry information, 'tf' for transformation information, 'joint_states' for joint information, and sensor information related topics are essential when using TurtleBot3 hereafter.

Topic Name	Format	Function
sensor_state	turtlebot3_msgs/SensorState	Topic that contains the values of the sensors mounted on the TurtleBot3
battery_state	sensor_msgs/BatteryState	Contains battery voltage and status
scan	sensor_msgs/LaserScan	Topic that confirms the scan values of the LiDAR mounted on the TurtleBot3
imu	sensor_msgs/Imu	Topic that includes the attitude of the robot based on the acceleration and gyro sensor.
odom	nav_msgs/Odometry	Contains the TurtleBot3's odometry information based on the encoder and IMU
tf	tf2_msgs/TFMessage	Contains the coordinate transformation such as base_footprint and odom

Topic Name	Format	Function
joint_states	sensor_msgs/JointState	Checks the position (m), velocity (m/s) and effort (N · m) when the wheels are considered as joints.
diagnostics	diagnostic_msgs/DiagnosticArray	Contains self diagnostic information
version_info	turtlebot3_msgs/VersionInfo	Contains the TurtleBot3 hardware, firmware, and software information
cmd_vel_rc100	geometry_msgs/Twist	This topic is published when the Bluetooth-based controller, RC100, is connected to control the velocity (m/s) and angular speed (rad/s) of mobile robot.

* Topics used in TurtleBot3 may change depending on the purpose.

10.7.4. Verify Robot Status using Published Topics

Published topics mentioned above transmit the sensor value of the robot, the motor status, and the position of the robot on topics. In this section, you will subscribe some topics and verify the current state of the robot.

The 'sensor_state' topic mainly deals with analog sensors connected to the OpenCR embedded board. You can get information such as bumper, cliff, button, left_encoder, right_encoder as the following example.

```
$ rostopic echo /sensor_state
stamp:
  secs: 1500378811
  nsecs: 475322065
bumper: 0
cliff: 0
button: 0
left_encoder: 35070
right_encoder: 108553
battery: 12.0799999237
---
```

The 'odom' topic can be used to obtain odometry information, which records driving information. In TurtleBot3, the essential odometry information can be obtained based on gyro and encoder. The odometry is necessary for navigation.

```
$ rostopic echo /odom
header:
  seq: 30
  stamp:
    secs: 1500379033
    nsecs: 274328964
  frame_id: odom
child_frame_id: ''
pose:
  pose:
    position:
      x: 3.55720114708
      y: 0.655082702637
      z: 0.0
    orientation:
      x: 0.0
      y: 0.0
      z: 0.113450162113
      w: 0.993543684483
  covariance: [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.0]
twist:
  twist:
    linear:
      x: 0.0
      y: 0.0
      z: 0.0
    angular:
      x: 0.0
      y: 0.0
      z: -0.00472585950047
  covariance: [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.0]
---
```

The 'tf' topic is the translation and rotation information of each joint of the robot transformed in the relative coordinate to the 'base_footprint', such as coordinate transformation between 'base_footprint', which is the center position of the robot on the XY plane, and odom, which is the odometry information.

```
$ rostopic echo /tf
transforms:
  -
    header:
      seq: 0
      stamp:
        secs: 1500379130
        nsecs: 727869913
      frame_id: odom
    child_frame_id: base_footprint
    transform:
      translation:
        x: 3.55720019341
        y: 0.655082404613
        z: 0.0
      rotation:
        x: 0.0
        y: 0.0
        z: 0.112961538136
        w: 0.993599355221
---
```

You can also use the 'tf_tree' plugin in 'rqt' as shown in Figure 10-8 to visualize the tf information in GUI environment. In Figure 10-8, since the robot model information is missing and therefore each coordinate is not connected, but when performing coordinate translation with robot model information, the connection information of each joint can be used as shown in Figure 10-14.

```
$ rosrun rqt_tf_tree rqt_tf_tree
```

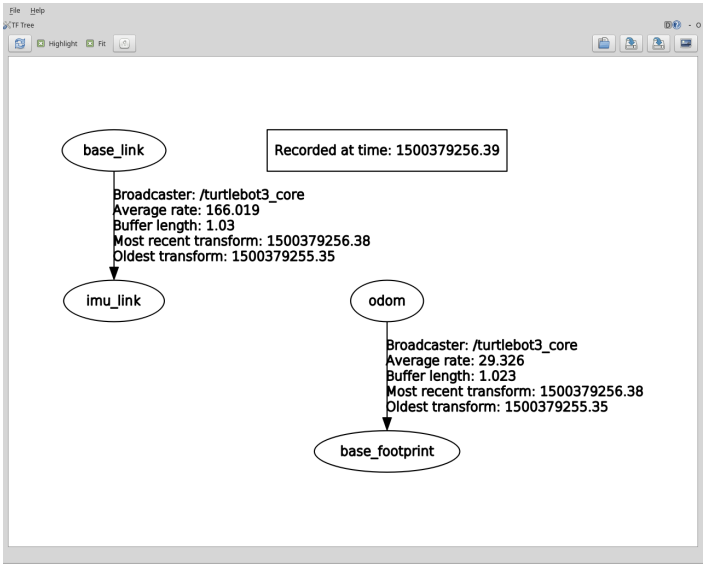

FIGURE 10-8 Visualization of the Coordinate Transformation through tf_tree

We have completed the topic section. ROS uses topic, service and action to communicate among nodes, which are the processors. Especially, topic is the most widely used message communication method.

10.8. TurtleBot3 Simulation using RViz

10.8.1. Simulation

TurtleBot3 supports development environment that can be programmed and developed with a virtual robot in the simulation. There are two development environments to do this, one is using 3D visualization tool 'RViz' and the other is using the 3D robot simulator 'Gazebo'.

In this section, we will first look into how to use RViz. RViz is a very useful to control TutleBot3 and test SLAM and Navigation using the 'turtlebot3_simulations' metapackage. To use virtual simulation with this metapackage, the 'turtlebot3_fake' package should be installed first. This is mentioned in Section '10. 5 TurtleBot3 Development Environment'. If you have already installed the package, move on to the next section.

10.8.2. Launch Virtual Robot

To launch the virtual robot, execute the 'turtlebot3_fake.launch' file in the turtlebot3_fake package as shown below.

```
$ export TURTLEBOT3_MODEL=burger
$ roslaunch turtlebot3_fake turtlebot3_fake.launch
```

The above command loads the 3D modeling file in the turtlebot3_description package and executes the 'turtlebot3_fake_node' which publishes fake topics as actual robot publishes, and the 'robot_state_publisher' node which publishes the transformation of each wheel to 'tf' topic. However, the sensor information cannot be used in RViz, a 3D simulator based on physics engine 'Gazebo' should be used. As Gazebo will be explained in the next section, we will take a look at Odometry and TF that can be verified during a simple movement.

In order to visualize TurtleBot3 on RViz, run RViz and go to [Global Options] → [fixed frame] and select '/odom'. Then click the 'Add' button in the bottom left corner of the window to add 'RobotModel' and display the 3D model file loaded from 'turtlebot3_fake.launch' in the center of the screen as shown in Figure 10-9.

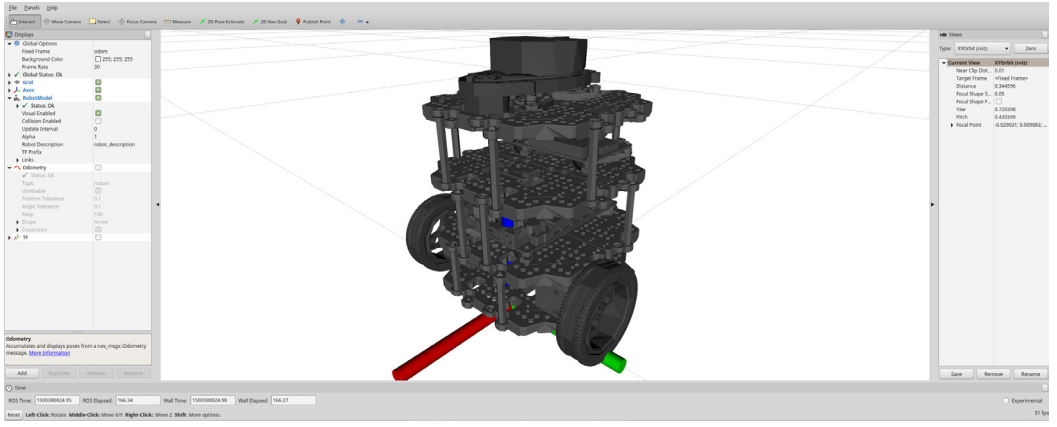

FIGURE 10-9 Load Virtual Robot

Next, let's run a virtual robot. Run the 'turtlebot3_teleop_key.launch' file from the 'turtlebot3_teleop' package, which lets you operate the robot remotely with the keyboard.

```
$ roslaunch turtlebot3_teleop turtlebot3_teleop_key.launch
```

When ‘turtlebot3_teleop_key.launch’ file is executed, ‘turtlebot3_teleop_keyboard’ node will be started. In ‘turtlebot3_fake_node’, the translational speed and the rotational speed are received from the ‘/cmd_vel’ topic published by ‘turtlebot3_teleop_keyboard’ node to operate TurtleBot3 virtually. In the terminal window where the ‘turtlebot3_teleop_key.launch’ file is running, use the keys below to control the robot.

- **w key**: Forward(+0.01 step, unit = m/sec)
- **x key**: Backward(-0.01 step, unit = m/sec)
- **a key**: Rotate CCW Direction(+0.1 step, unit = rad/sec)
- **d key**: Rotate CW Direction (-0.1 step, unit = rad/sec)
- **spacebar or s key**: Reset translation and rotation speed to 0
- **Ctrl + c**: Terminate the node

10.8.3. Odometry and TF

Now that we practiced driving the virtual robot, let’s check other topic values. For example, as shown in Figure 10-10, the ‘turtlebot3_fake_node’ receives the speed command to generate odometry information. The node publishes ‘odometry’, ‘joint_state’, and ‘tf’ via topic so they can be used to visualize the movement of TurtleBot3 in RViz.

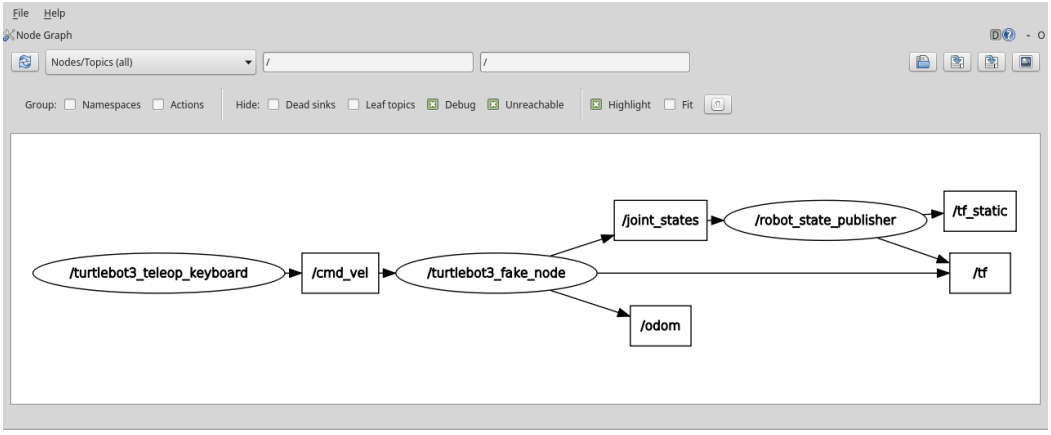

Figure 10-10 Visualized Nodes and Topics by rqt_graph

Let's first make sure that odometry information is generated and published properly. Although 'rostopic echo /odom' command in the terminal window can verify the information, let's visualize the odometry information with RViz. Click the 'Add' button at the bottom left of RViz, then select the 'By Topic' tab as shown in Figure 10-11 and add 'Odometry' by selecting it. A red arrow appears on the screen indicating the odometry in the forward direction of the TurtleBot. Uncheck 'Covariance' under the Odometry option and adjust the size of the Shaft and Head size since the initial arrow is too big for the robot.

FIGURE 10-11 Adding the odometry display to verify the odom topic

Now, let's move around the virtual TurtleBot3 using the 'turtlebot3_teleop_keyboard' node. As shown in Figure 10-12, red arrows are displayed on the robot's trajectory. This odometry is a very basic information indicating where the robot is currently located at. In the above practice, we checked that odometry information is displayed correctly.

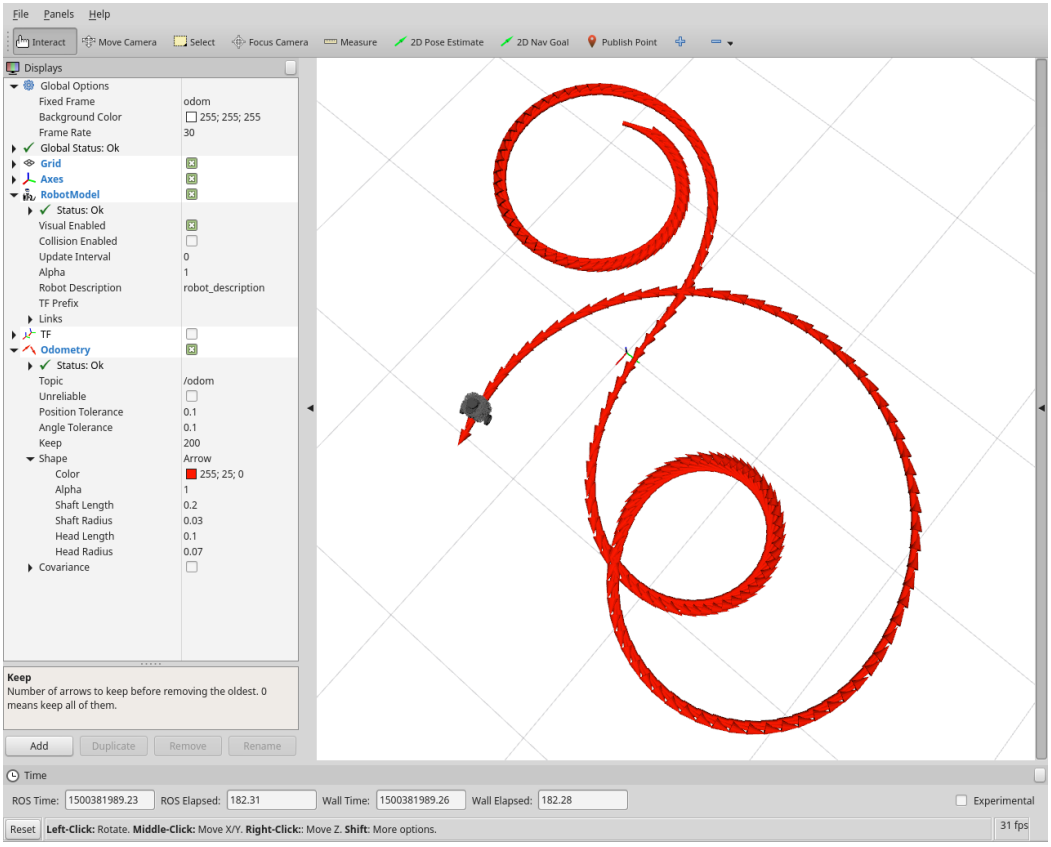

FIGURE 10-12 Movement and odometry information of the virtual TurtleBot3 Burger

The tf topic containing the relative coordinates of TurtleBot3 components can be verified with the rostopic command as before, but let's visualize it with RViz like odom and visualize the hierarchy with 'rqt_tf_tree'.

Click the Add button at the bottom left of RViz and select 'TF'. This will display 'odom', 'base_footprint', 'imu_link', 'wheel_left_link', 'wheel_right_link', etc., as shown in Figure 10-13. Let's move the virtual TurtleBot3 again using the 'turtlebot3_teleop_keyboard' node. As TurtleBot3 moves, you can see the 'wheel_left_link' and 'wheel_right_link' rotate.

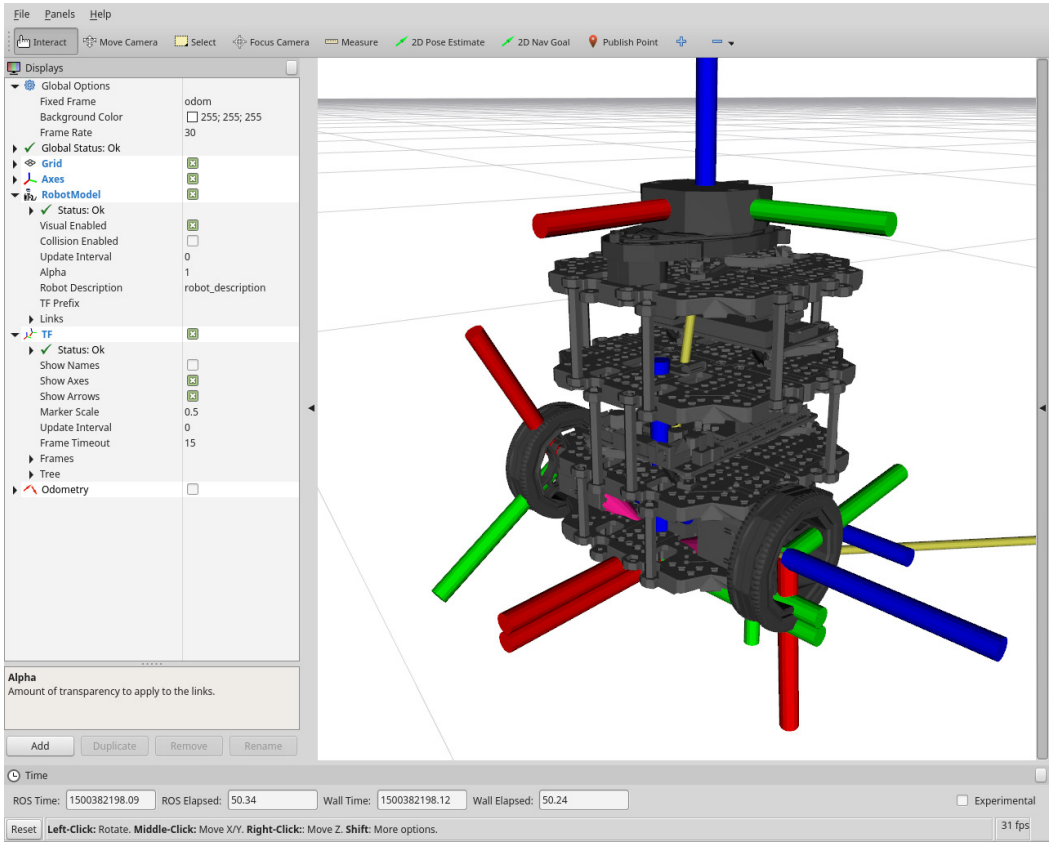

FIGURE 10-13 Visualized tf topics in Rviz

Now, run 'rqt_tf_tree' with the following command. We can see that the TurtleBot3 components are relatively transformed as shown in Figure 10-14. The position of sensors that can be mounted on the robot can be expressed likewise. This will be covered in detail in the next chapter.

```
$ rosrun rqt_tf_tree rqt_tf_tree
```

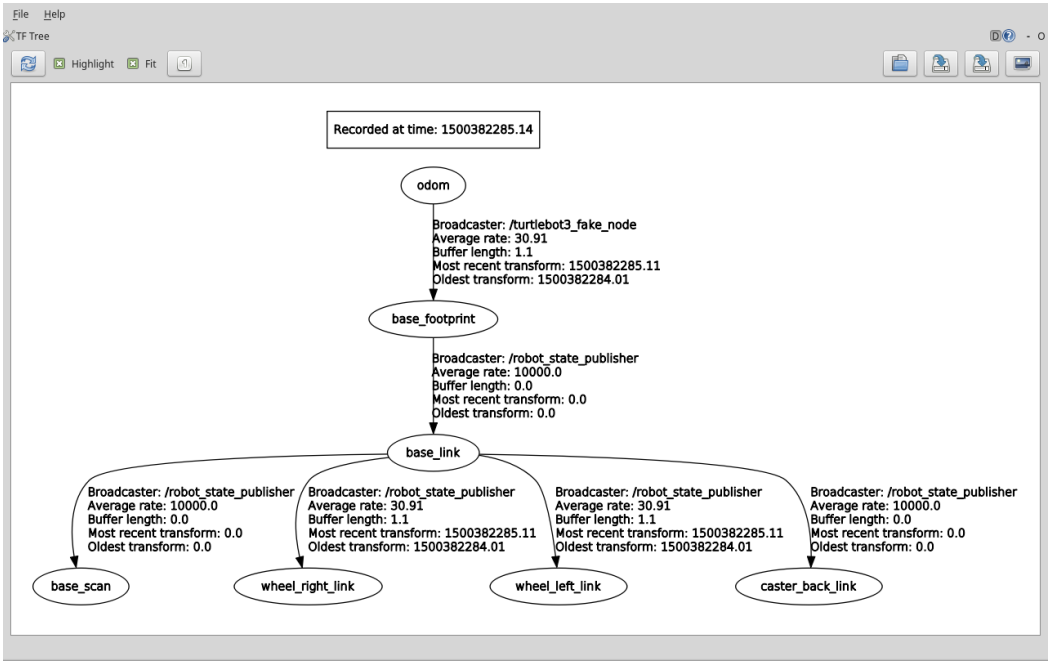

FIGURE 10-14 Visualized tf topics in rqt_tf_tree

10.9. TurtleBot3 Simulation using Gazebo

10.9.1. Gazebo Simulator

Gazebo is a 3D simulator that provides robots, sensors, environment models for 3D simulation required for robot development, and offers realistic simulation with its physics engine. Gazebo is one of the most popular simulators for open source in recent years, and has been selected as the official simulator of the DARPA Robotics Challenge[7] in the US. It is a very popular simulator in the field of robotics because of its high performance even though it is open source. Moreover, Gazebo is developed and distributed by Open Robotics which is in charge of ROS and its community, so it is very compatible with ROS.

The following are the characteristics[8] of Gazebo.

7 http://www.darpa.mil/program/darpa-robotics-challenge
8 http://gazebosim.org/

- Dynamics Simulation: In the early days of development, only ODE(Open Dynamics Engine) was supported. However, since version 3.0, various physical engines such as Bullet, Simbody, and DART are used to meet the needs of various users.
- 3D Graphics: Gazebo uses OGRE(Open-source Graphics Rendering Engines), which is often used in games, not only the robot model but also the light, shadow and texture can be realistically drawn on the screen.
- Sensors and Noise Simulation: Laser range finder (LRF), 2D/3D camera, depth camera, contact sensor, force-torque sensor and much more are supported and noise can be applied to the sensor data similar to the actual environment.
- Plug-ins: APIs are provided to enable users to create robots, sensors, environment control as a plug-in.
- Robot Model: PR2, Pioneer2 DX, iRobot Create, and TurtleBot are already supported in the form of SDF, a Gazebo model file, and users can add their own robots with an SDF file.
- TCP/IP Data Transmission: The simulation can be run on a remote server and Google's Protobufs, a socket-based message passing is used.
- Cloud Simulation: Gazebo provides cloud simulation CloudSim environment for use in cloud environments such as Amazon, Softlayer, and OpenStack.
- Command Line Tool: Both GUI and CUI tools are supported to verify and control the simulation status.

The latest version of Gazebo is 8.0, and just five years ago, it was 1.9. The current version is the adopted as a default application of the ROS Kinetic Kame used in this book. If ROS is installed as instructed in '3.1 ROS installation', Gazebo can be used without additional installation.

Now let's run Gazebo. If there are no problems, you can see that Gazebo is running as shown in Figure 10-15. For now, Gazebo can be seen as an independent simulator as it is not related to ROS.

```
$ gazebo
```

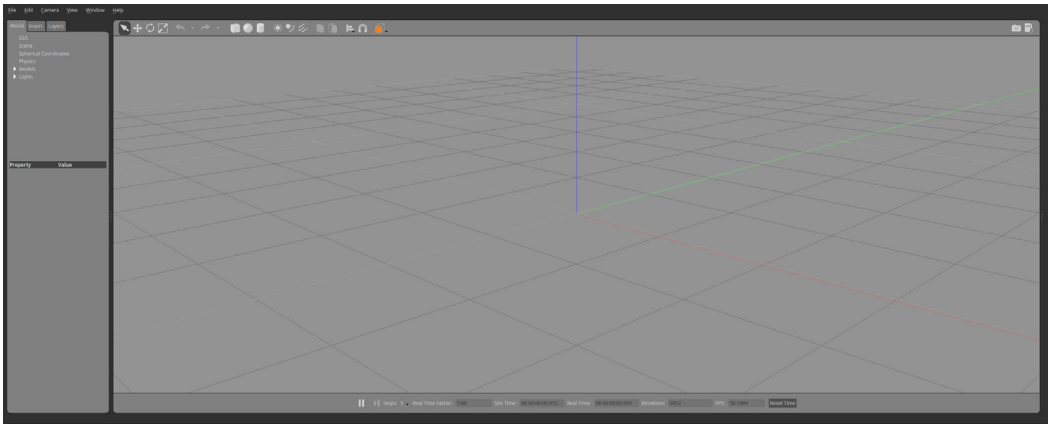

FIGURE 10-15 Gazebo initial screen

10.9.2. Launch Virtual Robot

Let's install dependent packages to run TurtleBot3 on the Gazebo simulator. The 'gazebo_ros_pkgs' metapackage that connects Gazebo and ROS is already installed, and 3D simulation package for TurtleBot3 'turtlebot3_gazebo' is also required, but it has already been installed in '10.5 TurtleBot3 Development Environment'.

Below is a command to set the 3D model file to Burger or Waffle, Waffle Pi. The example command is based on the Waffle which can get the camera information. Set the TURTLEBOT3_MODEL variable to 'waffle' with the following command. If the command is written in the '~/.bashrc' file, you do not need to set the model every time when opening a new terminal window.

```
$ export TURTLEBOT3_MODEL=waffle
```

Now run the launch file as shown in the following example. The 'gazebo', 'gazebo_gui', 'mobile_base_nodelet_manager', 'robot_state_publisher', and 'spawn_mobile_base' nodes are executed together and TurtleBot3 Waffle appears in the Gazebo screen, as shown in Figure 10-16. Gazebo is a 3D simulator that uses a lot of CPU, GPU and RAM resources due to the use of physics engine and graphic effects. Depending on the hardware specifications of PC, it may take a considerable amount of time to load the application.

```
$ roslaunch turtlebot3_gazebo turtlebot3_empty_world.launch
```

As shown in the following figure, you can see that only the robot is displayed in an empty plane.

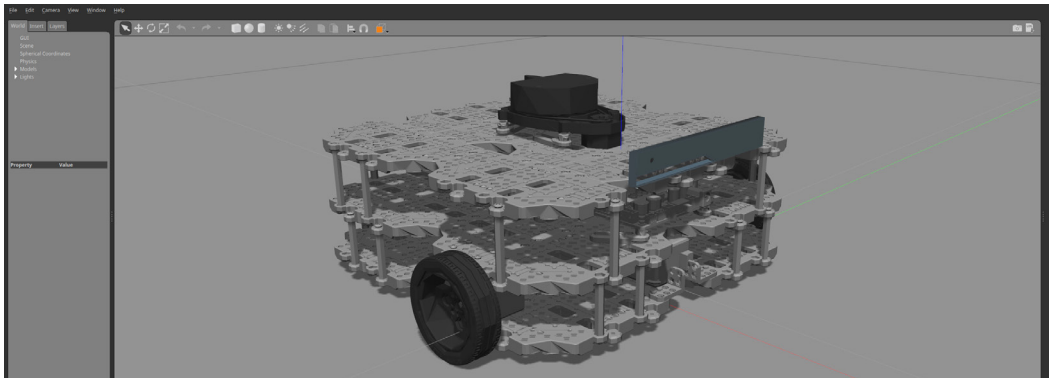

FIGURE 10-16 3D view of TurtleBot3 on Gazebo

In the above example, only the robot is loaded in the Gazebo. In order to perform the actual simulation, the user can specify the environment or load the environment model provided by Gazebo. An environment model can be added to Gazebo by clicking on 'Insert' at the top of the screen and selecting a file. There are various robot models and objects as well as environment models, so let's add them when necessary.

In this instruction, we will use provided environment. Close the currently active Gazebo screen by clicking the 'X' button in the upper left corner of the screen or enter [Ctrl + c] in the terminal window where you launched Gazebo.

Run the 'turtlebot3_world.launch' file as follows. The 'turtlebot3_world.launch' file will be loaded with the 'turtlebot3.world' environment model we already created. The turtlebot3.world environment model has been created from the symbol of TurtleBot series as shown in Figure 10-17. If you wish to create your own environment model, check the '/models/turtlebot3.world' file in the 'turtlebot3_gazebo' package to see how it is configured.

```
$ roslaunch turtlebot3_gazebo turtlebot3_world.launch
```

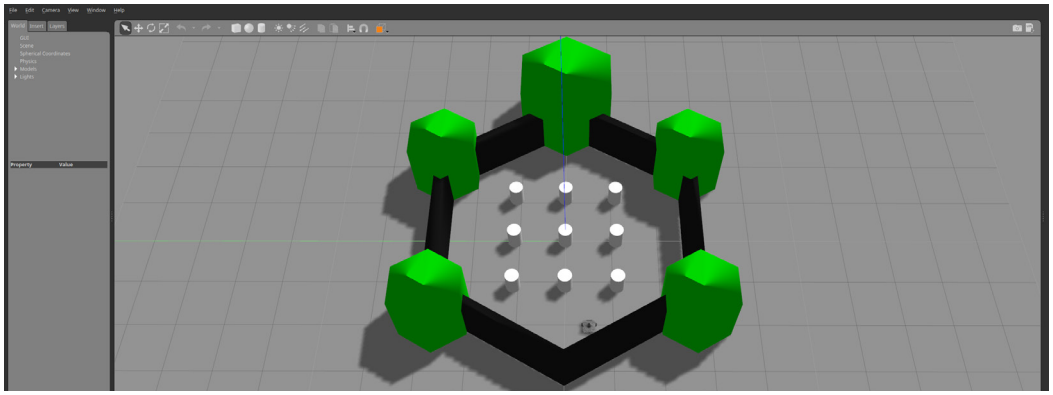

FIGURE 10-17 TurtleBot3 and Environment Model

Now run the remote control launch file in the following example to control the virtual TurtleBot3 in the Gazebo environment using the keyboard.

```
$ roslaunch turtlebot3_teleop turtlebot3_teleop_key.launch
```

Gazebo looks pretty similar to the RViz simulation of the previous section until now. Gazebo, however, is not only designed to look like a virtual robot, but it can also virtually check the collision, calculate the position, and use the IMU sensor and camera. Below is an example of using this launch file. When the file is executed, the virtual TurtleBot3 moves randomly in the loaded environment and avoid obstacles before running into the wall as shown in Figure 10-18. This is a great example of learning Gazebo.

```
$ export TURTLEBOT3_MODEL=waffle
$ roslaunch turtlebot3_gazebo turtlebot3_simulation.launch
```

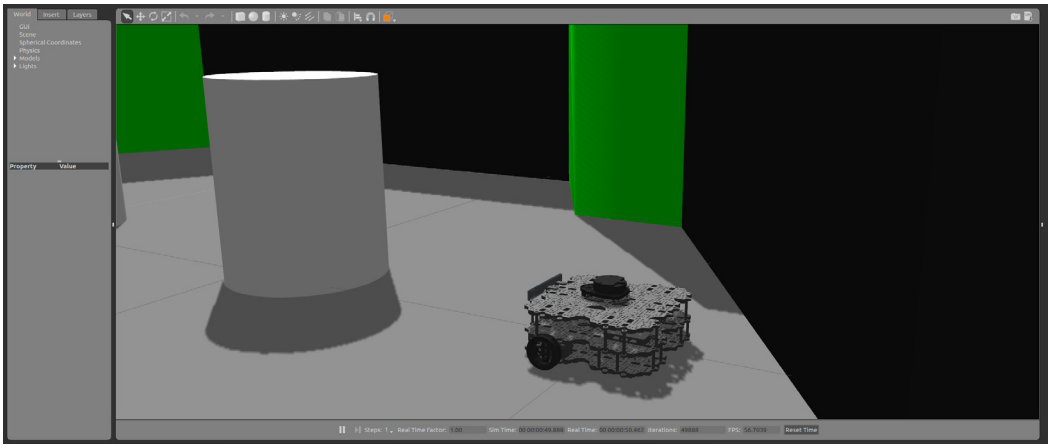

FIGURE 10-18 TurtleBot3 automatically moving and avoiding obstacles on Gazebo

In addition to this, let's run RViz with the following command. As shown in Figure 10-19, RViz can visualize the position of robot operating in Gazebo, distance sensor data and camera image. This simulation output is almost identical to operating an actual robot in the environment model designed just like the one in Gazebo.

```
$ export TURTLEBOT3_MODEL=waffle
$ roslaunch turtlebot3_gazebo turtlebot3_gazebo_rviz.launch
```

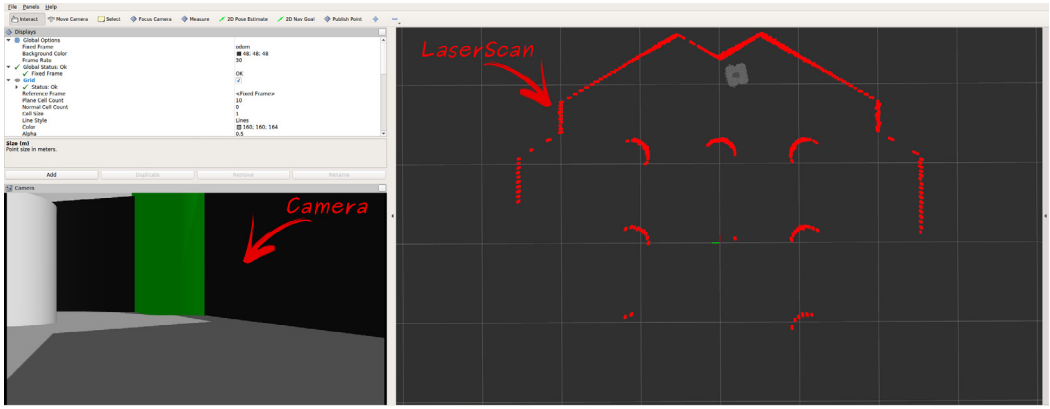

FIGURE 10-19 Visualized virtual LiDAR data and camera image in RViz

10.9.3. Virtual SLAM and Navigation

This section explains the use of command for virtual SLAM and Navigation. In the next chapter, we will cover SLAM for creating a map using the actual TurtleBot3 and navigation for moving to a specific destination on the given map. After familiarizing yourself with SLAM and navigation through Gazebo, try the actual operation in the next chapter.

Virtual SLAM Execution Procedure

When you run the dependent packages and move the robot in virtual space and create a map as shown below, you can create a map as shown in Figure 10-20 and the finalized map will look like the Figure 10-21.

Launch Gazebo

```
$ export TURTLEBOT3_MODEL=waffle
$ roslaunch turtlebot3_gazebo turtlebot3_world.launch
```

Launch SLAM

```
$ export TURTLEBOT3_MODEL=waffle
$ roslaunch turtlebot3_slam turtlebot3_slam.launch
```

Execute RViz

```
$ export TURTLEBOT3_MODEL=waffle
$ rosrun rviz rviz -d `rospack find turtlebot3_slam`/rviz/turtlebot3_slam.rviz
```

Remotely Control TurtleBot3

```
$ roslaunch turtlebot3_teleop turtlebot3_teleop_key.launch
```

Save the Map

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

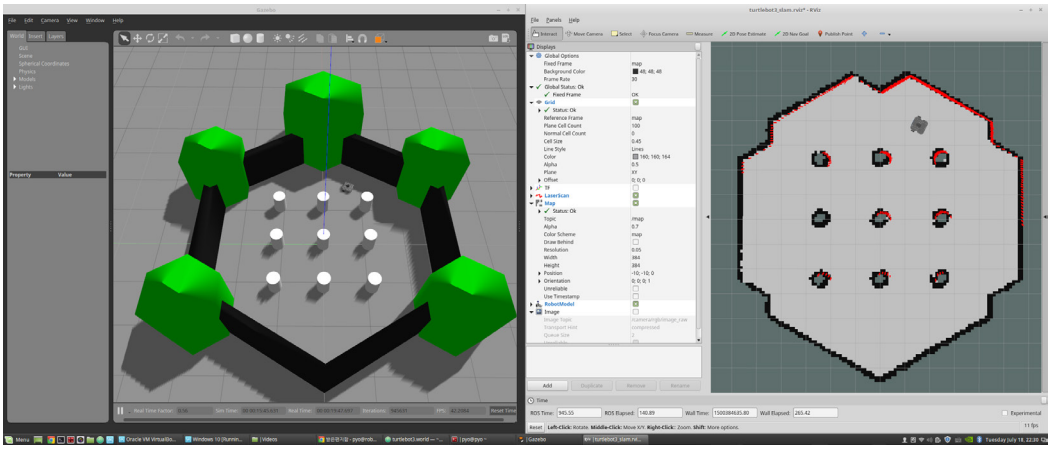

FIGURE 10-20 Running SLAM on Gazebo (Left: Gazebo, Right: RViz)

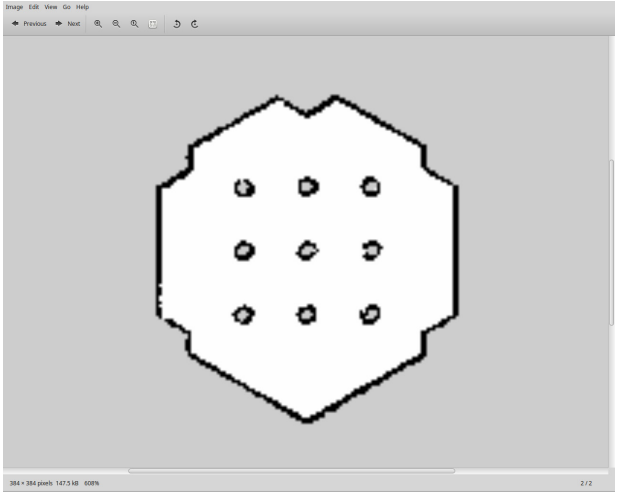

FIGURE 10-21 Generated Map of Gazebo Environment

Virtual Navigation Execution Procedure

Terminate all applications that were executed during the virtual SLAM practice and execute related packages in the following instruction, the robot will appear on the previously generated map. After setting the initial position of the robot on the map, set the destination to run the navigation as shown in Figure 10-22. The initial position only needs to be set once.

Execute Gazebo

```
$ export TURTLEBOT3_MODEL=waffle
$ roslaunch turtlebot3_gazebo turtlebot3_world.launch
```

Execute Navigation

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

Execute RViz and Set Destination

```
$ export TURTLEBOT3_MODEL=waffle
$ rosrun rviz rviz -d `rospack find turtlebot3_navigation`/rviz/turtlebot3_nav.rviz
```

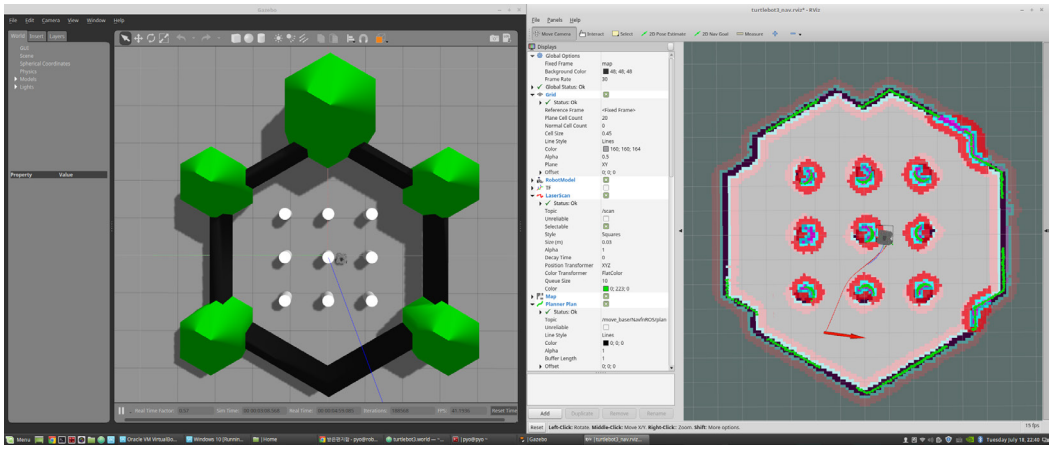

FIGURE 10-22 Running Naviation on Gazebo (Left: Gazebo, Right: RViz)

Two simulation methods of the TurtleBot3 package were introduced. One is to use RViz, a 3D visualization tool of ROS, and the other is to use Gazebo, a 3D robot simulator. Simulation is a great tool for users because it allows to perform programming tasks very close to the actual environment with a robot.

TurtleBot Simulation

TurtleBot supports three types of simulation which are stage, stdr, and Gazebo. Refer to the Wiki below to perform various simulations with a virtual robot.

```
http://wiki.ros.org/TurtleBot_stdr
http://wiki.ros.org/TurtleBot_gazebo
http://wiki.ros.org/TurtleBot_stage
```