ECE 5463 Introduction to Robotics
Spring 2018

ROS
TUTORIAL 3

Guillermo Castillo (Wei Zhang)
Department of Electrical and Computer Engineering
Ohio State University

02/2018

Outline

- **Rviz (Ros Visualization)**
  - Rviz – ROS
- **TurtleBot3**
  - Turtlebot components – laser sensor
  - Installing “TurtleBot3” Packages
  - Exploring “TurtleBot3” files (launch, world, URDF, XACRO)
- **TurtleBot3 simulation**
  - Running TurtleBot3 simulation (launch files)
  - Nodes and topics (current and needed)
  - Getting laser data (python script)
  - Rviz for laser data visualization
  - Goal: Make TurtleBot3 to move around avoiding obstacles

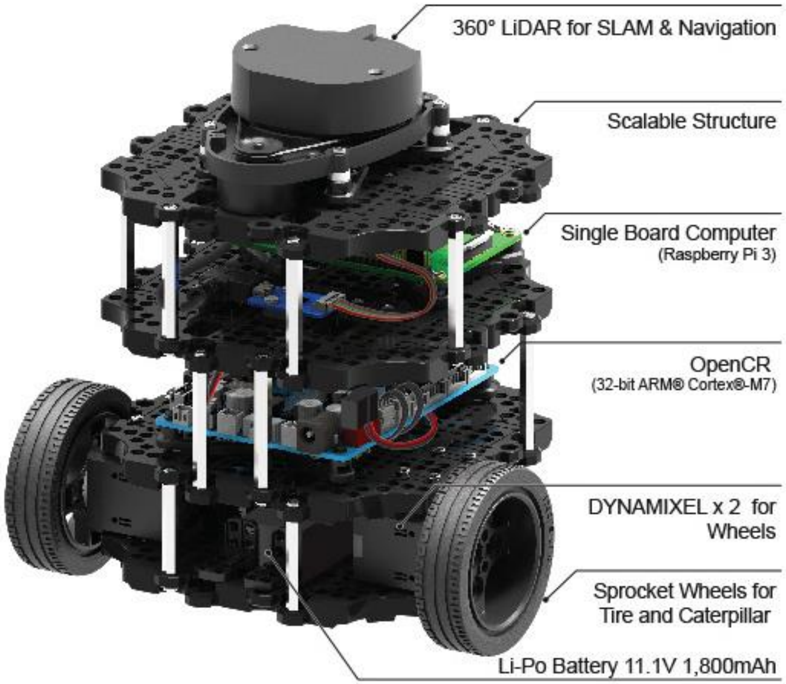
Rviz (Ros Visualization)

- Powerful 3D visualization tool for ROS.
- It allows the user to view the simulated robot model, log sensor information from the robot's sensors, and replay the logged sensor information.
- If an actual robot is communicating with a workstation that is running rviz, rviz will display the robot's current configuration on the virtual robot model.
- Command:

```
rosrun rviz rviz
```

TurtleBot3

- **Features**
  - Low-cost, personal robot kit with open-source software and hardware. Two models available: burger / waffle.
  - Modular, compact and customizable.
  - World's most popular ROS platform
- **Components (Burger model)**
  - Single Board Computer (SBC)
  - Sensors
    - Laser Sensor
    - Depth Camera
    - Video Camera
  - Control Board
  - Actuators – Dynamixel Series



Laser Sensor

360 LASER DISTANCE SENSOR LDS-01 (LIDAR)

- 2D laser scanner that collects a set of data around the robot to use for SLAM (Simultaneous Localization and Mapping).
- Light source: Semiconductor Laser Diode (λ=785nm)
- Distance Range: 120 ~ 3,500mm
- Angular Range: 360°
- Angular Resolution: 1°

Installing TurtleBot3 packages

- ROS released packages

```
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 ros-kinetic-interactive-markers
```

- From source code packages

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

Exploring TurtleBot3 files

LAUNCH FILES

Directory: `~/catkin_ws/src/turtlebot3_simulations/turtlebot3_gazebo/launch`

```
<launch>
  <arg name="model" default="$(env TURTLEBOT3_MODEL)" doc="model type [burger, waffle]"/>
  <arg name="x_pos" default="0.0"/>
  <arg name="y_pos" default="0.0"/>
  <arg name="z_pos" default="0.0"/>

  <include file="$(find gazebo_ros)/launch/empty_world.launch">
    <arg name="world_name" value="$(find turtlebot3_gazebo)/models/empty.world"/>
    <arg name="paused" value="false"/>
    <arg name="use_sim_time" value="true"/>
    <arg name="gui" value="true"/>
    <arg name="headless" value="false"/>
    <arg name="debug" value="false"/>
  </include>

  <param name="robot_description" command="$(find xacro)/xacro.py $(find turtlebot3_description)/
urdf/turtlebot3_$(arg model).urdf.xacro" />

  <node name="spawn_urdf" pkg="gazebo_ros" type="spawn_model" args="-urdf -model turtlebot3_burger
-x $(arg x_pos) -y $(arg y_pos) -z $(arg z_pos) -param robot_description" />
</launch>
```

Command:

```
$ export TURTLEBOT3_MODEL=burger
$ roslaunch turtlebot3_gazebo turtlebot3_empty_world.launch
```

Exploring TurtleBot3 files

WORLD FILES

Directory: `~/catkin_ws/src/turtlebot3_simulations/turtlebot3_gazebo/models`

```
<sdf version='1.4'>
  <world name='default'>
    <!-- A global light source -->
    <include>
      <uri>model://sun</uri>
    </include>

    <!-- A ground plane -->
    <include>
      <uri>model://ground_plane</uri>
    </include>

    <!-- A turtlebot symbol -->
    <include>
      <uri>model://turtlebot3</uri>
    </include>

    <scene>
      <ambient>0.4 0.4 0.4 1</ambient>
      <background>0.7 0.7 0.7 1</background>
      <shadows>true</shadows>
    </scene>
```

File: `turtlebot3_world.launch`

Exploring TurtleBot3 files

URDF FILES

Directory: `~/catkin_ws/src/turtlebot3/turtlebot3_description/urdf`

```
<?xml version="1.0" ?>
<robot name="turtlebot3_burger" xmlns:xacro="http://ros.org/wiki/xacro">
  <xacro:include filename="$(find turtlebot3_description)/urdf/common_properties.xacro"/>
  <xacro:include filename="$(find turtlebot3_description)/urdf/turtlebot3_burger.gazebo.xacro"/>

  <link name="base_footprint"/>

  <joint name="base_joint" type="fixed">
    <parent link="base_footprint"/>
    <child link="base_link"/>
    <origin xyz="0.0 0.0 0.010" rpy="0 0 0"/>
  </joint>

  <link name="base_link">
    <visual>
      <origin xyz="-0.032 0 0.0" rpy="0 0 0"/>
      <geometry>
        <mesh filename="package://turtlebot3_description/meshes/bases/burger_base.stl" scale="0.001
0.001 0.001"/>
      </geometry>
      <material name="light_black"/>
    </visual>
  </link>

  <joint name="wheel_left_joint" type="continuous">
    <parent link="base_link"/>
    <child link="wheel_left_link"/>
    <origin xyz="0.0 0.08 0.023" rpy="${-M_PI*0.5} 0 0"/>
    <axis xyz="0 0 1"/>
  </joint>
```

File: `turtlebot3_burger.urdf.xacro`

Running TurtleBot3 Simulation

- **Moving TurtleBot3 using teleop_key.**

```
$ export TURTLEBOT3_MODEL=burger
$ roslaunch turtlebot3_gazebo turtlebot3_empty_world.launch
```

In a separate terminal´s window:

```
roslaunch turtlebot3_teleop turtlebot3_teleop_key.launch
```

- **Moving TurtleBot3 using publisher node**
  - Create your own package (**Recall:** New packages must be created in the *src* folder from *catkin_ws*).
  - Create your own Python script for moving TurtleBot3 (**Recall:** Give execution permissions to the file using: *chmod +x name_of_the_file.py*)
  - Use commands *rosnode list, rostopic list, rostopic info, rosmsg show.*

Python Script for moving TurtleBot3

- **Package created:** *move_turtlebot3*
- **Python file created:** *trajectory.py*
  - You can use as a template the code used for your PA1. (**Recall:** Give execution permissions to the file using: *chmod +x name_of_the_file.py*).
  - Run the file using *rosrun* or *roslaunch* commands.

```
#! /usr/bin/env python

import rospy                                 # Import the Python library for ROS
from geometry_msgs.msg import Twist          # Import the Twist message from the std_msgs package

def talker():
        rospy.init_node('vel_publisher')             # Initiate a Node named 'vel_publisher'
        pub = rospy.Publisher('cmd_vel', Twist, queue_size=10)  # Create a Publisher object
        move = Twist()                   # Create a var named move of type Twist
        rate = rospy.Rate(1)             # Set a publish rate of 0.5 Hz
        while not rospy.is_shutdown():
                move.linear.x = 1
                move.angular.z = 1
                pub.publish(move)
                rate.sleep()

if __name__ == '__main__':
    try:
        talker()
    except rospy.ROSInterruptException:
        pass
```

Laser sensor application

- **Autonomous Navigation Demostration.**

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

In a separate terminal´s window

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

- **Visualizing sensor data using Rviz**

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

- Laser sensor data is shown as red dots in the Rviz (each dot corresponds to a laser beam).

Getting laser data using ROS commands and Python script

- Laser data is published on the topic *scan.* Therefore, to access this data we have to subscribe to this topic, obtain the required data and use it for our desired application.
- Obtain information about the topic (in a separate window):

```
$ rostopic list
$ rostopic info scan
$ rosmsg show LaserScan
$ rostopic echo scan
```

- Information of interest of laser scan:

float32 range_min: 0.1199
float32 range_max: 3.5
float32 angle_min: 0.0
float32 angle_max: 6.28000020981
float32[] ranges: [1.3471, 1.3377, ….. , 1.3471, 1.3377] – 360 elements
float32[] intensities: [4.05e-08, 4.54+30, …. , 4.54e+30, 4.05e-08] – 360 elements ***

Getting laser data using ROS commands and Python script

- Create a new node to subscribe to the topic *scan* and get the information from the laser sensor.

```
gedit laser_data.py
```

```
#! /usr/bin/env python

import rospy
from sensor_msgs.msg import LaserScan

def callback(msg):        # Define a function called 'callback' that receives a parameter named 'msg'
    print('===================================================')
    print('s1 [0]')       #value front-direction laser beam
    print msg.ranges[0]   # print the distance to an obstacle in front of the robot. the sensor returns a vector
                          # of 359 values, being the initial value the corresponding to the front of the robot

    print('s2 [90]')
    print msg.ranges[90]

    print('s3 [180]')
    print msg.ranges[180]

    print('s4 [270]')
    print msg.ranges[270]

    print('s5 [359]')
    print msg.ranges[359]


rospy.init_node('laser_data')                          # Initiate a Node called 'laser_data'
sub = rospy.Subscriber('scan', LaserScan, callback)   # Create a Subscriber to the laser/scan topci

rospy.spin()
```

Turtlebot3 - Obstacle avoidance

- Now it’s time to put everything together: Subscriber, Publisher, Messages. You will need to use all of this concepts in order to succeed!
- Goal: Make robot avoid obstacles in front of him.
- Baby step: Make the robot to stop when an obstacle in front of the robot is closer than 0.5 m.
- Hints:
  - Create a node which is a publisher and subscriber at the same time.
  - The node should subscribe to the topic *scan* and publish on the topic *cmd_vel*
  - Use the code implemented in the previous scripts and put everything together.
  - Use conditionals to make the robot behave as you want.

Turtlebot3 - Obstacle avoidance

- Create node:

```
$ gedit avoid_obstacle.py
$ chmod +x avoid_obstacle.py
```

```
#! /usr/bin/env python

import rospy
from sensor_msgs.msg import LaserScan
from geometry_msgs.msg import Twist

def callback(msg):          # Define a function called 'callback' that receives a parameter named 'msg'
    print('===================================================')
    print('s1 [270]')       #value right-direction laser beam
    print msg.ranges[270]
    print('s2 [0]')         #value front-direction laser beam
    print msg.ranges[0]     # print the distance to an obstacle in front of the robot. the sensor returns a vector
                            # of 359 values, being the initial value the corresponding to the front of the robot
    print('s3 [90]')        #value left-direction laser beam
    print msg.ranges[90]


    #If the distance to an obstacle in front of the robot is bigger than 1 meter, the robot will move forward
    if msg.ranges[0] > 0.5:
      move.linear.x = 0.5
      move.angular.z = 0.0
    else:
      move.linear.x = 0.0
      move.angular.z = 0.0

    pub.publish(move)

rospy.init_node('obstacle_avoidance')                 # Initiate a Node called 'obstacle_avoidance'
sub = rospy.Subscriber('/scan', LaserScan, callback)  # Create a Subscriber to the /scan topic
pub = rospy.Publisher('/cmd_vel', Twist)              #Create a publisher on the /cmd_vel topic
move = Twist()

rospy.spin()
```

Thanks for your attention