Simulating Robots Using ROS and Gazebo

Daniel Tobón Collazos, *Member, IEEE.*
2126550
Universidad Autónoma de Occidente

Abstract*— *The present work is the application of chapter 3 of mastering ros for robotic programming for the simulation of the seven dof arm and the differential mobile robot using Gazebo. In this document, we are going to use Gazebo simulator to simulate the seven DOF arm and the differential wheeled mobile robot that was designed in the previous laboratory [2]. The list of topics that we are going to cover is: simulating robotic arms in Gazebo, adding sensors to the robotic arm simulation, interfacing Gazebo to ROS, adding ROS controllers to robots, working with the robotic arm joint control, simulating the mobile robot in Gazebo, adding sensors to mobile robot simulation and moving the mobile robot in Gazebo using a keyboard teleop.

***Index Terms*—Gazebo, ROS, Xacro.**

I. Introduction

Gazebo [1] is a multirobot simulator for complex indoor and outdoor robotic simulation. In Gazebo, we can simulate complex robots, robot sensors, and a variety of 3D objects; also, has a good interface in ROS, which exposes the whole control of Gazebo in ROS.

After designing the 3D model of a robot, the next phase is its simulation. Robot simulation will give you an idea about the working of robots in a virtual environment.

In this document, we are going to use Gazebo simulator to simulate the seven DOF arm and the differential wheeled mobile robot that was designed in the previous laboratory [2]. The list of topics that we are going to cover is: simulating robotic arms in Gazebo, adding sensors to the robotic arm simulation, interfacing Gazebo to ROS, adding ROS controllers to robots, working with the robotic arm joint control, simulating the mobile robot in Gazebo, adding sensors to mobile robot simulation and moving the mobile robot in Gazebo using a keyboard teleop.

Before starting with Gazebo and ROS, we should install the following package to work with gazebo and ROS:

```
$ sudo apt-get install ros-kinetic-gazebo-ros-pkgs ros-kinetic-gazebo-ros-control
```

For this step is necessary have the `ros-kinetic-desktop-full` software installed in Ubuntu. The use of each package is a follows: `gazebo-ros-pkgs,` contains wrappers and tools for interfacing ROS with Gazebo; `gazebo-ros-control,` contains standard controllers to communicate between ROS and Gazebo.

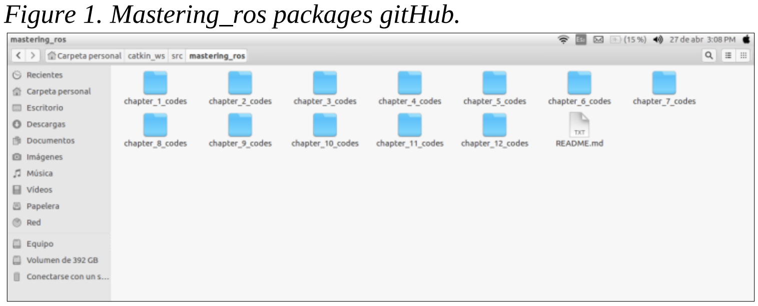
After installation, check whether the Gazebo is properly installed in Ubuntu using the following command: `$ gazebo.`

On the other hand, the necessary packages to carry out this practice are taken from the exercises of chapter 3 of the book Mastering Ros for Robotics Programming, so in the page github we can download the repository that contains the exercises of each chapter or in a terminal of Ubuntu type:

```
$ git clone
https://github.com/qboticslabs/mastering_ros.git
```

This repository should be downloaded into the src folder of catkin_ws.

Figure 1. Mastering_ros packages gitHub.



Inside the chapter_3_codes folder are the packages with the files needed to work in this lab. The mastering_ros_robot_description_pkg and the seven_dof_arm_gazebo packages must be copied to the src folder of catkin_ws to build the executables with *catkin_make* and then to be able to visualize the model of the robot in Gazebo. Only need to obtain a package more to be able to proceed to the practical content of this laboratory.

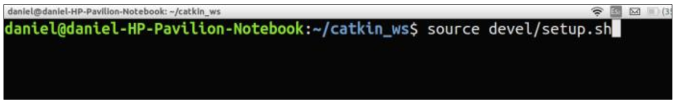
If you already have in the *catkin_ws* the package xacro of ROS Wiki this step can be omitted; Otherwise, the following should be done: on the ROS Wiki page, you will find the repository of the xacro package that contains the necessary files for the elaboration of this practice. Just as it was done to get the mastering_ros package, it is downloaded into the src folder of catkin_ws the xacro package with the following command.

The Robotic Arm Simulation Model for Gazebo

The Seven Dof Arm xacro contains the robot description; Some simulation parameters belong to Gazebo, so the xacro file is updated. The difference with the previous laboratory is that instead of executing the launch file of the robot description to visualize it in Rviz, the launch that describes the robot in the graphic environment of Gazebo is executed; Seven_dof_arm_world.launch is the file that must be executed to display the model of the robot in Gazebo.

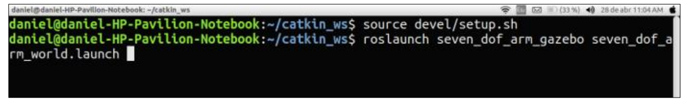
As a first step, it runs in a terminal the setup.sh (file necessary for ubuntu to recognize in Gazebo the xacro description of the robot).

Figure 2. Sourcing the setup.bash.



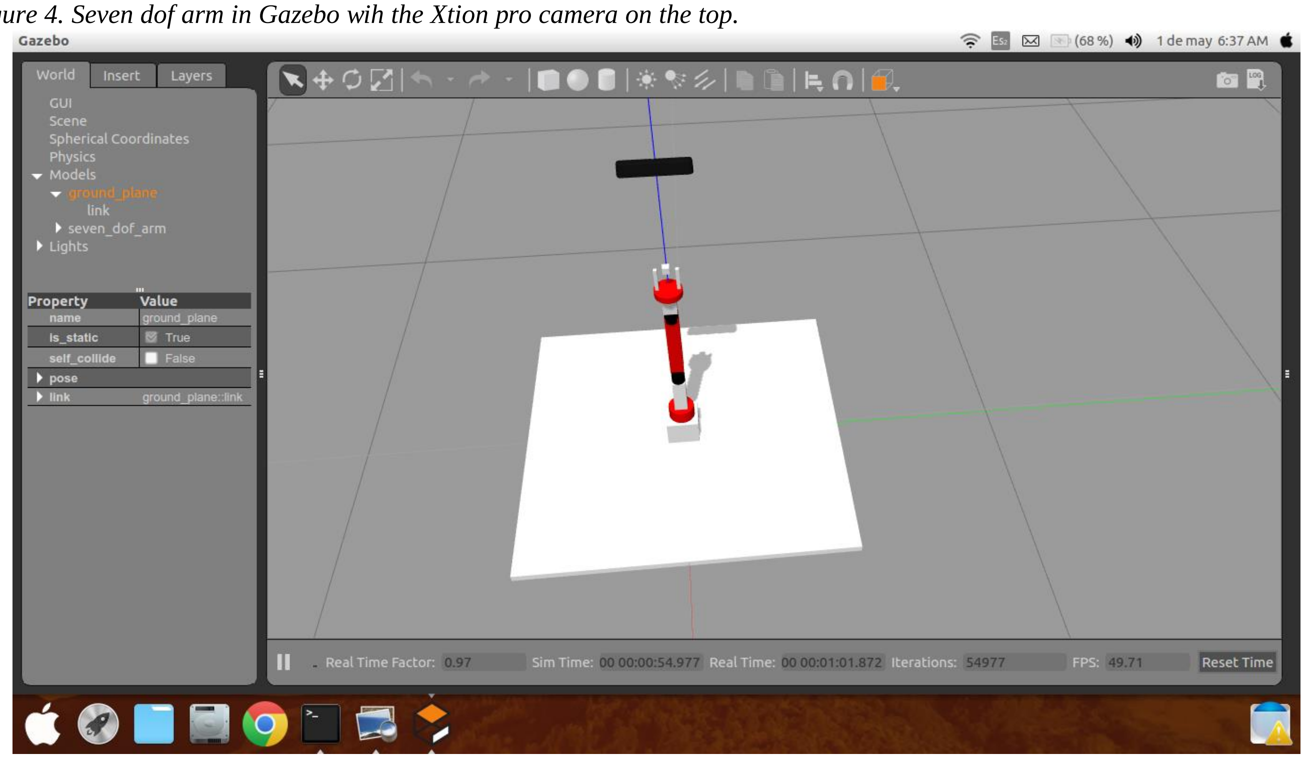
Once the setup.sh is executed, the robot's xacro model can be visualized in gazebo as follows: the *roslaunch* code line is executed to execute the launch file itself (seven_dof_arm_world.launch), as the second argument *[package name=seven_dof_arm_gazebo]*, And finally, the name of the launch file to be executed, in this case, *seven_dof_arm_world.launch*. In the terminal of ubuntu would look like this.

Figure 3. Launching the seven_dof_arm_world.launch.



If all goes well, you should run Gazebo and view the model of the seven dof arm robot, like this:

Figure 4. Seven dof arm in Gazebo wih the Xtion pro camera on the top.

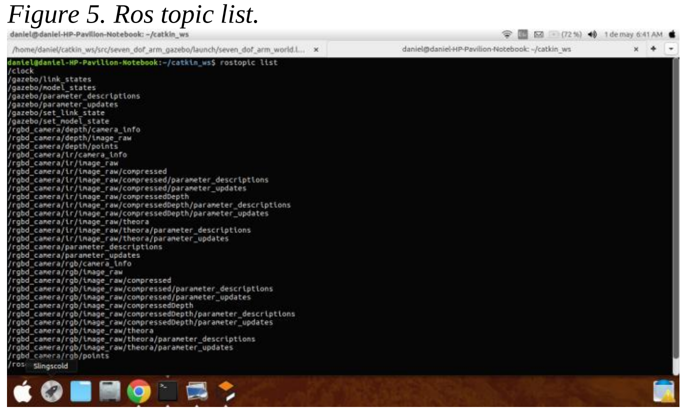


On the top, we can see a sensor Xtion Pro that provides the correct image output. We can listo ut the topics generated while performing simulation for visualizing the 3D sensor data.

The following line of code shows the visualization of the 3d sensor data.

```
$ rostopic list
```

Figure 5. Ros topic list.



Just as it was done in the previous laboratory to describe the texture and color of each robot link, a tag is defined for the color and texture in gazebo within the xacro file that describes the robot.

Inside of xacro file that describes the robot model of seven dof arm is added a tag that allows to act the model in Gazebo using the controls of ROS defined as <transmission> tag, that define the transmission element to link actuator to joint. To build a sensor in Gazebo, we have to model the behavior of that sensor in Gazebo. There are some prebuilt sensor models in Gazebo that can be used directly in our code without writing a new model. Here, we are adding a 3D vision sensor called the Asus Xtion Pro model in Gazebo. The sensor model is already implemented in the gazebo_ros_pkgs/gazebo_plugins ROS package, which we already installed in our ROS system.

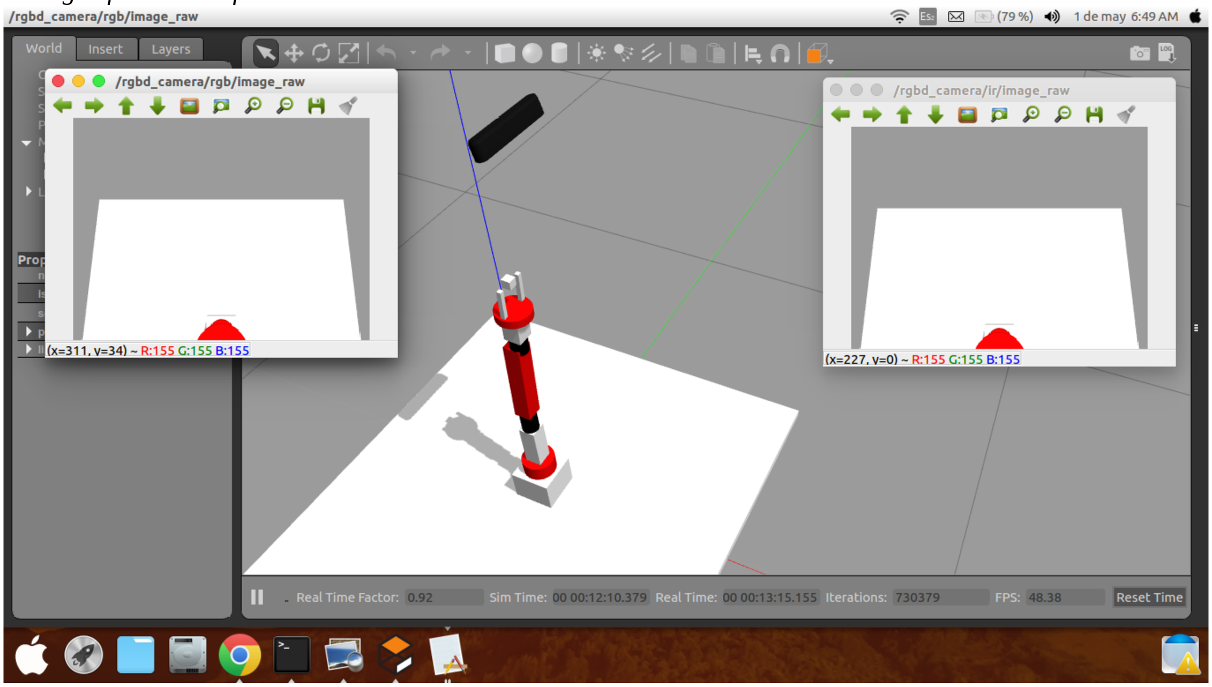
We can work with the Xtion Pro data from Gazebo and check whether it provides the correct image. Let's view the image data of a 3D vision sensor using the following tool called image_view.

```
$ rosrun image_view image_view image:=/rgbd_camera/rgb/image_raw
```

```
$ rosrun image_view image_view image:=/rgbd_camera/ir/image_raw
```

Here is the screenshot with all these images.

Figure 6. Image of Xtion Pro from Gazebo.

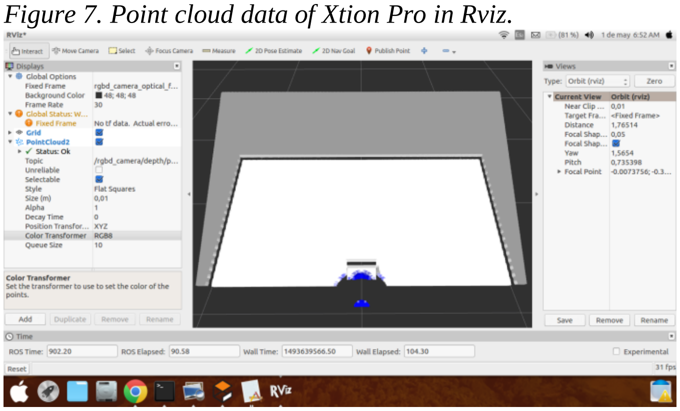


We can also view the point cloud data of this sensor in RViz.

```
$ rosrun rviz rviz -f /rgbd_camera_optical_frame
```

Add a *PointCloud2* display type and Topic as */rgbd_camera/depth/points*. We will get a point cloud view as follows:

Figure 7. Point cloud data of Xtion Pro in Rviz.



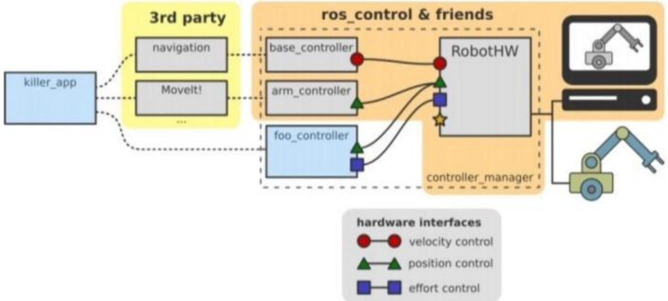
However, we are going to discuss how to move each joint of the robot in Gazebo. To move each joint, we need to assign a ROS controller. In each joint, we need to attach a controller that is compatible with the hardware interface mentioned inside the transmission tags.

The main function of the hardware interface is that it will act as a mediator between ROS controllers and the Real Hardware/Simulator and locate the necessary resources for the controllers.

In this robot, we have defined the position controllers, velocity controllers, effort controllers, and so on. The ROS controllers are provided by a set of packages called ros_control.

The following figure shows the interconnection of the ROS controller, robot hardware interface, and simulator/real hardware.

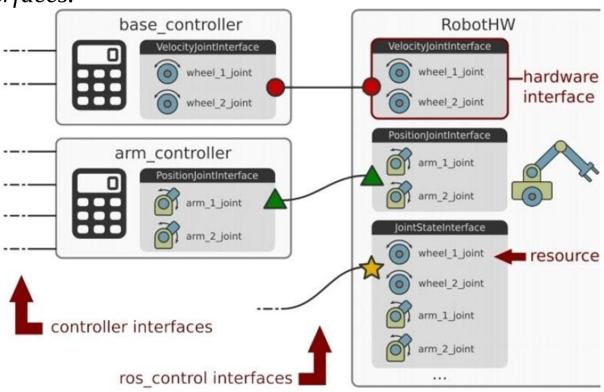
Figure 8. Interacting ROS controllers with Gazebo.



Source: Interacting ROS controllers with Gazebo. Chapter 3: Simulating Robots Using ROS and Gazebo, Pag. 140.

The hardware interface allocates each resource for the controllers and sends values to each resource. The detailed diagram of communications between the robot controllers and robot hardware interfaces are shown as follows:

Figure 9. Illustration of ROS controllers and hardware interfaces.



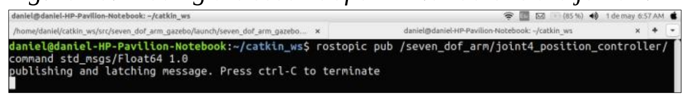
The hardware interface is a software representation of the robot and its abstract hardware. The resource of the hardware interfaces are actuators, joints, and sensors.

The launch files start the Gazebo simulation of the arm, load the controller configuration, load the joint state controller and joint position controllers, and at last, it runs the robot state publisher, which publishes the joint states and tf. Let's check the controller topics generated after running this launch file:

```
$ roslaunch seven_dof_arm_gazebo seven_dof_arm_gazebo_control.launch
```

To move a robot joint in Gazebo, we have to publish a joint value with a message type std_msgs/Float64 to the joint position controller command topics. Here is an example of moving the fourth joint to 1.0 radians.

Figure 10. Moving the seven dof arm 1.0 radians the joint 4.



The following figure shows the movement of the seven dof arm 1.0 radians, around of joint 4.

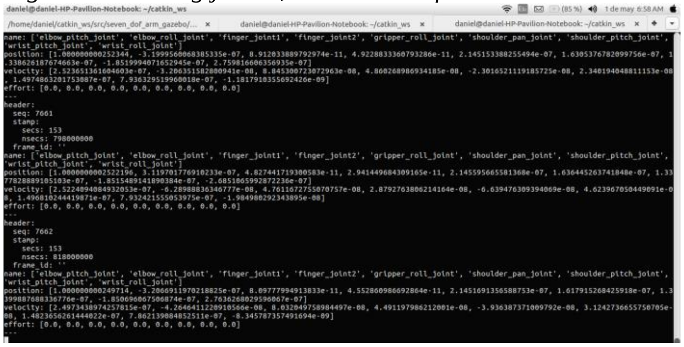
Figure 11. Moving joint 4, 1.0 radians of the arm in Gazebo.

We can also view the joint states of the robot by using the following command:

```
$ rostopic echo /seven_dof_arm/joint_states
```

The following figure shows the joint states of the robot:

Figure 11. Moving joint 4, 1.0 radians of the arm in Gazebo.

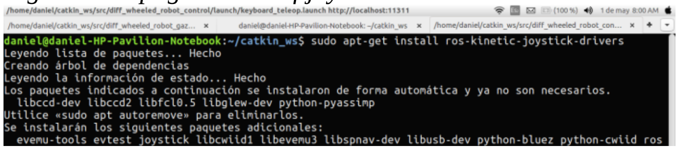


THE ROBOTIC DIFFERENTIAL DRIVE SIMULATION MODEL FOR GAZEBO

Before you can visualize the differential robot in Rviz, you need to download the joy package, which contains the drivers for the joystick of the robot and to be able to move the robot with the keys.

To do this, you write the following line of code in the terminal of Ubuntu:

Figure 12. Apt get install of joystick drivers.

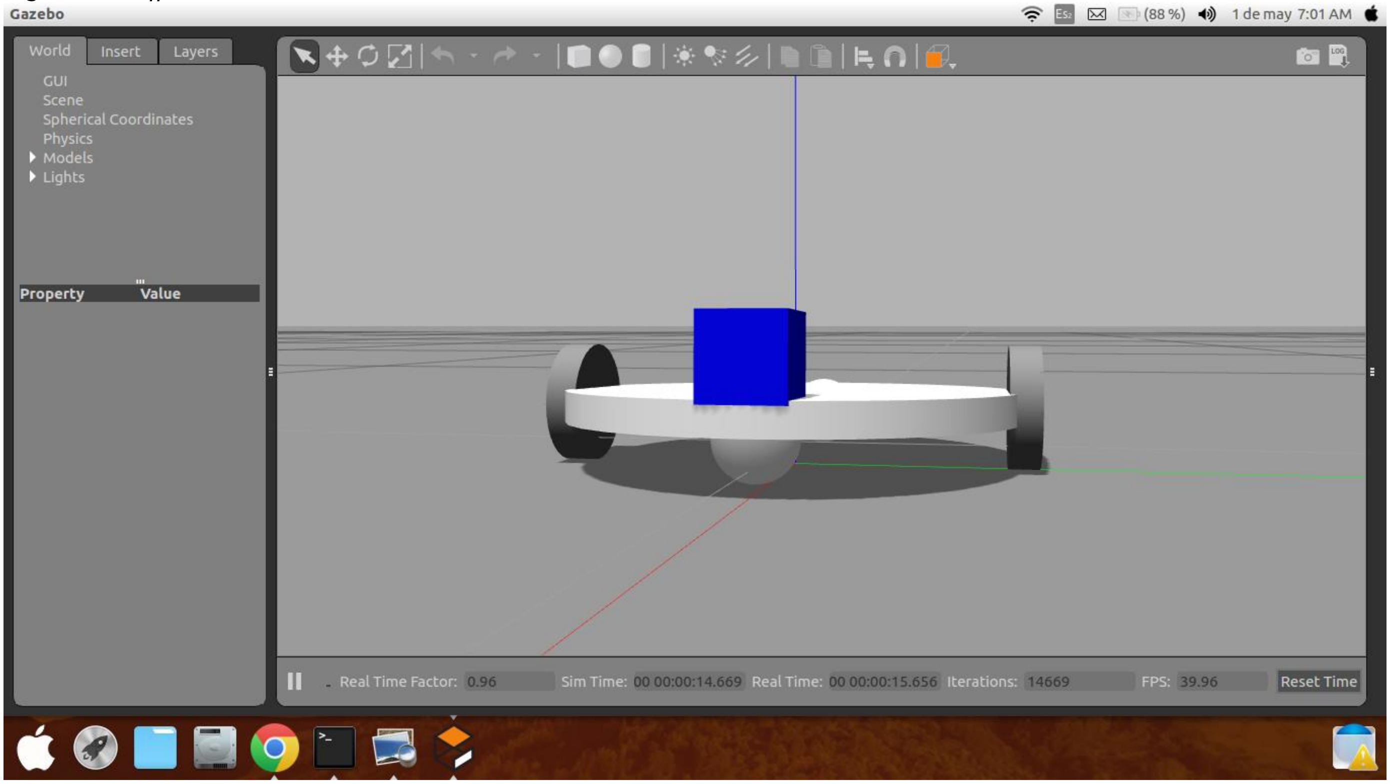


After the drivers have been downloaded from the joystick, you can proceed to visualize the mobile robot differential in gazebo with the following line of code:

```
$ roslaunch diff_wheeled_robot_gazebo diff_wheeled_robot_gazebo.launch
```

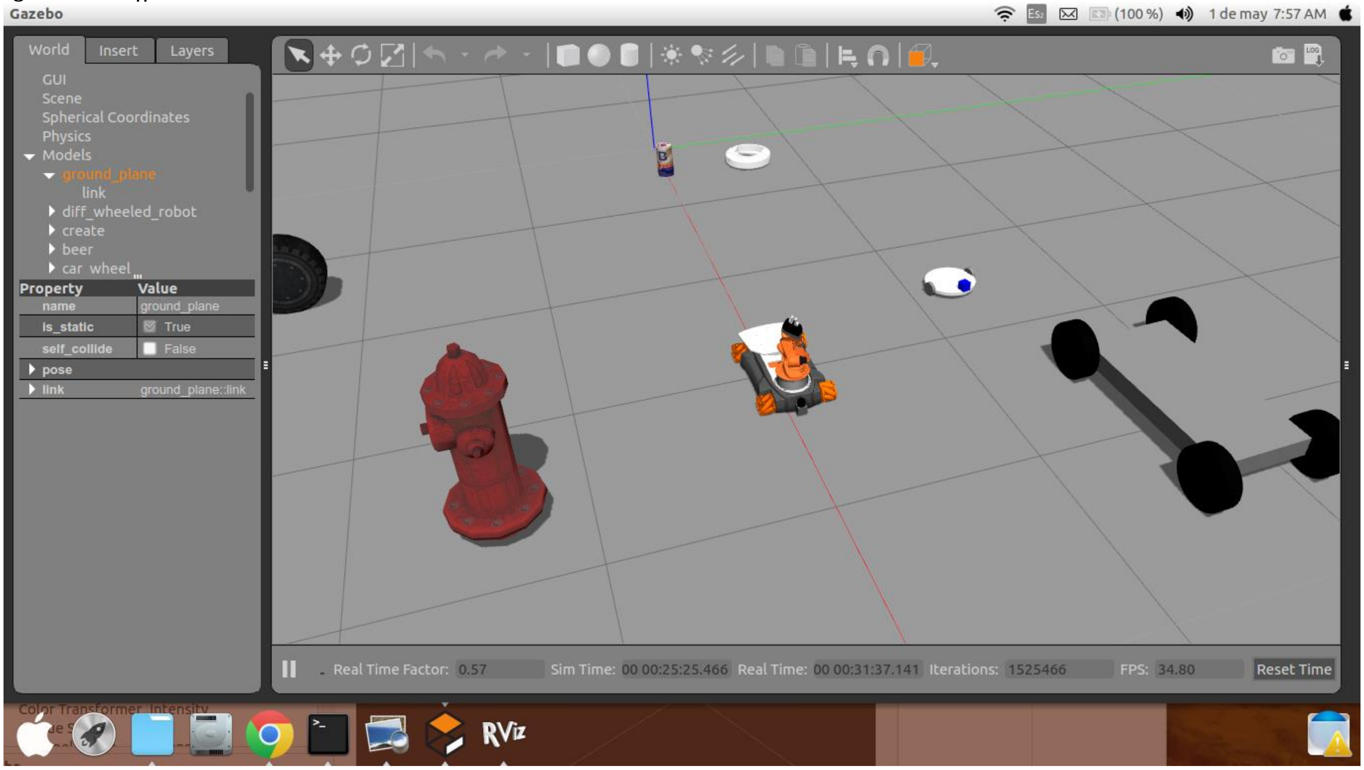
Figure 13 shows the differential mobile robot in gazebo.

Figure 13. Differential wheeled robot in Gazebo.



After successful simulation, let's add the laser scanner to the robot. In the preceding figure, we can see a box on the top of the robot, which is the sensor we added to the xacro. The Gazebo ROS plugin file called libgazebo_ros_laser.so simulate the laser scanner. We can view the laser scanner data by adding some objects in the simulation environment. Here, we add some objetcs around the robot and can see the corresponding laser view.

Figure 14. Differential wheeled robot in Gazebo with obstacles.



The robot we are working with is a differential robot with two wheels, and two caster wheels. The complete characteristics of the robot should model as the Gazebo-ROS plugin for the simulation. In order to move the robot in Gazebo, we should add a Gazebo ROS plugin file called libgazebo_ros_diff_drive.so to get the differential drive behavior in this robot.

The ROS teleop node publishes the ROS Twist command by taking keyboard inputs. From this node, we can generate both linear and angular velocity and there is already a standard teleop node implementation available; we can simply reuse the node.

Launch the Gazebo with complete simulation settings using the following command:

```
$ roslaunch diff_wheeled_robot_gazebo
diff_wheeled_gazebo_full.launch
```

Start the teleop node:

```
$ roslaunch diff_wheeled_robot_control
keyboard_teleop.launch
```

Start RViz to visualize the robot state and laser data:

```
$ rosrun rviz rviz
```

Add Fixed Frame : /odom, add Laser Scan and the topic as /scan to view the laser scan data and add the Robot model to view the robot model. In the teleop terminal, we can use some keys (u, i, *o*, *j*, *k*, *l*, *m*) for direction adjustment. Here is the screenshot showing the robot moving in Gazebo using teleop and its visualization in RViz.

Figure 15. Teleop node for mobile drive differential robot.

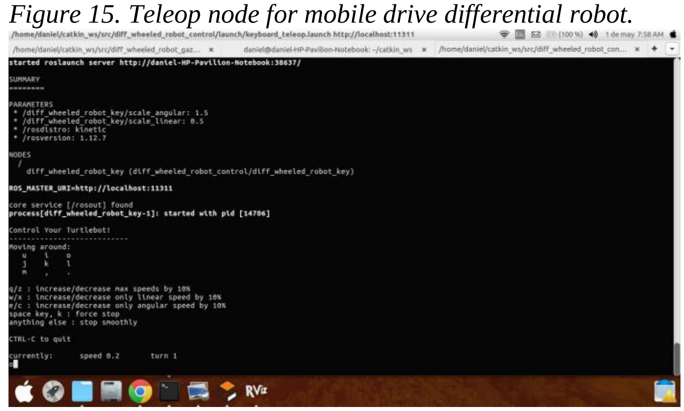
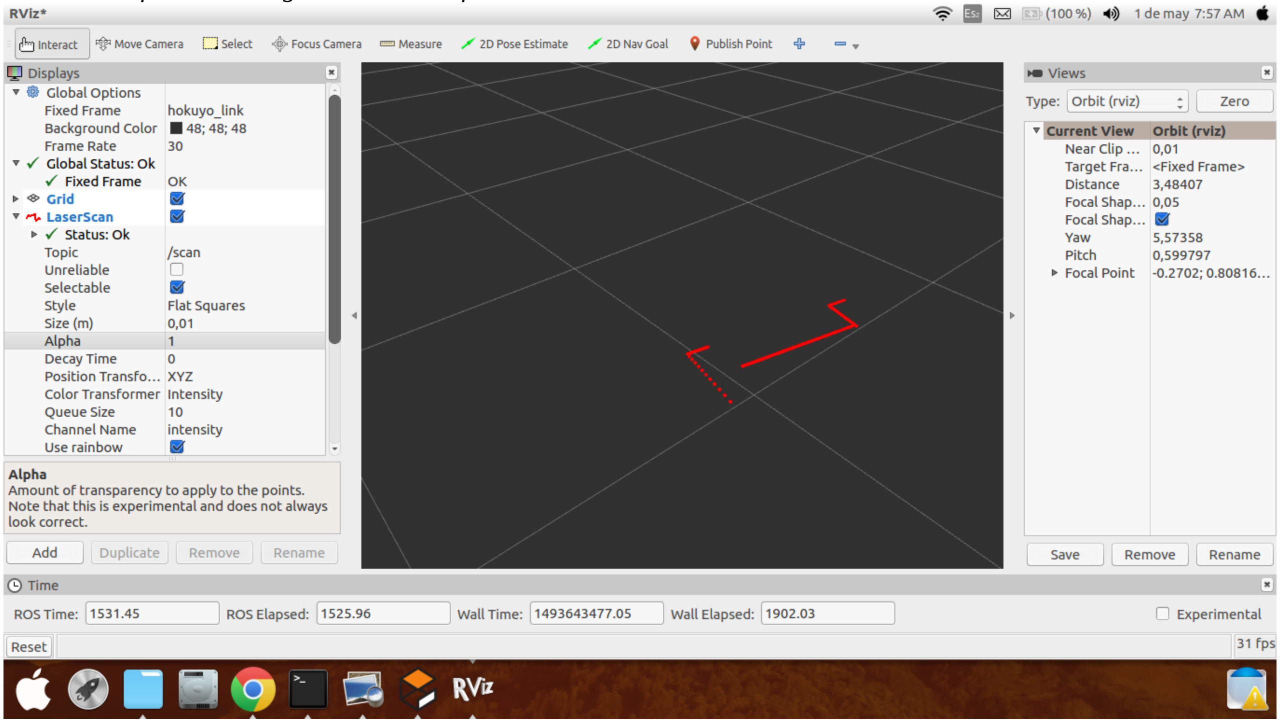


Figure 16. View of Rviz showing the obstacles of the robot in Gazebo.



The robot will only move when we press the appropriate key inside the teleop node terminal. If this terminal is not active, pressing the key will not move the robot. If everything works well, we can explore the area using the robot and visualizing the laser data in RViz.

CONCLUSION

The gazebo simulator is a graphical interface that allows to simulate the behavior of a robot in ROS in an environment with physical variables allowing to have an idea of the operation before arriving at the phase of implementation of the mechanism. It is a very powerful software for the simulation of robots, it allows the real time control of the actuators that manipulate the mechanisms that make up the robot. The mobile robot can be controlled with the node teleop allows to publish a topic with the controls that manipulate the wheels of the robot, its speed and its orientation in the graphic environment of gazebo. The robot manipulator differently, the orientation of the joints of each link and its respective translation is controlled. Finally, it is concluded that the laboratory practice was successfully achieved demonstrating the importance and power of software such as ROS for the implementation, design and manipulation of mobile robots and manipulators

REFERENCES

[1] Letin Joseph, “Design, build, and simulate complex robots using Robot operating system and master its out-of-the-box functionalities,” Mastering ROS for Robotics Programming in *Packt open source.* Birmingham B3 2PB, UK. ISBN 978-1-78355-179-8.

[2] Tobon, Daniel. Working with 3D Robot Modeling in ROS. Mastering ROS for Robotics Programming Laboratory No.1, Universidad Autónoma de Occidente, (2017).