

PROGRAMMING AND SIMULATION OF A ROBOT MODEL USING ROS AND GAZEBO

Bhanu Praharsha Rapelly¹, Divya Vajroju², Shruti Bhargava Choubey³

^{1,2}Department of Electronics and Communication Engineering, Sreenidhi Institute of Science and Technology, 501301, Telangana, India

³Associate Professor, Department of Electronics and Communication Engineering, Sreenidhi Institute of Science and Technology, 501301, Telangana, India

1bhanurapelly@protonmail.com

2divyavajroju37@gmail.com

3shrutibhargava@sreenidhi.edu.in

Abstract – As the planet evolves into a much more technologically advanced world, there is a need for development of much advanced machines to reduce the effort put in by the mankind. The development of machines is a large field of work and to test the optimal working and efficiency of these machines, prototypes are to be modelled and put through a thorough observation. This process requires a lot of raw material for building and creating the environment for the machine to be tested in. An efficient alternative is virtually simulating the machine along with the testing environment. For machines like robots, there are many virtual modelling softwares and simulators available making the testing process much simple. One such alternative is ROS – Robot Operating System. It has all the required tools for modelling, testing and visualizing the robot inside a laptop. ROS comes with a simulator called Gazebo. Inside gazebo, one can create a virtual environment – a world, as gazebo refers it – and deploy robot models for testing. ROS makes it easy to create a robot model using URDF – Unified Robot Description Format – an XML based text program that can model the robot as required. In this paper, creating a custom modelled robot and deploying it in a “world” is discussed.

Keywords – Gazebo, ROS, simulation, URDF, world, XML.

I. INTRODUCTION

ROS is an open-source, meta-operating system for your robot. It provides the services you would expect from an operating system, including hardware abstraction, low-level device control, implementation of commonly-used functionality, message-passing between processes, and package management. ROS is not a real-time framework, though it is possible to integrate ROS with realtime code^[1]. The first commit of ROS code was made to SourceForge on the seventh of November, 2007 by Willow Garage members Eric Berger, Keenan WYROBEK and Scott Hassan. Gazebo was developed in 2002 at the University of Southern California.

II. ROBOT OPERATING SYSTEM

ROS is made to be open source i.e., it intends users to choose configuration of tools and libraries as fit for their need. Hence, there is very little which is core to ROS. In reality, ROS itself is a greater ecosystem consisting of a rich set of tools, a wide range of robot-agnostic capabilities provided by packages. A typical ROS ecosystem consists of the following:

Nodes – A node is a single process running in ROS. Every node has a unique name, with which it registers with the ROS master before taking any action. Multiple nodes can exist at once and interact with each other through ROS master. Nodes are at the center of ROS programming, as most ROS client code is in the form of a ROS node which takes actions based on information received from other nodes, sends information to other nodes, or sends and receives requests for actions to and from other nodes^[2].

Topics - Topics are named buses over which nodes send and receive messages^[3]. One can think of topics as channels which are connected to nodes through ROS master. A node receives or sends information by subscribing or publishing to a topic. The publish/subscribe model is anonymous: no node knows which node is sending/receiving on that topic^[4].

Services - A node may also advertise services. A service represents an action that a node can take which will have a single result. As such, services are often used for actions which have a defined beginning and end, such as capturing a single-frame image, rather than processing velocity commands to a wheel motor or odometer data from a wheel encoder. Nodes advertise services and call services from one another^[5].

Parameter Server - The parameter server is a database shared between nodes which allows for communal access to static or semi-static information. Data which does not change frequently and as such will be infrequently accessed, such as the distance between two fixed points in the environment, or the weight of the robot, are good candidates for storage in the parameter server^[6].

ROS master - The ROS Master provides naming and registration services to the rest of the nodes in the ROS system. It tracks publishers and subscribers to topics as well as services. The role of the Master is to enable individual ROS nodes to locate one another. Once these nodes have located each other they communicate with each other peer-to-peer. The Master also provides the Parameter Server^[7].

The Master is most commonly run using the *roscore* command, which loads the ROS Master along with other essential components.

ROS also has a lot of tools which augment its core functionality. These tools allow developers to visualize, record data ,create scripts and much more^[8]. Some are listed following:

rviz - A three dimensional visualizer used to visualize robots, environments and sensor data. It is highly configurable and one can add plugins to increase its functions.

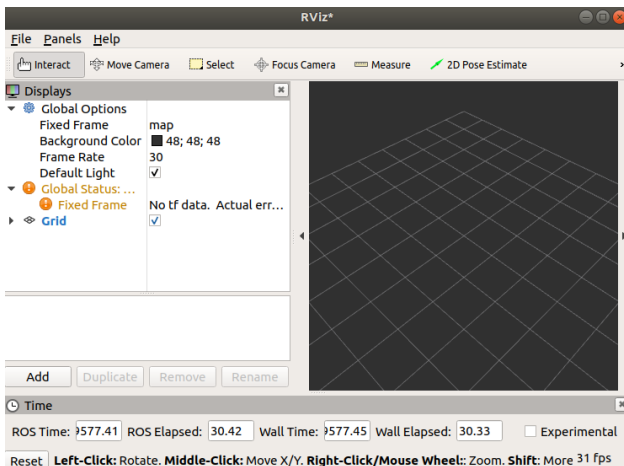


Figure i. Rviz visualizer

Catkin - It is the ros build system. It is based on cmake and is cross-platform, open source and language-independent.

Rosbag - A command line tool used to record and playback message data sent and received by ROS nodes over topics in ROS. *rqt_bag* provides a GUI interface to rosbag.

Rosbash - A package with a suite of tools augmenting the functions of bash shell. Some of the tools are *rosls*, *rosed*, *roscp* which replicate the functionality of

ls, *cd*, *cp* respectively in bash.

Roslaunch - A command line tool to launch multiple ros nodes locally and remotely while setting parameters on the parameter server. Roslaunch configuration files are written in XML and can easily automate a complex startup and configuration into a single command.

III. URDF OF A BASIC BOT

URDF or Unified Robot Description Format is a standard designed for representing a robot model in ROS. It is a package specifically made for ROS and is written in C++^[9]. It requires users to specify the robot model in form of XML code. The code structure is similar to HTML but has user defined tags for representing certain parts of the robot. The code usually begins with a line to specify the XML version being used, followed by the <robot> tag to denote the robot model. </robot> is the end tag for the file. The robot model consists of base link, chassis, joints,links, wheels, sensors etc. A screenshot of a model urdf code of a two wheeled robot is provided below.

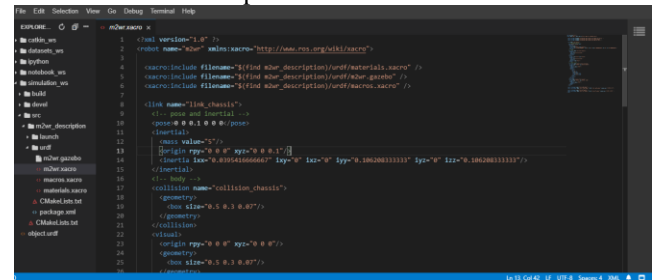


Figure ii. Sample urdf code of a two wheeled robot model

The XML file of robot description should be saved with a .xacro extension. This specifies that the file is an XML file. A simple XML file cannot deploy or launch the robot model into the simulation environment. The XML file is to be placed inside a self-created package inside a workspace created specifically for ROS to operate on. The following steps are to be executed after installing ROS on a Linux machine, to create a workspace and a package, and to launch the robot model^[10]:

1. Open a terminal window
2. Execute the following command:
\$ mkdir -p catkin_ws/src
the above command creates a workspace named catkin_ws and inside that workspace a directory named src is created.
3. Traverse into the src directory:
\$ cd catkin_ws/src
4. Create a catkin package with a name and a dependency set to urdf
\$catkin_create_pkg myrobot_description urdf
5. A new directory along with two files pop up. Go into the newly created package directory and create a new folder with name "urdf"
6. Save the XML file of the robot model into the urdf directory.
\$ mkdir urdf

7. In the same package, create a new folder named “launch” to store the launch files used to deploy the robot model.

```
$ mkdir launch
```

8. Save the launch files into the directory “launch”. Then move back to the workspace directory.

```
$ cd ~/catkin_ws
```

9. Build the package just created by executing the following command.

```
$ catkin_make
```

10. Source the workspace to make ROS interact with the package

```
$ source devel/setup.bash
```

11. Deploy the robot model using roslaunch tool and launch files

```
$ roslaunch myrobot_description rviz.launch
```

IV. LAUNCH FILES

Launch files are XML files with head and tail tags <launch> and </launch>. These files are used to run ROS nodes locally and remotely^[11]. They are saved in the launch directory of a package with a .launch extension. These files automate a complex startup process and remove the need for opening multiple terminal instances at once on the monitor. Launch files can be used to run nodes, change parameters on parameter server and even launch more files from within. A sample launch file is shown below.

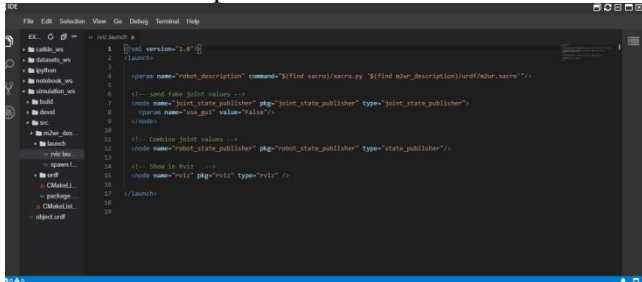


Figure iv. A sample launch file to deploy the robot in rviz

Launch files are a necessary part of the ROS ecosystem as they initiate the ROS master needed to run the nodes. One can modify the launch files to deploy the robot model in the *rviz* or an external simulator like *gazebo*. The content of the launch file changes based on the environment one chooses.

V. GAZEBO

Gazebo is an open-source 3D robotics simulator. Gazebo was a component in the Player Project from 2004 through 2011. Gazebo integrated the ODE physics engine, OpenGL rendering, and support code for sensor simulation and actuator control^{[12][13]}. Gazebo development began in the fall of 2002 at the University of Southern California.

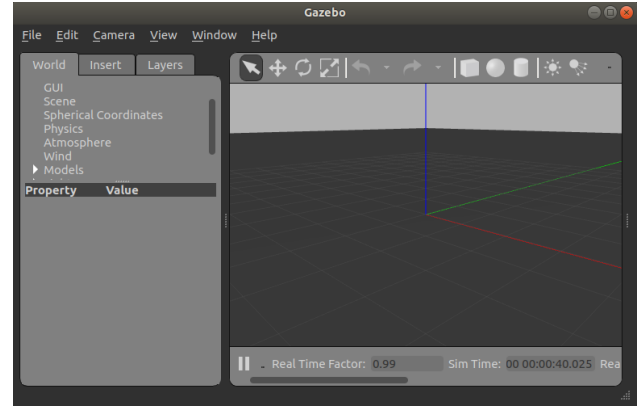


Figure v. Gazebo environment

The original creators were Dr. Andrew Howard and his student Nate Koenig. The concept of a high-fidelity simulator stemmed from the need to simulate robots in outdoor environments under various conditions. As a complementary simulator to Stage, the name Gazebo was chosen as the closest structure to an outdoor stage^[14].

Over the years, Nate continued development of Gazebo while completing his PhD. In 2009, John Hsu, a Senior Research Engineer at Willow, integrated ROS and the PR2 into Gazebo, which has since become one of the primary tools used in the ROS community.

A few years later in the Spring of 2011, Willow Garage started providing financial support for the development of Gazebo. In 2012, Open Source Robotics Foundation (OSRF) spun out of Willow Garage and became the steward of the Gazebo project. After significant development effort by a team of talented individuals, OSRF used Gazebo to run the Virtual Robotics Challenge, a component in the DARPA Robotics Challenge, in July of 2013^[15].

It has Client/Server architecture. The inter-process communication is topic-based Publish/Subscribe^[16].

Gazebo has a native model description format called SDF – Simulation Description Format. The SDF will be able to completely describe the simulated world together with the complete robot model. The process of conversion is handled by adding so called *gazebo-plugins* into URDF file. These plugins can attach into ROS messages and services to create a complete interface between both ROS and Gazebo. The examples for using these *gazebo-plugins* in URDF and SDF files is provided below^[17]:

```
<gazebo>
  <plugin name="differential_drive_controller" \
    filename="libdiffdrive_plugin.so">
    ... plugin parameters ...
  </plugin>
</gazebo>
```

(a) The URDF file

```
<model name="P3DX_robot_model">
  <plugin name="differential_drive_controller" \
    filename="libdiffdrive_plugin.so">
    ... plugin parameters ...
  </plugin>
</model>
```

(b) The SDF file

Figure vi. Using gazebo-plugins

To deploy a robot model into Gazebo simulator, a launch file must be written to start the robot model node and the gazebo server. The below is a screenshot of what a sample launch file for deploying a robot model into gazebo looks like:

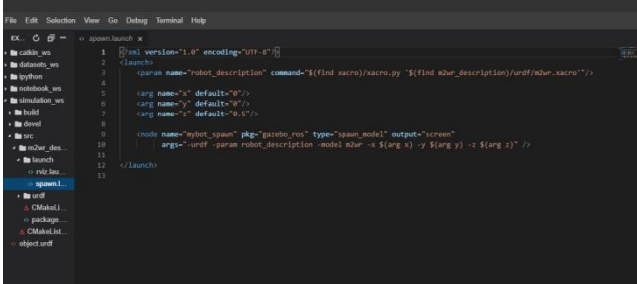


Figure vii. Sample launch file to deploy robot model into gazebo environment

Gazebo has a method to create and customize a virtual environment. These are called worlds. One can create any number of worlds and use them inside a launch file to deploy the world and the robot model inside the world.

VI. RESULTS

Following the above steps, one can create a robot model meeting all the requirements one has. The created robot model can be deployed onto any kind of virtual “world” and the performance and statistical analysis of the robot model can be observed. Changes to the robot model can be applied easily. The modular nature of robot model is very helpful in adding sensors and other components to capture data and process the same data for other components needing it.

The robot model we created and customized for our paper is a two wheeled differential drive robot. The robot has two wheels at the back and one caster wheel in the front of the base link. The world we chose is an empty world without any virtual objects. The screenshot of the robot deployed in the world is provided below:

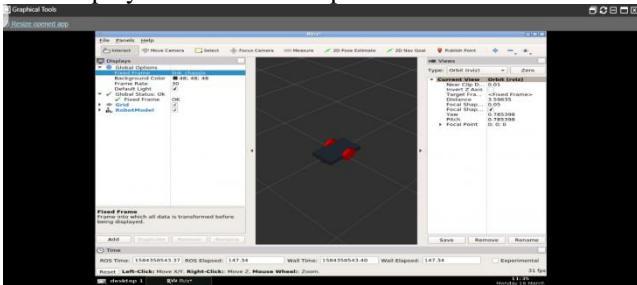


Figure viii. Robot model deployed in gazebo

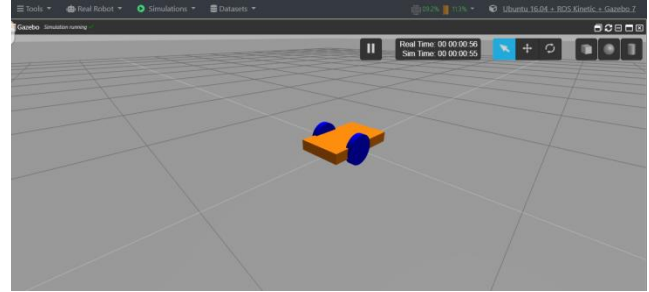


Figure ix. Robot model deployed in rviz

The output observed on the terminal window is also provided below for clear understanding.

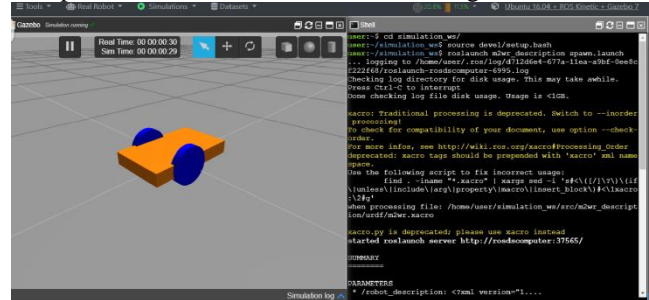


Figure x. Output from the terminal window

References:

- [1] What is ROS? - <http://wiki.ros.org/ROS/Introduction>
- [2] Robot Operating System: Design “Nodes” - https://en.wikipedia.org/wiki/Robot_Operating_System
- [3] "ROS/Tutorials/UnderstandingTopics – ROS Wiki". wiki.ros.org. Retrieved 29 April 2019.
- [4] Robot Operating System: Design “Topics” - https://en.wikipedia.org/wiki/Robot_Operating_System
- [5] ROS/Tutorials/UnderstandingServicesParams – ROS Wiki. wiki.ros.org. Retrieved 29 April 2019.
- [6] ROS/Tutorials/UnderstandingServicesParams – ROS Wiki. wiki.ros.org. Retrieved 29 April 2019.
- [7] Master: Overview - <http://wiki.ros.org/Master>
- [8] Robot Operating System: Tools - https://en.wikipedia.org/wiki/Robot_Operating_System
- [9] <http://wiki.ros.org/urdf>
- [10] <https://www.theconstructsim.com/exploring-ros-2-wheeled-robot-part-01/>
- [11] <http://wiki.ros.org/roslaunch/XML>
- [12] Gazebo simulator - https://en.wikipedia.org/wiki/Gazebo_simulator#cite_ref-1
- [13] "Gazebo". Gazebo Simulator. Archived from the original on 2018-01-16. Retrieved 2019-03-24.
- [14] History – gazebosim.org
- [15] History – gazebosim.org
- [16] Kenta Takaya, Toshinori Asai, Valeri Kroumov and Florentin Smarandache – Simulation Environment for Mobile Robots Testing Using ROS and Gazebo (ICSTCC Oct 2016)
- [17] Gazebo plugins in ROS. (2016) Tutorial: Using Gazebo plugins with ROS. [Online]. Available: <http://gazebosim.org/tutorials?tut=rosgzplugins>