Home Automation using ROS

Srinivas Peri[1], Tejaswini Kallakuri[2], Bhanu Praharsha Rapelly[3], Dr. Shruti Bhargava Choubey[4]

[1,2,3,4]*Department of Electronics and Communication Engineering*

Sreenidhi Institute of Science and Technology, 501301, Telangana, India

Abstract - Home automation is the field of makingsmart homes where all the appliances can communicate with each other and also be operated using a single centralised controller such as using an application in a smart phone or laptop. ROS is one of the best options for making home automation lot more easier ROS – Robot Operating System. It has all the required tools for modelling, testing and visualizing the robot inside a laptop. ROS has an efficient visualisation tool called Rviz(Robot Visualizer)and a simulator called Gazebo. In this paper, we would like to put forward some details regarding ROS and how ROS can be used for home automation.

I. INTRODUCTION

In general, ROS consists of code and tools that helps to run project code and do the required job—including the infrastructure for running it, like messages passing between processes. ROS is designed to be a loosely coupled system where a process is called a node and every node should be responsible for one task. Nodes communicate with each other using messages passing called topics. Each node can send or get data from the other node using the publish/subscribe model. The primary goal of ROS is to support code reuse in robotics research and development so you can find a built-in package system. ROS is not an OS, a library, or an RTOS. It's a framework using the concept of an OS.

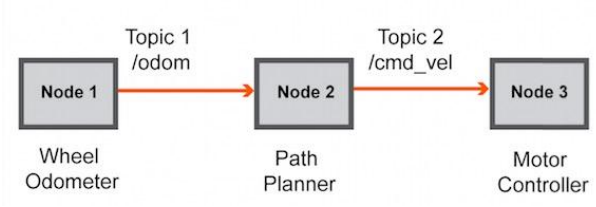

Figure i. Communication between nodes

ROS supports machines that run Linux with Ubuntu or Debiandistro. It has a lot of releases with names ordered alphabetically.

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[].

Topics - Topics are named buses over which nodes send and receive messages[]. Onc 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[].

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[].

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[].

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[].

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[]. 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.

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.

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++[]. 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.

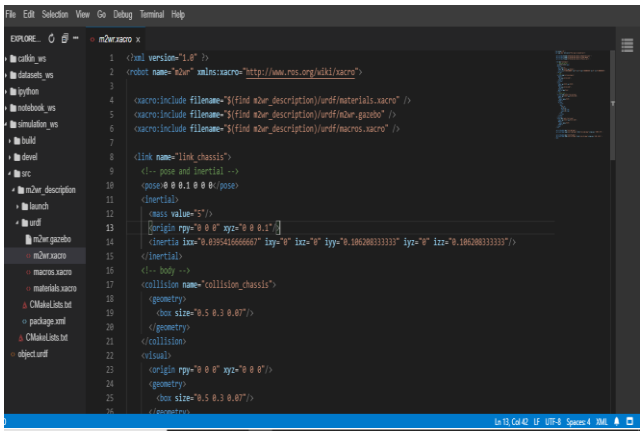

Figure ii. Sample urdf code of a two wheeled robot model

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[]:

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:

$ cdcatkin_ws/src

4. Create a catkin package with a name and a dependency set to urdf

$ catkin_create_pkgmyrobot_descriptionurdf

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”

$ mkdirurdf

6. Save the XML file of the robot model into the urdf directory.

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

$ sourcedevel/setup.bash

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

$ roslaunchmyrobot_descriptionrviz.launch

IV. LAUNCH FILES

Launch files are XML files with head and tail tags <launch> and </launch>. These files are used to run ROSnodes locally and remotely[]. 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.

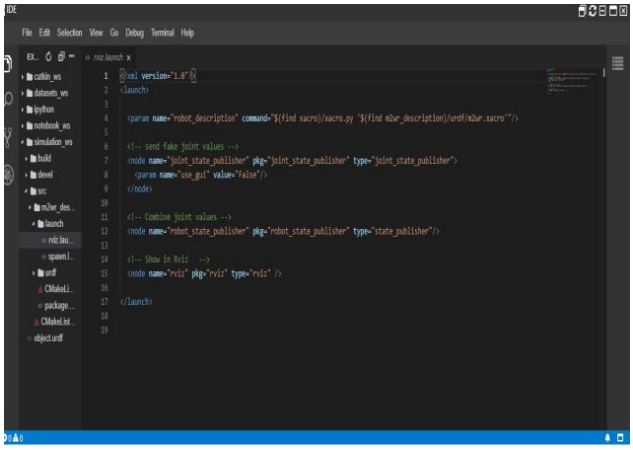

Figure iii. A sample launch file to deploy the robot in *rviz*

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 deply 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. RVIZ

RVIZ is a ROS graphical interface that allows you to visualize a lot of information, using plugins for many kinds of available topics.

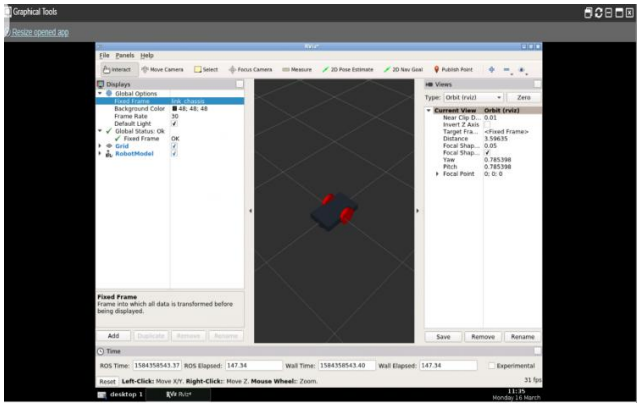

Figure iv. visualize robot model in rviz

VI. GAZEBO

Gazebo is a 3D simulator, while ROS serves as the interface for the robot. Combining both results in a powerful robot simulator. With Gazebo you are able to create a 3D scenario on your computer with robots, obstacles and many other objects. Originally Gazebo was designed to evaluate algorithms for robots.

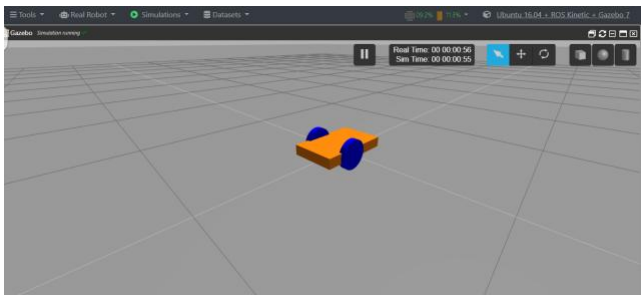

Figure v. Robot model simulated in gazebo

Controlling the robot using command

The package called teleop_twist_keyboard allows the robot to be operated remotely by the command:

rosrunteleop_twist_keyboard teleop_twist_keyboard.py

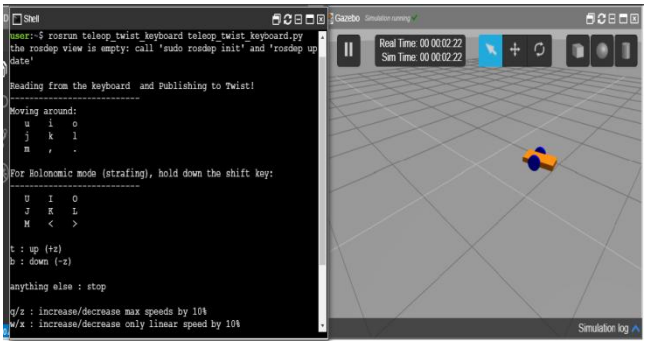

Fig vi. Controlling bot using teleop_twist command

VII. 2-D Navigation

To monitor and command the robot from remote computer you have two main options: one is using a graphical remote access program such as teleop_twist_keyboard in gazebo and the other option is using a command-line remote connection such as **SSH**.

Running ROS across multiple machines and Network Setup is a better way.

First, You will need to install the robotican meta-package on the remote machine by assuming that the remote computer and the robot are connected to the same network and have that the robot IP is **10.0.0.14** and the remote computer IP is **10.0.0.13(For example)**.

VIII. RESULTS AND CONCLUSION

ROS therefore can be used by communicating with sensors and other devices by using publishers and subscribers. The ROS control can be done by using a laptop by using SSH as metioned above.Therefore using ROS makes it easy to build home automation systems. The ease to build home automation systems would create more applications for home automation. ROS could help to build home assistant robots and home cleaning at a more cheaper costs as the current cost to buy a home cleaning robots are very expensive.

REFERENCES

[1] https://www.google.com/search?q=2d+navigation+remote+access+using+ROS&source=lmns&bih=632&biw=1242&rlz=1C1AZAA_enIN857IN857&hl=en&ved=2ahUKEwjUlNHO-qPoAhWc7jgGHQg3BroQ_AUoAHoECAEQAA

[2] http://wiki.ros.org/rviz/Tutorials

[3] https://www.researchgate.net/publication/320782369_Mapping_of_unknown_industrial_plant_using_ROS-based_navigation_mobile_robot

[4] http://www.cs.binghamton.edu/~szhang/teaching/18spring/reports/Luo-Ma-Zhou.pdf

[5] https://iopscience.iop.org/article/10.1088/1757-899X/257/1/012088

[6] https://zenodo.org/record/2670059/files/%2814-18%29Autonomous%20Navigation%20with%20Collision%20Avoidance%20using%20ROS-format.pdf

[7] http://wiki.ros.org/navigation

[8] http://wiki.ros.org/navigation/Tutorials/Using%20rviz%20with%20the%20Navigation%20Stack