Robot Operating System

Report written by:

- Shivam Laddha
- Asit Gautam

Affiliated with The LNM Institute of Information Technology,Jaipur, Rajasthan.

E-mail addresses :

- 19uec111@lnmiit.ac.in
- 19uec112@lnmiit.ac.in

Abstract

This paper discusses ROS, an open-source robot operating system. For starters, ROS is a structured communications layer that sits on top of the host operating systems of a heterogeneous compute cluster, rather than an operating system in the traditional sense of process management and scheduling. In this paper, we examine how ROS interacts with different robot software frameworks and provide a fast overview of some of the most popular ROS-based applications. Basically our aim is to solve the problem of V2V (Vehicle to Vehicle) which is still in process. There is a scenario in which four to five vehicles are crossing the paths of each other. In this situation the bots solve and communicate with each other and solve the problem . In the individual level if we talk about a particular bot then we tested and tried two possibilities . In the first possible situation we can control the bot's direction and speed with our keyboard and in other situations, we can fix the path and object the bot can manage its direction and speed with its own. This possibility can be discussed further in this report in greater detail.

● Introduction

It's difficult to write software for robots, especially as the scale and breadth of robotics develops. It's tough to reuse code since different types of robots have such distinct hardware. Furthermore, the sheer scope of the needed programming, which must contain a deep stack beginning with driver-level software and continuing to perception, abstract reasoning, and beyond, may be overwhelming. The needed breadth of expertise is well beyond the capabilities of any single researcher, hence robotics software architectures must also support large-scale software integration operations.

Several robotics researchers, including ourselves, have built a variety of frameworks to handle complexity and facilitate rapid software development for experiments, culminating in the multiple robotic software systems. It is now in use in both academia and industry. Each of these frameworks was designed for a specific reason, such as to address perceived problems in current frameworks or to stress key aspects of the design process.

Throughout the development of ROS, the framework outlined in this paper, compromises and prioritizations were made. We believe that as robotic systems become more complicated, the center's emphasis on large-scale integrative robotics research will be advantageous in a number of circumstances. In this article, we examine ROS' design goals, how our implementation achieves them, and how ROS handles a number of popular robotics software development use cases.

● Area of Work

In this project, we work on an advancement of a Single Bot to do a further advancement, for this we mainly focus on a software named Gazebo which is used in ubuntu. In this software all the activities of the bot can be seen in 2D image. Now, for better understanding we move to the next level. Rviz is used to see the bot in the 3D image. So, we can see all important parameters. Firstly, the packages have been made as shown in the hierarchy figure 1. The packages of the robot are made in CPP and by combining the files, and by testing this file in the gazebo with the help of the plugins and after that we move towards Rviz the bot has been built.

```
ros_pkg
├── action
│   └── demo.action
├── CMakeLists.txt
├── include
│   └── ros_pkg
│       └── demo.h
├── msg
│   └── message.msg
├── src
│   └── demo.cpp
└── srv
    └── service.srv
```

Figure 1. Hierarchy to how the packages has been created

- Tools Used

All the ROS development is done in UBUNTU because it is open source and has many wide applications. For understanding the basics of ROS, we follow the ros wiki. It's a free website and all the steps are given in a line wise order. So, firstly we learn to operate ROS by giving commands in the terminal, which is used in ubuntu. So, by the commands we learn a Turtlesim. It’s basically a package from which a turtle bot is present and it can be controlled by the keyboard by giving the command of “twist teleop keyboard”. It's shown in figure 1.

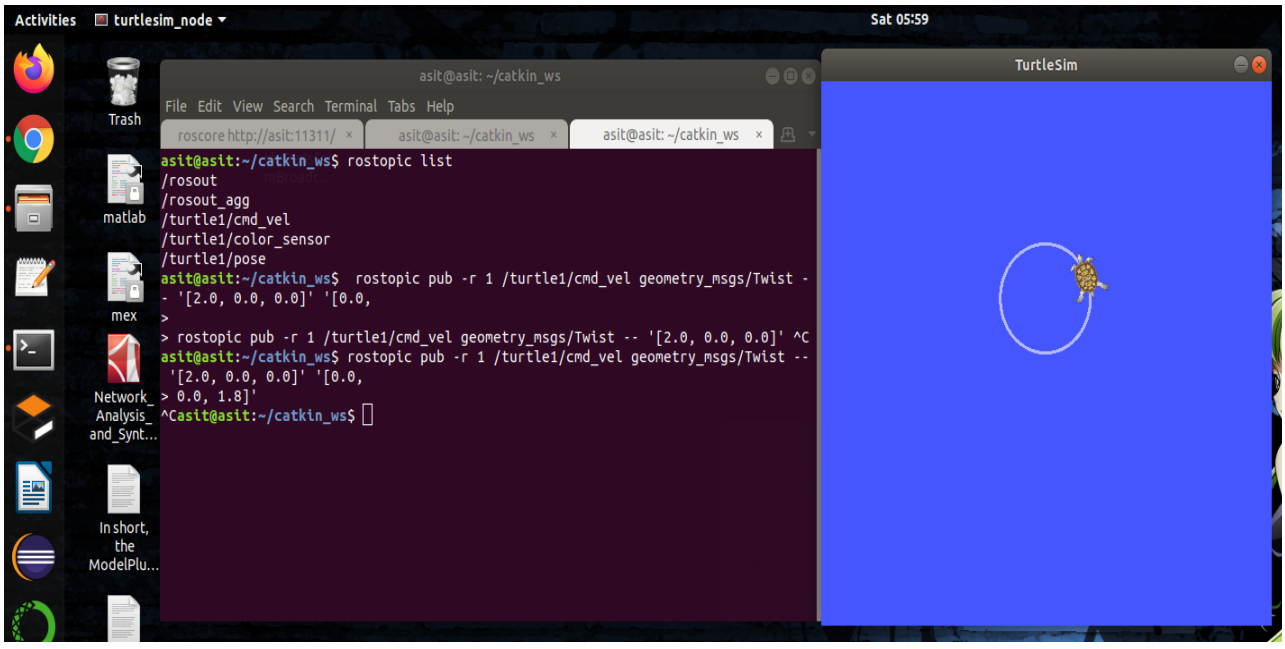

Figure 2. Turtle Sim

After understanding all the packages of the turtlesim like speed of the turtle bot, directions, how messages are getting transferred. This is only a package to understand the basics of the ROS. In actuality Gazebo is used to train a bot. It's shown in figure 2.

Gazebo - is a free and open-source 3D robotics simulator. It includes the ODE physics engine, OpenGL visuals, as well as support code for sensor simulation and actuator control.ROS communicates with gazebo using gazebo executables.Main executables includes gzserver which is responsible for physics and sensor data generation and gzclient which is responsible for visualization.gzserver reads URDF to generate model and world, gzserver and gzclient communicates to each other through gazebo communication libraries.Plugins are loaded by gzserver or gzclient depending upon physics or custom GUI.Libraries for gazebo system includes physics library(collision shapes,joints), rendering library(lightning texture), sensor generation(implements sensors),GUI libraries.Theselibraries supports plugin.These plugins provides user with access to the respective libraries without using the communication system,gazebo plugin directly interacts to C++ classes(or libraries).

This knowledge will help in understanding communication between gazebo and ROS.

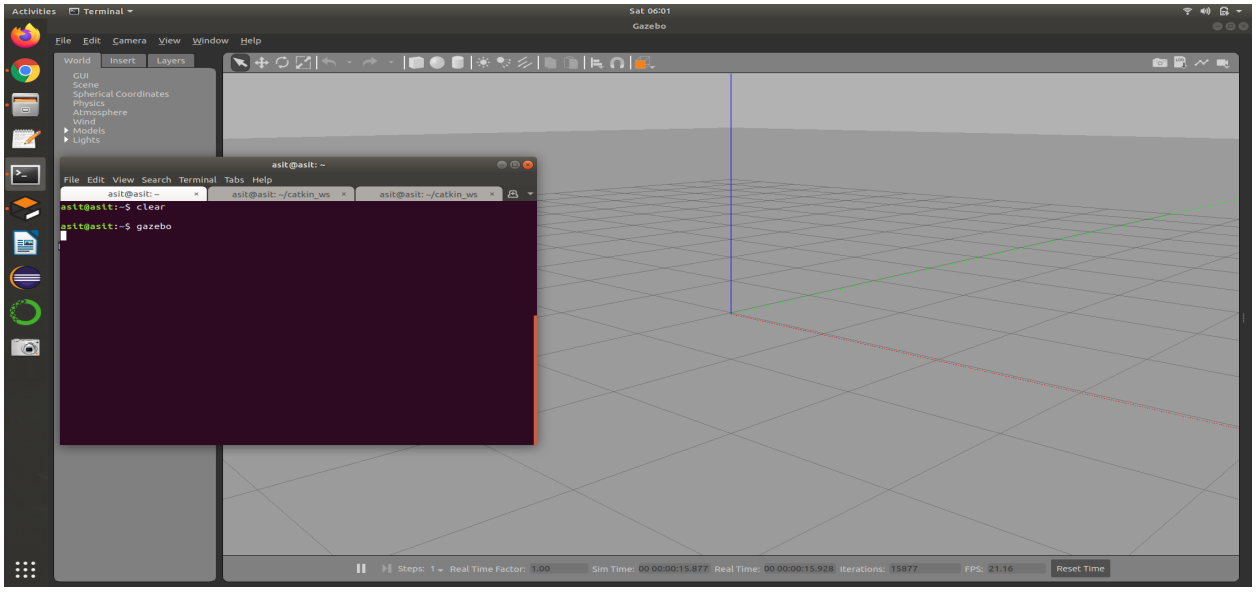

Figure 3. Gazebo

From the terminal if we wanna go to the gazebo then simply we put a gazebo command in the terminal and then the gazebo will open as the interface is shown in figure 2. In the gazebo there are various bot models present like car, ambulance ,truck ,etc. We can train our bot here.

From the Terminal if we want to go to Rviz then we can board a rosrun Rviz.

Rviz- is basically used to see the bot or a design in 3D.It is a visualizer which can subscribe to rostopics and can visualize the data graphically,these data can be transformed frames or laser data or any other sensor.

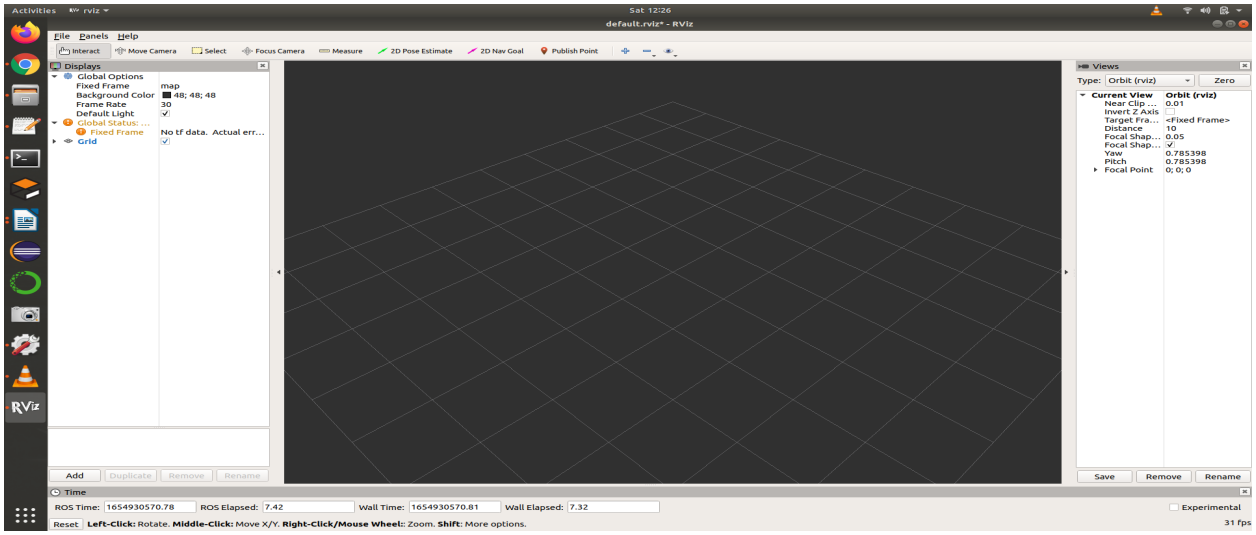

Figure 4. Rviz

Before gazebo or Rviz there are some of the basic steps which we have to take in counts or in layman language we have to activate them.

- Rosmaster : It's a Roscore. It's necessary for nodes to register at startup with the master for communication between nodes.
- Rostopic : Nodes communicate with each other over topics. Nodes can publish or subscribe for topics. It's a name for a message stream.
- Rosnode : Nodes which are made in ROS.
- Rostopic List : It's a list of topics which is present in a particular package.
- Catkin Build : for updating the environment we use this tool.CAtkin packages are built by cmake,cmake build, test and package software.Cmake is invoked to build the catkin packages present in src-cmakelist.txt and is kept in build space.
- Launching node: Ros Launch package name with a file name,this will launch a file in XMC file.
- Gazebo Node: A ROS node that integratesROS message passing with gazebo's integral message passing.

1.2 Methodology

The task is being accomplished using ROS,RViz and Gazebo .Two tasks are being accomplished here.

1.2.1 Keyboard controlled bot

In this project keyboard sends the messages ,or particularly publishes the messages over rostopic **/joy_teleop/cmd_vel** then this message is transferred to **/husky_velocity_controller/cmd_vel** through rosnode **/twist_mux** and gazebo node subscribes to this topic and convert it to show the movements in the simulation.Using RViz also we can show the transformations of the bot by subscribing to the topic responsible for state of bot.

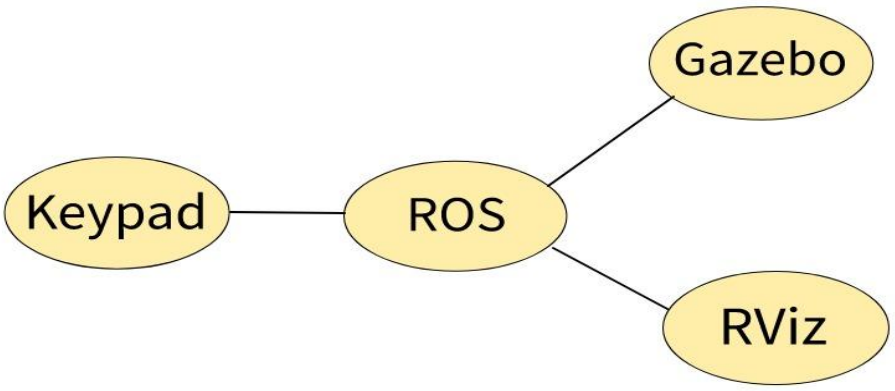

Figure 5. Method or Raw Structure for Keyboard controlled Bot

Launch file-

```
<?xml version="1.0"?>
<launch>
<include file="$(find husky_gazebo)/launch/husky_empty_world.launch">
</include>
<node name="teleop" pkg="teleop_twist_keyboard" type="teleop_twist_keyboard.py"
output="screen"/>
<node name="husky"pkg="husky_highlevel_controller" type="husky_highlevel_cont
roller">
<rosparam command="load" file="$(find husky_highlevel_controller)/config/par.yaml"/>
</node>
</launch>
```

The above launch file launches the two nodes husky and teleop, now these are two independent nodes where the first one is responsible for the bot and its joints and the second one is responsible for the conversion of keypad keys into appropriate velocities both linear and angular.

The message published over the topics will be of type **twist** whose definition contains two vector3 classes containing linear and angular variables each around 3 axis x,y and z.

So these messages are what the gazebo receives from ROS and gazebo do the actuation using its physics engine -ODE showing us the simulation, gazebo interacts with ros using gazebo node,gazebo node is a ROS node that integrates ROS message passing with gazebo's internal message(using gzserver and gzclient) passing with messages on rostopic like **/link_states** and /**model_states**.

1.2.3 Line following bot

This project contains an autonomous bot which follows the yellow line path using visuals.

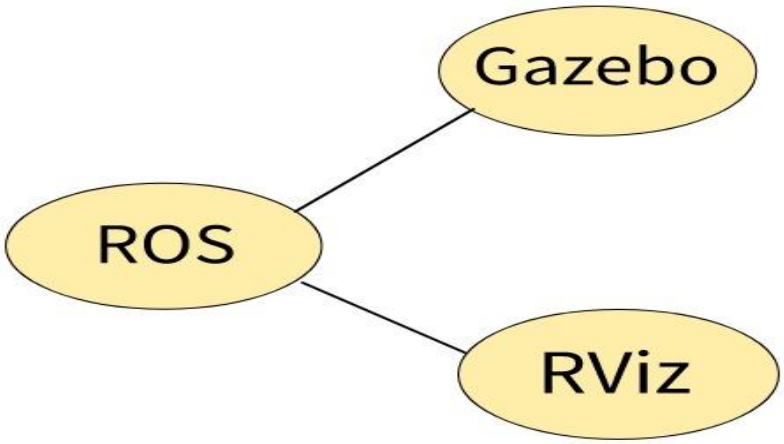

Figure 6. Method or Raw Structure for Line following Bot

Customized URDF file is created which can be used to build up a robot and show in ros but for the same URDF file to work in Gazebo some special tags are to be added.

To spawn a model in gazebo with ros we have to use spawn_model which makes a service call request to gazebo node to add a custom URDF in gazebo.In the above mention URDF file all The physics parameters are mentioned which can be easily changed according to practical use,same is true for the bot dimensions also.To contro

l the model and sense the environment in the gazebo using ros there are two options available first **ros_control** and **transmission tags** and second using **gazebo plugins**.Gazebo plugins are more convenient so in this project gazebo plugins are used namely **Skid_steer_drive_controller** and **camera_controller** .Gazebo-ros plugins are just a C++ code and its work is to publish or subscribe gazebo to the ros topics.
The Skid steer drive controller plugin sends a velocity command to the single rear wheel and single front steer of a steering drive wheel base, which is then separated. Odometry is calculated and reported based on the feedback.
Camera controller publishes the CameraInfo and Image ROS messages as described in sensor msgs to provide a ROS interface for mimicking cameras like the wge100 camera.

Now subscribing and publishing to these messages(over rostopics) the task of following the line is achieved.
These are the main components of the launch file -

```
<param name="robot_description" command="$(find xacro)/xacro --inorder $(find
 first_one)/urdf/descr.urdf.xacro"/>

<node name="spawn_model" pkg="gazebo_ros" type="spawn_model" output="screen"
args="-urdf -param robot_description -model car"/>
<node name="rviz" pkg="rviz" type="rviz" required="true"/>
<node name="line_follower" pkg="first_one" type="script.py" output="screen"/>
```

Now the main node to inspect is the line_follower.This is the node which takes the input of sensor image by subscribing to topic **/cam/image** and then applying CV algorithms on it to publish linear and angular velocity message on **/cmd_vel** with message type as twist.The algorithms finds the moment of the yellow color(Moments signify the distribution of matter about a point or an axis. In OpenCV, moments are the average of the intensities of an image's pixels.) visible in a frame and then find the error between screen center and the moment coordinates and accordingly publishes the velocity.Gazebo plugin receives the velocity and actuate it to show the simulation.

line_follower node looks like a normal python code only but it requires some extra chunk of code as mentioned below.

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

from sensor_msgs.msg import Image
from geometry_msgs.msg import Twist

rospy.init_node('line_follower')
FIlower=follower()
rospy.spin()
```

Track is loaded by using world description file and using image file of the path as a material in it.

1.3 Result

Keyboard controlled bot-

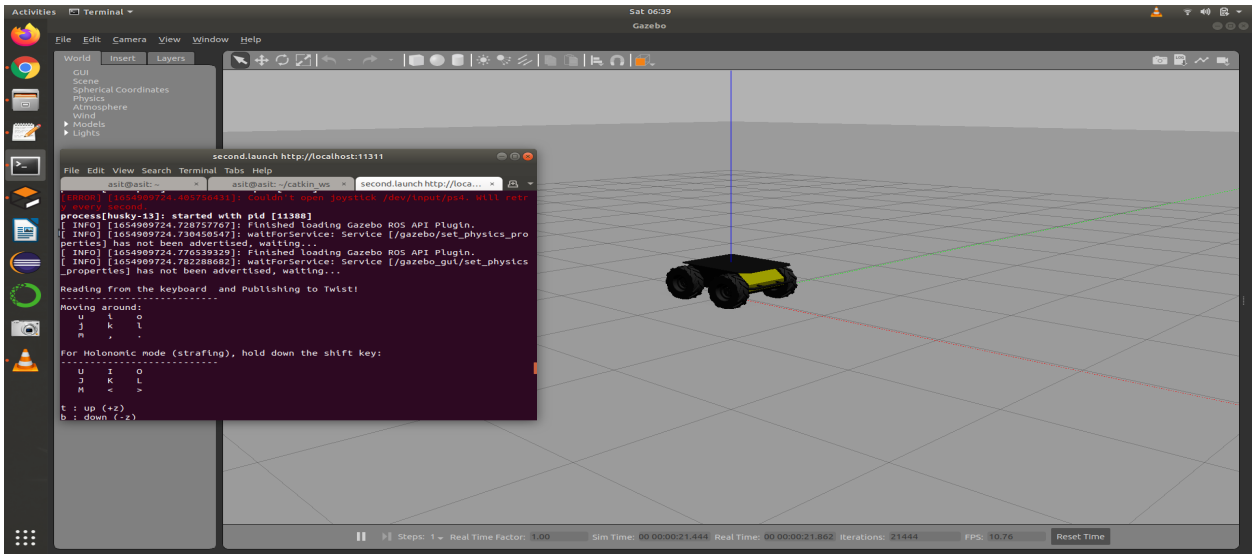

Figure 7. Husky Robot following the command given by keyboard

This husky bot is controlled by the system keyboard whose speed and directions bot can be changed by the controller.

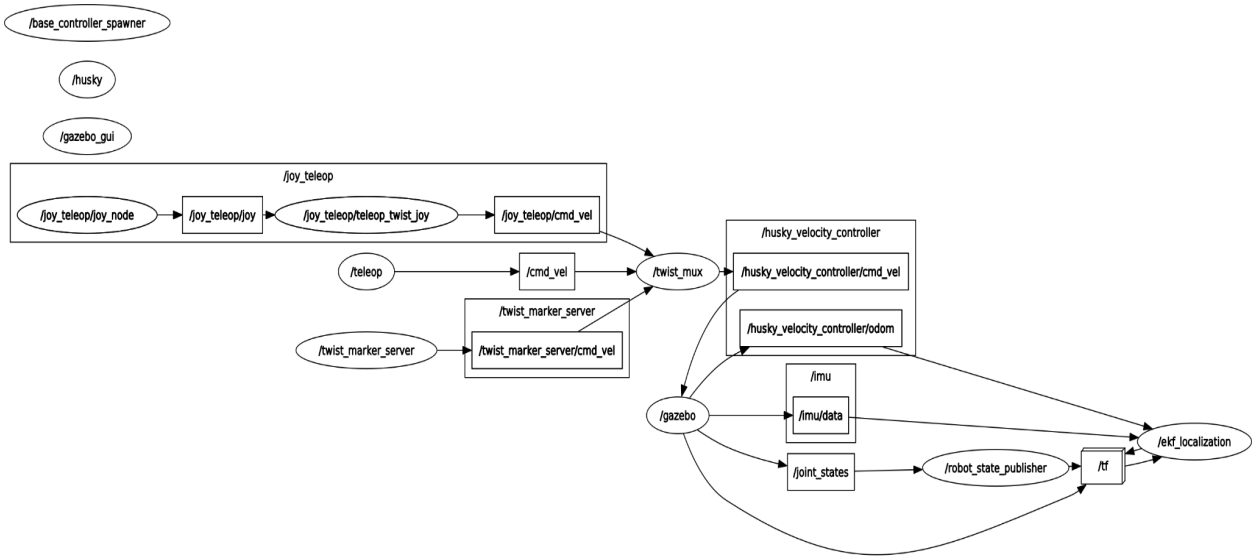

Figure 8. RQT of a Keyboard controlled Bot

Figure 8 shows the ROS nodes and topics and how they are interconnected. It is a complete representation of the communication network behind the simulation. /base_controller_spawner, /gazebo_gui ,/husky are the nodes which are not connected as these are only publishing to /rosout and subscribing to /clock ,these are permanently present internal topics hence not mentioned in rqt graphs.
/joy_teleop/joy_node and /joy_teleop/teleop_twist_joy works together to publish over topic /joy_teleop/cmd_vel the twist message which contains the converted form of the keypads pressed.This topic is then subscribed by /twist_mux node.This node receives velocity from three nodes at a time,as all are active but that doesn't mean all are publishing also, only /cmd_vel topic is getting published.The message by /twist_mux is published to /husky_velocity_controller/cmd_vel which is subscribed by /gazebo node.Now /gazebo node handles all the communication between gazebo and ros, so the velocity(twist message) is received by /gazebo and it publishes the transformed(as programmed like in tf message that is transformed frames) data to /imu,/joint_states,/robot_state_publisher node./robot_state_publisher publishes the transformation frames to /tf.Now the rest of the communication happens internally in gazebo by different internal libraries and gzserver and gzclient which are not a part of ROS so is not shown in RQT graph.

ROS topics list	ROS nodes list
-/clock /cmd_vel /diagnostics /e_stop /gazebo/link_states /gazebo/model_states /gazebo/parameter_descriptions /gazebo/parameter_updates /gazebo/set_link_state /gazebo/set_model_state /husky_velocity_controller/cmd_vel /husky_velocity_controller/odom	/base_controller_spawner /ekf_localization /gazebo /gazebo_gui /husky /joy_teleop/joy_node /joy_teleop/teleop_twist_joy /robot_state_publisher /rosout /teleop /twist_marker_server /twist_mux

/husky_velocity_controller/parameter_descriptions /husky_velocity_controller/parameter_updates /imu/data /imu/data/accel/parameter_descriptions /imu/data/accel/parameter_updates /imu/data/bias /imu/data/rate/parameter_descriptions /imu/data/rate/parameter_updates /imu/data/yaw/parameter_descriptions /imu/data/yaw/parameter_updates /joint_states /joy_teleop/cmd_vel /joy_teleop/joy /joy_teleop/joy/set_feedback /navsat/fix /navsat/fix/position/parameter_descriptions /navsat/fix/position/parameter_updates /navsat/fix/status/parameter_descriptions /navsat/fix/status/parameter_updates /navsat/fix/velocity/parameter_descriptions /navsat/fix/velocity/parameter_updates /navsat/vel /odometry/filtered /rosout /rosout_agg /scan /set_pose /tf /tf_static /twist_marker_server/cmd_vel /twist_marker_server/feedback /twist_marker_server/update /twist_marker_server/update_full	

Table 1. Commands for Communication Keyboard controlled Bot

Line following bot

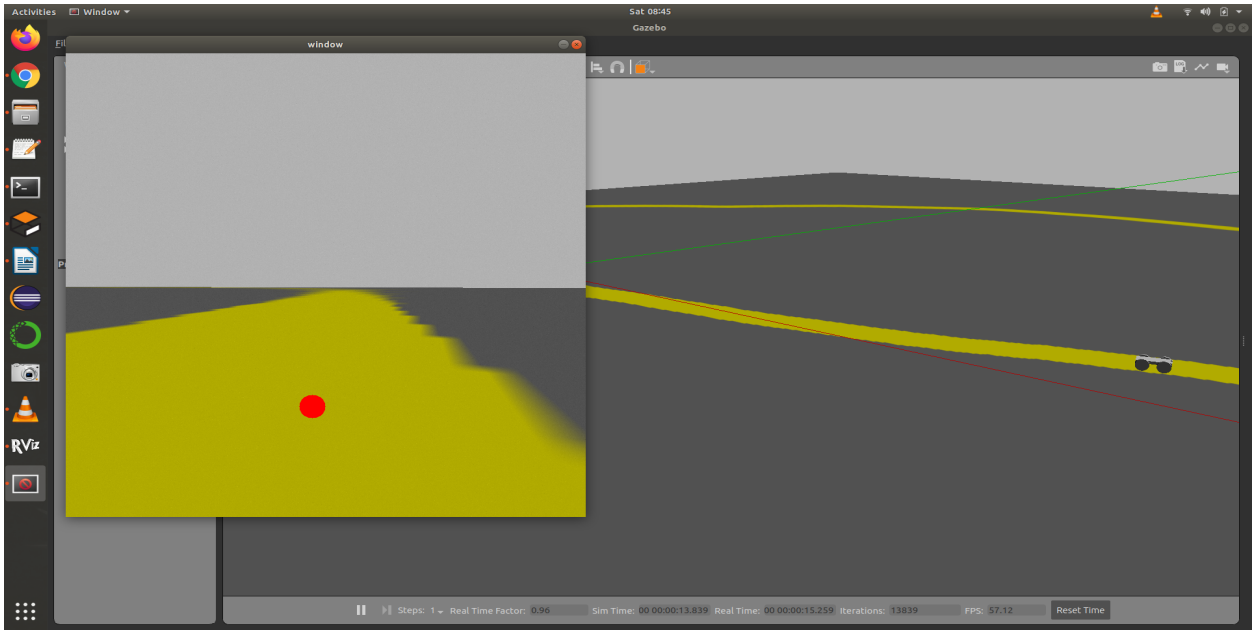

Figure 9. View from the robot

Small screen is the camera view that is the environment sensed by the gazebo plugin and getting published over ros topic with red dot showing the moment or center of pixels of yellow color.

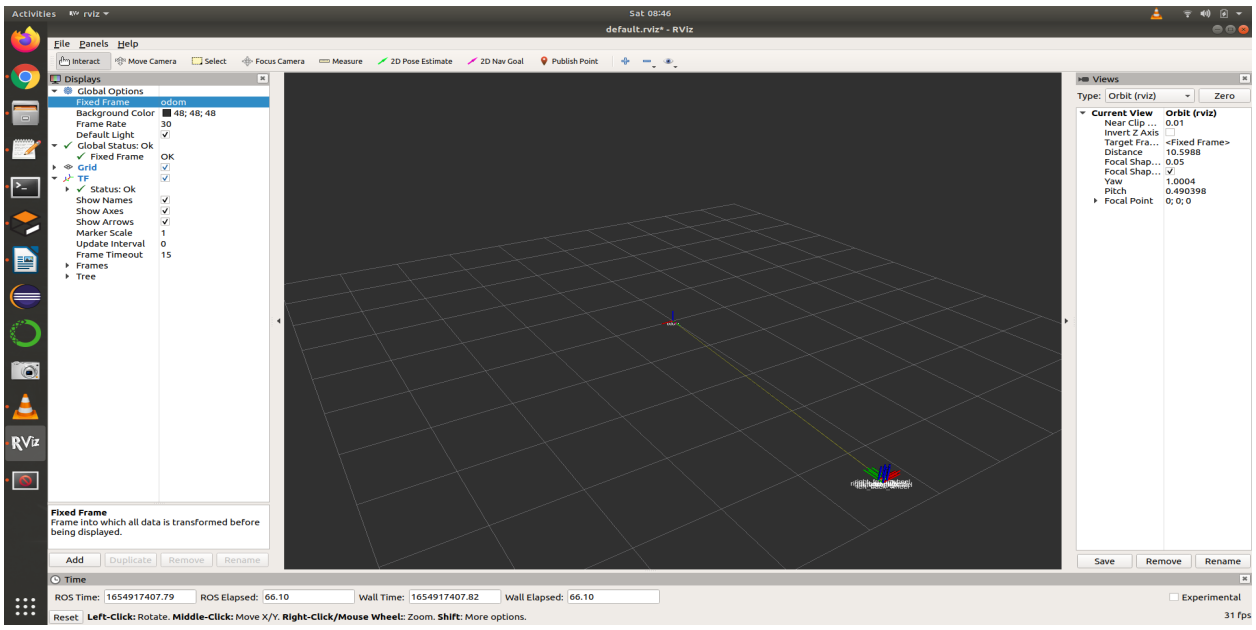

Figure 10. Environment

Subscribing to appropriate ROS topics we can visualize the odometry,point clouds,laser scan,place marker etc.

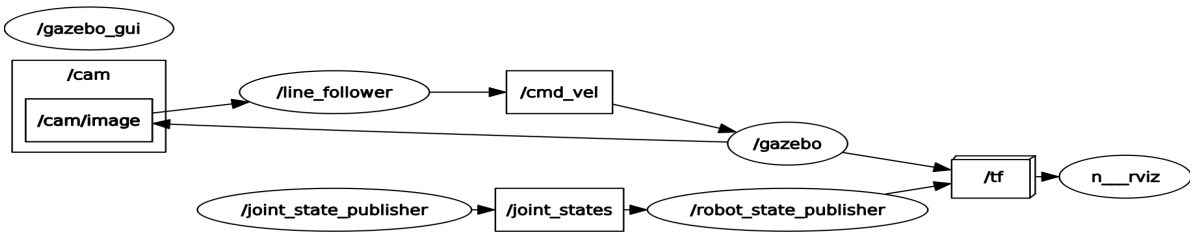

Figure 11. RQT Graph of Line following Robot

Entire communication and nodes responsible for the simulation are above showing in fig 11 how /gazebo_gui node is connected internally and is publishing to topic /rosout with no subscription so not represented by rqt graph. /cam/image is subscribed by /line_follower node which depending on the received data publishes velocity on /cmd_vel,now again this topic is subscribed by /gazebo which publishes tf frames on /tf now rviz node is also running and we have it subscribed to /tf to visualize the change in bot frames with time.This /gazebo node is the one which is responsible for the communication between gazebo and ros so it receives image message from gazebo internally(gazebo libraries) and publishes it to /cam/image topic./joint_state_publisher is a node responsible for publishing the states however this work is done by /gazebo here as /joint_state_publisher is not subscribed to any information or message.

ROS topics list	ROS nodes list
/cam/camera_info /cam/image /cam/image/compressed /cam/image/compressed/parameter_descriptions /cam/image/compressed/parameter_updates /cam/image/compressedDepth /cam/image/compressedDepth/parameter_descriptions /cam/image/compressedDepth/parameter_updates	/gazebo /gazebo_gui /joint_state_publisher /line_follower /robot_state_publisher /rosout /rviz

/cam/image/theora /cam/image/theora/parameter_descriptions /cam/image/theora/parameter_updates /cam/parameter_descriptions /cam/parameter_updates /clicked_point /clock /cmd_vel /gazebo/link_states /gazebo/model_states /gazebo/parameter_descriptions /gazebo/parameter_updates /gazebo/set_link_state /gazebo/set_model_state /initialpose /joint_states /move_base_simple/goal /odom /rosout /rosout_agg /tf /tf_static	

Table 2. Command for communication of Line following Bot

1.4 Conclusion

Our main focus was communication.These are the steps towards our final goal of creating a V2V simulation. One of the greatest advantages of this project is that before solving the actual problem we can simulate it and find the errors like the bot car. We can customize according to practical things like using actual weight and dimensions of the car and specifying friction values according to practical and actual measurements to create a simulation of reality.

Acknowledgements: I would like to extend my deepest gratitude to Professor Abhishek Sharma, who helped our team to go in a particular direction because ROS is

itself a very big branch and to find the direction in it is very difficult. The meeting helps us to get our project completed to this stage and we have to further continue this project under the guidance of Professor Abhishek Sharma.

Bibliography and References

[1] https://rsl.ethz.ch/education-students/lectures/ros.html

[2] https://wiki.ros.org/Documentation

[3] https://www.youtube.com/channel/UCh3TLV-vQzzcWGQ4u2jsMOw/playlists

[4]https://www.theconstructsim.com/intro-to-robot-programming-ros-learning-path/

[5] https://www.cell.com/current-biology/pdf/S0960-9822(12)01451-0.pdf

[6] J. Kramer and M. Scheutz, “Development environments for au- tonomous mobile robots: A survey,” *Autonomous Robots*, vol. 22, no. 2, pp. 101–132, 2007.

[7] **Robinson, M.M.** Presentation/Webinar - Open Source Software for Robotics - When to use it and when to use closed source, Virtual Event, March 16, 2022.

[8] **Armstrong, L.** Presentation - The hardening of ROS 2 for industrial robotics applications: Key technology & impact. AWS Cloud Robotics Summit, August 18-19, 2020

[9] **Ripperger, M.A., Robinson, M.M.** - Development of an Agile Platform for Aerospace Applications Leveraging Open-Source Tools, In-Person Presentation, November 17, 2021.