DRISHTI MINI PROJECT
TELEOPERATION ON CAR IN ROS

MENTORS

PRAKHAR DUBEY
PRABHAKAR JAISWAL
PRANAV PREMLANI
NAVEEN RAWAT
PRAJWAL SONAWANE

MEMBERS

TIRTH JAIN
GHANISTHA SINGHAL
CHINMAYANANDA SAHU
AISHWARY MEHTA
SANIYA KURESHI
CHAITANYA RATHOD
ANIKET MOTWANI
SMITH PATEL

Acknowledgement

It gives immense pleasure in bringing out this synopsis of the project entitled "Tele-operation of car in ROS"

Firstly we would like to thank our mentors who gave us their valuable suggestions and guided us throughout the project when needed. They encouraged us to work on this project to complete in time.

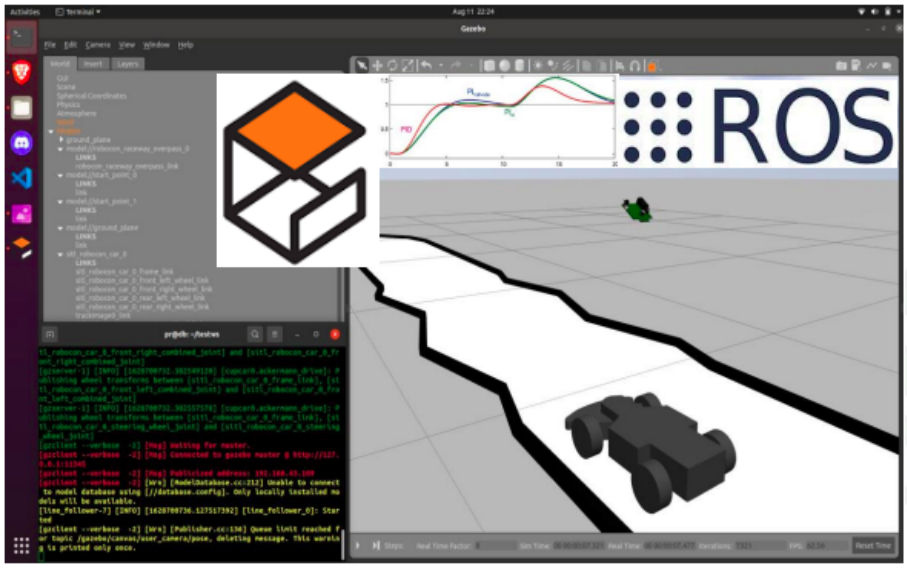
We are grateful to all of those with whom we have had the pleasure to work during this and other related projects. Each of the team members has shown a great team spirit towards the project and learn a lot from each other and enjoy working together

Abstract

This paper presents to simulate a car model in Gazebo usingROS and control the motion of the car using teleoperations.In Teleoperation,It indicates operation of a system of a system or machine at a distance.The user can control the car by pressing keys on keyboard.Here we used ROS and Gazebo to simulate our car.ROS(Robot Operating System) which used in the Linux environment to set the communication and Gazebo is an open-source 3D robotics simulator which will help us to monitor our model.

Contents

1 INTRODUCTIONS



1.1 Abstract

How would you test your algorithms for Robot navigation without a physical robot? The simulation platform Gazebo and ROS framework helps in solving such problems. The aim of this project is to simulate a car model in Gazebo usingROS and control the motion of the car using teleoperations.

2 Installation: UBUNTU 20.04LTS

Process: DUAL BOOTED Software download Link:https://ubuntu.com
BIOS:(UEFI)
YouTube link to be followed How to Dual Boot Ubuntu 20.04 LTS and Windows 10
https://www.youtube.com/watch?v=-iSAyiicyQY||*AStepbyStepTutorial*||[2021]−*UEFILinux*−*YouTube*

2.1 Installation process

2.1.1 Requirements

You'll need to consider the following before starting the installation:

- Ensure you have at least 25 GB of free storage space, or 5 GB for a minimal installation.
- Have access to either a DVD or a USB flash drive containing the version of Ubuntu you want to install.
- Make sure you have a recent backup of your data. While it's unlikely that anything will go wrong, you can never be too prepared.

2.1.2 Boot from USB flash drive

- After downloading the Ubuntu 20.4LTS ISO image, make a bootable USB drive by formatting it using software and burn the ISO image to the USB drive.
- Make partitions and allocate free space for ubuntu in drive.
- and restart the pc and open the boot menu then select the bootable drive.

2.1.3 Prepare to install Ubuntu

- Select normal installation and install third party software. then continue.

2.1.4 Allocate drive space

- Choose 'something else' and choose the free space partition made initially.
- Select ext4 journaling file then mount point as " /" then continue.

2.1.5 Begin installation

2.1.6 Select your time zone

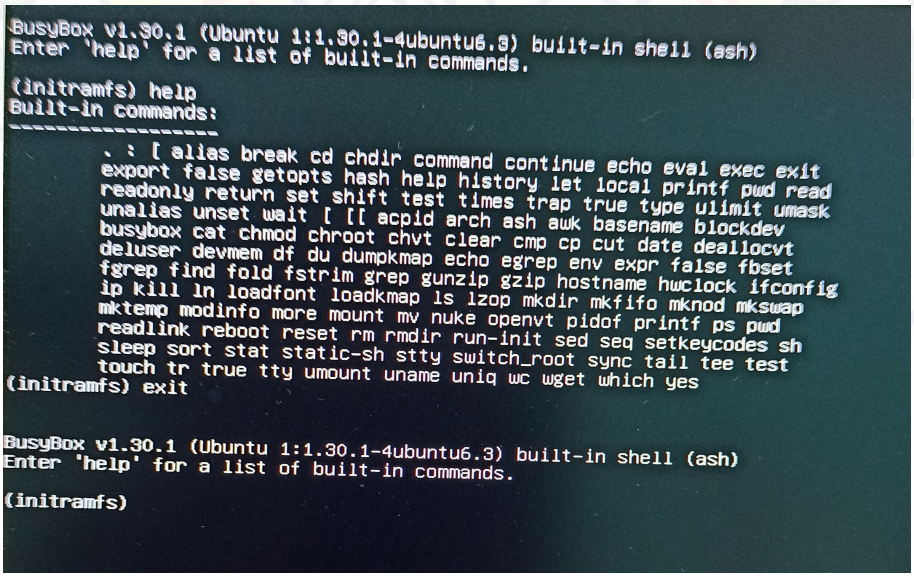
2.1.7 Login details

2.1.8 Installation complete

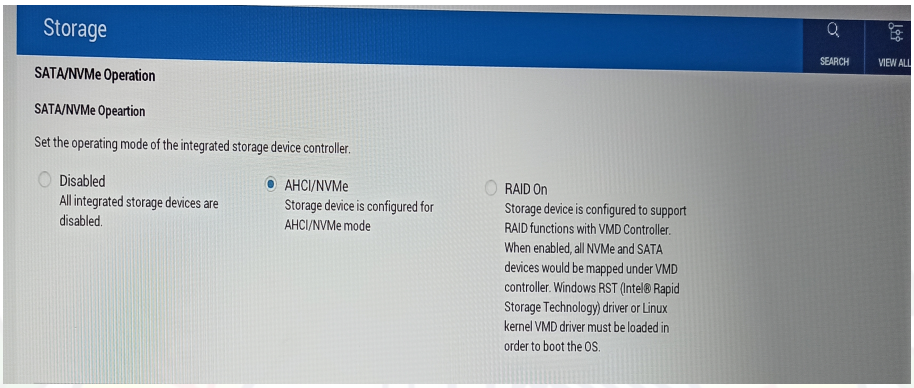
- Restart the PC, remove USB drive,GRUB will appear then select Ubuntu.

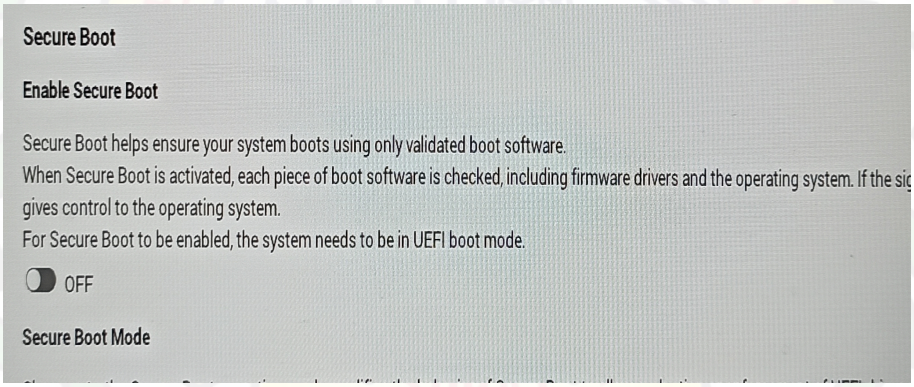
2.2 ERRORS SOLUTIONS

2.2.1 After booting Ubuntu a busy box command appeared.



Solution:

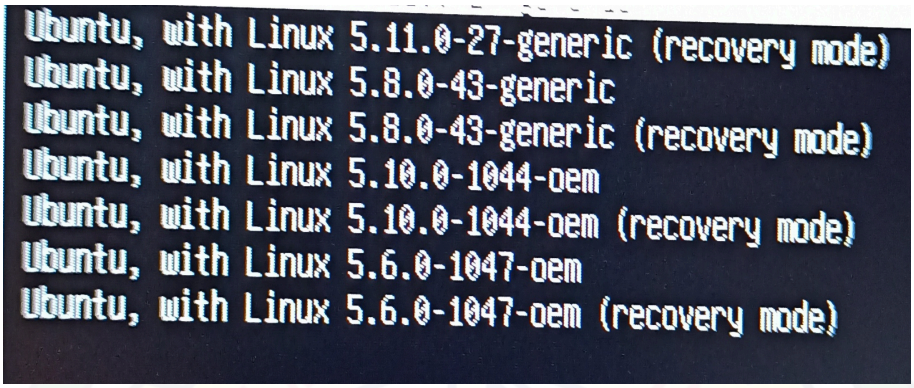




Disable secure boot from BIOS and from storage option from BIOS disable RAID On and enable AHCI/NVme.After this restart the pc and problem solved.(DELL laptop users)

2.2.2 WIFI issue in Ubuntu

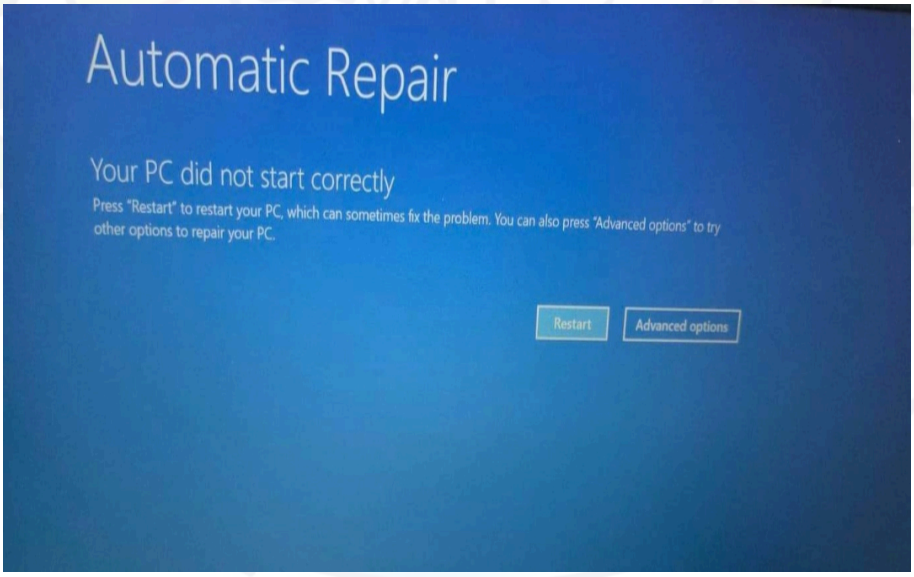
From the advanced option for Ubuntu in GRUB I selected 5.10.0-1044-oem kernel and made it default .where the WIFI driver was installed.in the current version wireless drivers were missing.



2.2.3 Window bitlocker encryption

Window didn't open after dual boot. Due to a hardware change my C and D drive was locked. And I was unaware of bitlocker but I recovered it from my Microsoft account later on.

2.2.4 Automatic repair problem in loop



Restart in loop appeared, but it wasn't solved after many trials. Finally, I reset my PC and re-installed W10 from this device.

2.2.5 Problem in opening boot

During the Ubuntu downloading process , after allocating memory to ubuntu i restarted my laptop to open ubuntu then i tried all functions key but i was not able to open the boot menu. Finally I found a youtube video for this problem,and i was able to open the boot menu by pressing the left side small key on my lenovo laptop and after i proceed further.

Reference link-https://www.youtube.com/watch?v=f_9_Fg6H0HI

2.2.6 NOTE

I faced one problem in installation. Problem was I follow full video without hearing carefully so I merged memory of ubuntu with E drive of pc which did in Video for uninstallation process so my Ubuntu memory deleted so I installed this one more time and finally Ubuntu downloaded.

3 ROS2 package installation

Reference link:
https://docs.ros.org/en/foxy/Installation/Ubuntu-Install-Binary.html

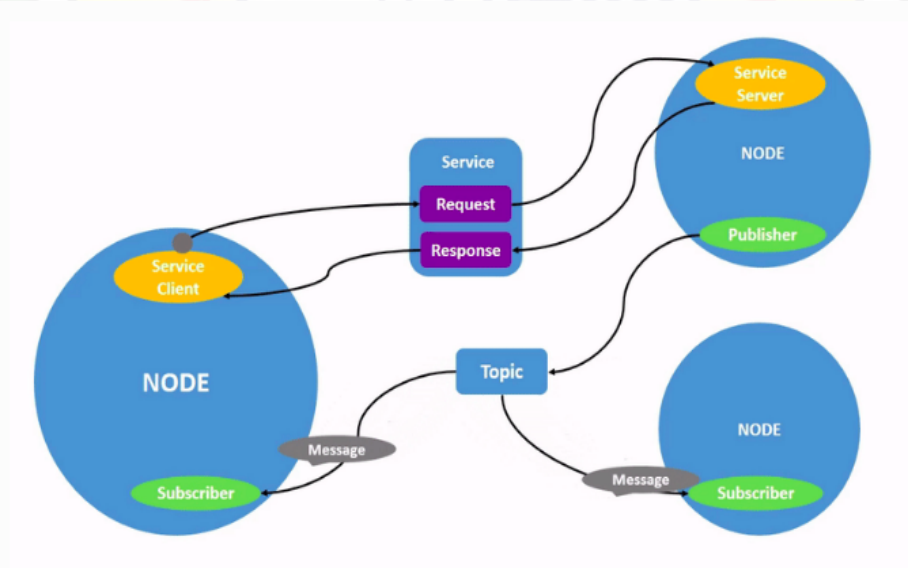
3.1 Types of ROS simulators-Galactic

Foxy
Rolling
Noetic
Melodic
Ardent
Bouncy
Crystal
Eloquent
Dashing
Indigo

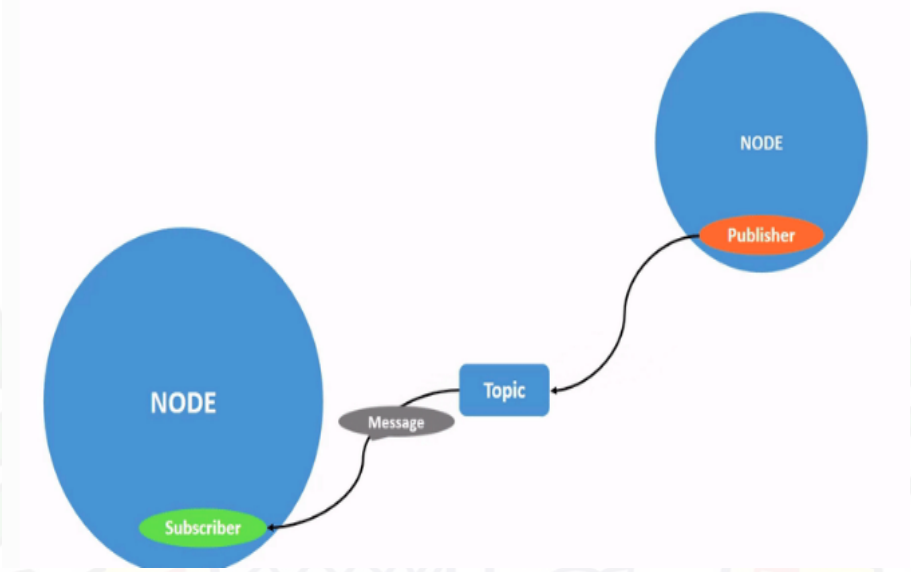
4 Understanding ROS 2 nodes:

Reference link:
https://m.youtube.com/watch?v=aeOS9xqblrglist=PLRE44FoOoKf7NzWwxt3W2taZ7BiWyfhCpind

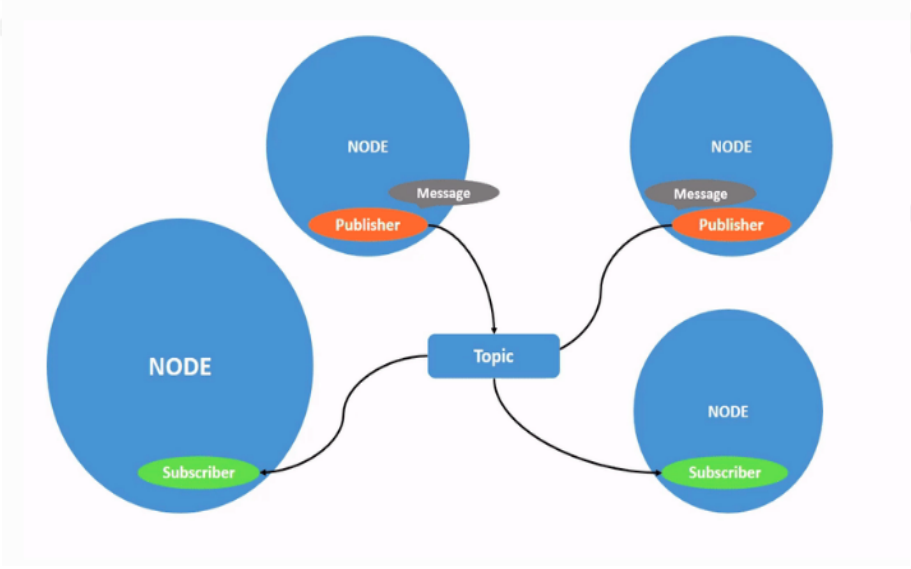
Each node in ROS should be responsible for a single module purpose (e.g. one node for controlling wheel motors, one node for controlling a laser range-finder, etc). Each node can send and receive data to other nodes via topics, services, actions, or parameters.
A full robotic system consists of many nodes working in concert. In ROS 2, a single executable (C++ program, Python program, etc.) can contain one or more nodes.



5 Understanding ROS 2 topics:



A node may publish data to any number of topics and simultaneously have subscriptions to any number of topics.
Topics are one of the main ways in which data is moved between nodes and therefore between different parts of the system.



6 Oops in python

Reference link:

https://youtube.com/watch?v=qiSCMNBIP2gfeature=share

https://youtube.com/watch?v=gfDE2a7MKjAfeature=sharet=29318

https://youtu.be/HfmFcj0NmHI

7 Problem Statements-

7.1 Creating Workspace

workspace:

https://m.youtube.com/watch?v=aeOS9xqblrglist=PLRE44FoOoKf7NzWwxt3W2taZ7BiWyfhCpind

Workspace is a directory containing ROS 2 packages. Before using ROS 2, it's necessary to source your ROS 2 installation workspace in the terminal you plan to work in. This makes ROS 2's packages available for you to use in that terminal.

Run in terminal

mkdir -p /trying/src
cd trying
cd src
ros2 pkg create –build-type ament_python py_pubsub

Publisher nodes publish data to topic(s) and subscriber nodes receive data from topic(s).

7.2 Basics of publisher subscriber code

7.2.1 Publisher code basics

Here we create a class called "MinimalPublisherAA". In this class two functions are created. The __init__ function creates a publisher.

Ex "self.publisherAA_ = self.create_publisher(Float64, 'topicb', 10)". Here the data type can be Int32, Int64, Float32, Float64 and String. And the topic name can be changed too. In order to use any of this data type we need to import them using this command "from std_msgs.msg import Float64"

"msg.data = your data" which is going to be transferring the data from publisher to subscriber via the respective topic.

"rclpy.spin()" is working like an infinite loop to call the class.

7.2.2 Subscriber code basics-

The code is very similar to that of the publisher but basically in __init__ function it creates a subscriber.

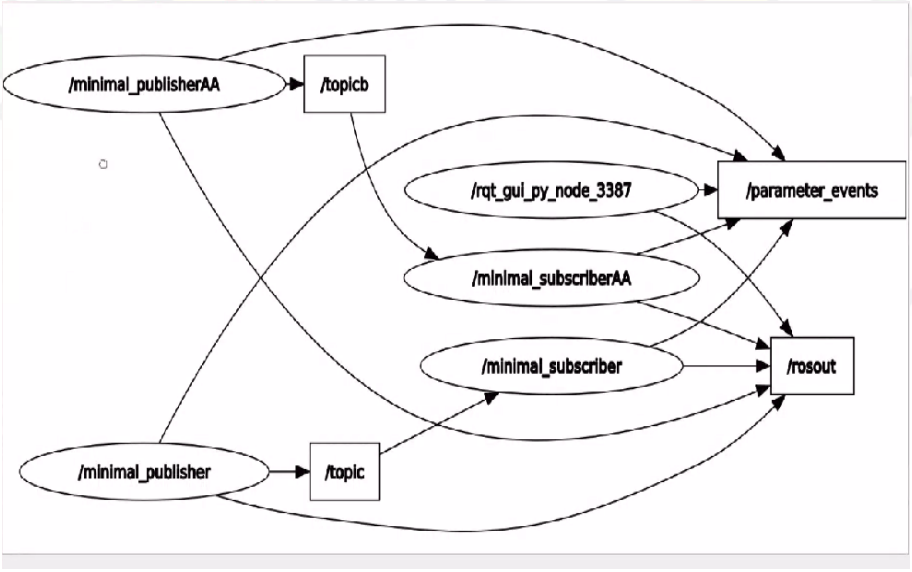
7.3 Problems(given and errors we faced)

7.3.1 In single publisher and subscriber(Problem 1)

Common error faced was that we didn't included this lines in the package.xml file:
$< exec\_depend > rclpy < /exec\_depend >$
$< exec\_depend > std\_msgs < /exec\_depend >$

7.3.2 Two publisher subscriber problem(Problem2)

Basic idea of the code we created with ROS graph(command to be written in terminal "rcqt_graph")
Publisher 1————————Topic1————————- subscriber1
Publisher 2————————Topic2————————subscriber2



Publisher1 and subscriber2 in one single _.py file and Publisher2 and subscriber1

in other _.py files

Publisher1 and subscriber2 they should print together in the terminal one after the other in infinite loop.

7.3.3 Two publisher subscriber problem with fibonacci series (Problem3)

One of the publishers should be publishing fibonacci series and the topics for both the pairs should be different.

Here we use an instance variable in one of the publishers to create the fibonacci series.

8 Gazebo installation

8.1 Download this documentation file

https://drive.google.com/file/d/1OSyDqvhxaql4De3Ve77An1VqKaN$_s$*PSK*/*view*?*usp* = *sharing*

Then copy it into src of created workspace

8.2 command after creating packages

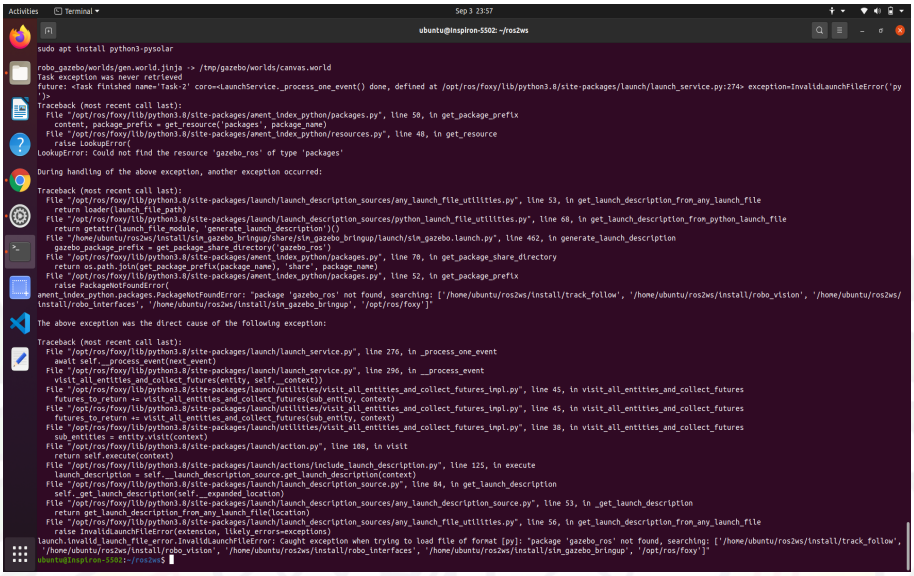
colcon build –packages-select sim_gazebo_bringup –symlink-install

Ssh-keygen

sudo apt install xclip git

xclip − *selclip* <∼ /.*ssh*/*id_rsa.pub*
Afterrunningabovecodesinyourterminalthengotoyourgithubaccountandfollowbelowsteps :
Yourgithubacoount > *settings* > *SSHandGPGkeys* > *clickonSHSkeyspressCTRL*+
V > *SAVEnowfollowvideowhichplacedinthisdrive* : *https* : //*drive.google.com*/*drive*/*folders*/1*RC*
sharing

8.3 Error while installing gazebo



Solution of this error by downloading this file:

https://firebasestorage.googleapis.com/v0/b/gitbook-28427.appspot.com/o/assets%2F-MWeiLnwrIKZJWLFsXhf%2F-MXcfaeFblmznqth$_0$$fk\%2F-MXcgSRrBPzq5-K6O-C5\%2Ffoxy_install_aim.sh?alt = mediatoken = 2adf3a55 - 8463 - 4ff1 - 890e - 67b6d32fe747$

Run in terminal for solving error:
$\sim /Downloads$

$chmod\ a + x\ foxy\_install\_aim.sh$

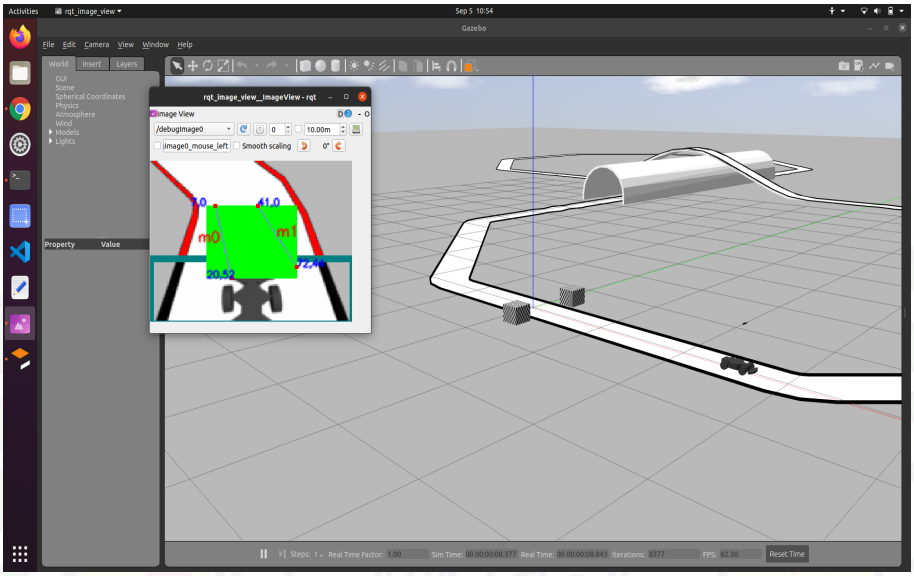
$./foxy\_install\_aim.sh$

$\sim /ros2ws$

$echo"source/home/\$USER/ros2ws/install/setup.bash" \gg \sim /.bashrc$

$source \sim /.bashrc$

ros2 launch sim_gazebo_bringup sim_gazebo.launch.py



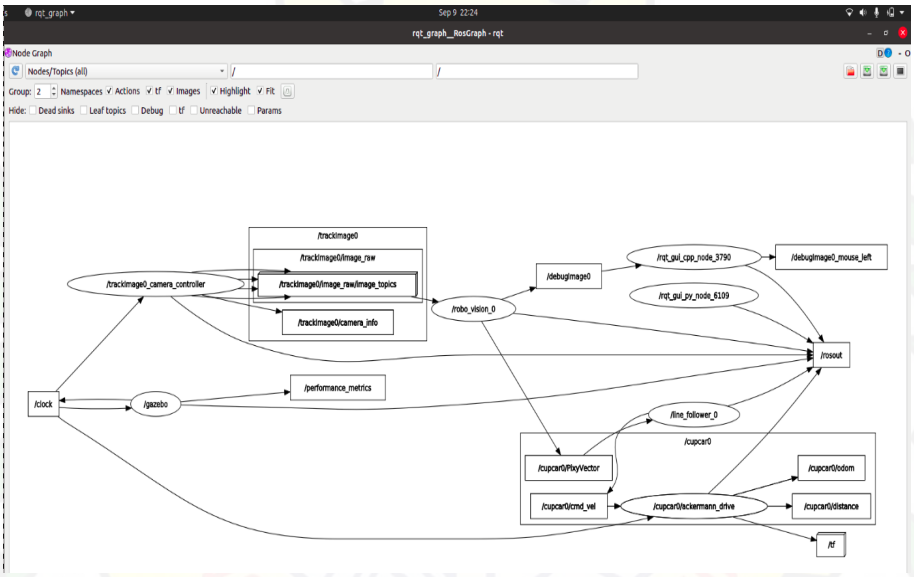
8.4 After installing Gazebo

To launch the simulation we run the command "ros2 launch sim_gazebo_bringup sim_gazebo.launch.py" in the terminal.

After installation, we explore the ros2ws package.

9 TELEOPERATION

Now we are going to find a way to control our car in gazebo simulator using a keyboard. So the first task was to find the subscriber which is responsible to control the wheels and this subscriber was pre-built.



So we used "rqt_graph"

Here the subscriber was **/cupcar0/ackermann**_drive and the topic of this subscriber was **cupcar0/cmd_vel**
so all that was needed was to publish using a twist class on this topic.
Twist contains two slots as _linear and _angular and further this linear velocity contains x, y z and the same goes for angular.

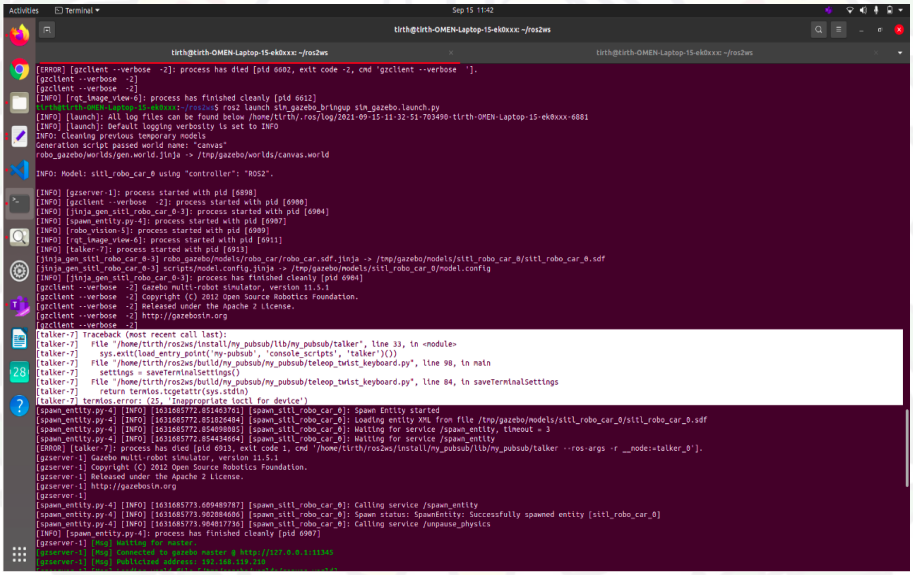
Further we used turtlesim node to check our teleop codes and the topic of this can be found using commands in terminal-
1. ros2 node list
2. ros2 topic list

The topic for this node was turtle1/cmd_vel responsible to control the motion of turtle Once we debugged the code then we used it to control the car. But before it we created one more package in SRC of ros2ws which is named as my_pubsub. In this package we copied the code from the below site. Here we changed the topic from

cmd_vel to cupcar0/cmd_vel and the car was working very well with the keyboard. Here the gazebo was running in 1st terminal and we manually called our teleop publisher in the other terminal. While controlling the car, the second terminal should also be open.

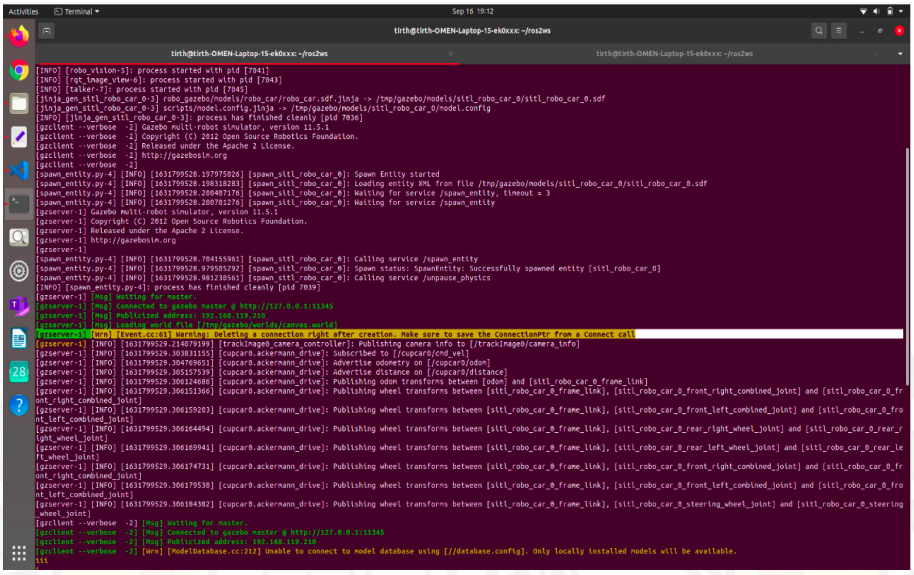
https://index.ros.org/r/teleop_twist_keyboard/github-ros2-teleop_twist_keyboard/

After this on 14/9/21 our mentors told us to make some changes to the launch file so that we can launch gazebo and my_pubsub both in the same terminal. We tried with our code, but we were facing some errors because of termios and the error is shown below in white.



To overcome this problem we created our own teleop code. We started on this on 9th of september and completed it on 11nth september. Here we used a blessed library of python which is specifically designed for terminal beautification. With this library we removed all the errors which were previously coming but still we were not able to launch both gazebo and my_pubsub together.
To install this library we used command "sudo pip3 install blessed" and then copied the code form the docs link ******

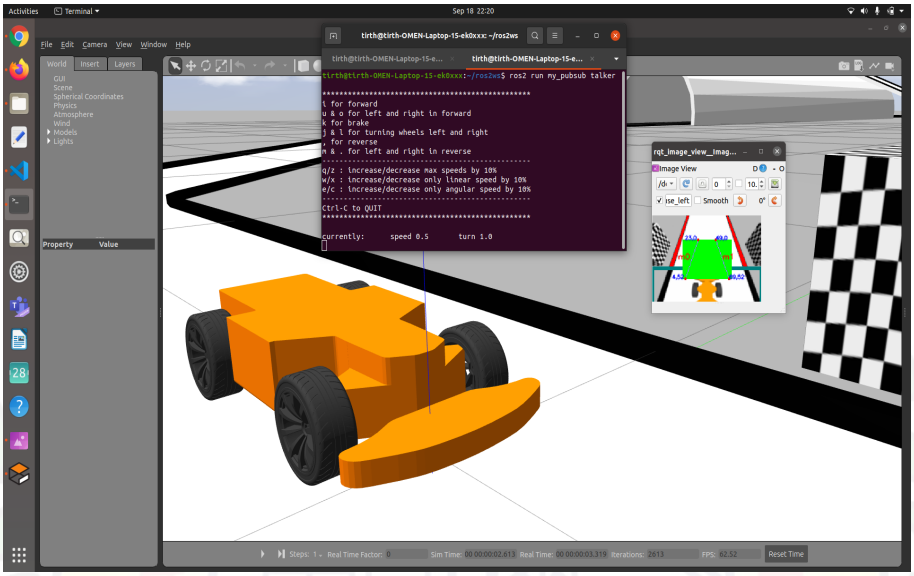
Terminal Image after implementation of this library and the new teleop code-



The reason is certainly that we were changing the settings of the virtual terminal which does not support gazebo launch file and so it was not working well. Further we even used stdin.read(1) function, curses library and termios but none of them were able to do it so finally it seems that we need to use two terminals, one for gazebo launch and other for teleop.

Note after "colcon build" we did "colcon build –symlink-install" and ". install/setup.bash". Here after this, we can do any changes in code files, then save it and on launching the simulation, those changes can easily be seen. Basically doing symlink install removes the task of doing colcon build continuously(which is not good to do for doing small changes in code). Further if new files are created than doing colcon build is must. Further symlink install is only available for python codes and should not be used with c++ or cmake.

1. Here for debugging purposes we can do this-
2. Simple create a file named xxx
3. Open notepad and save the untitled document as lol.py inside the newly created folder.
4. Type print("hello") inside lol.py file Now open virtual terminal and go in xxx by "cd xxx" and type "python3 lol.py"
5. This command will launch your python file and will print hello inside the terminal.
6. You can do this for any python file but before launching always save the file.



Final photo and the final video link of our project https://youtu.be/h5EK13J4QeI

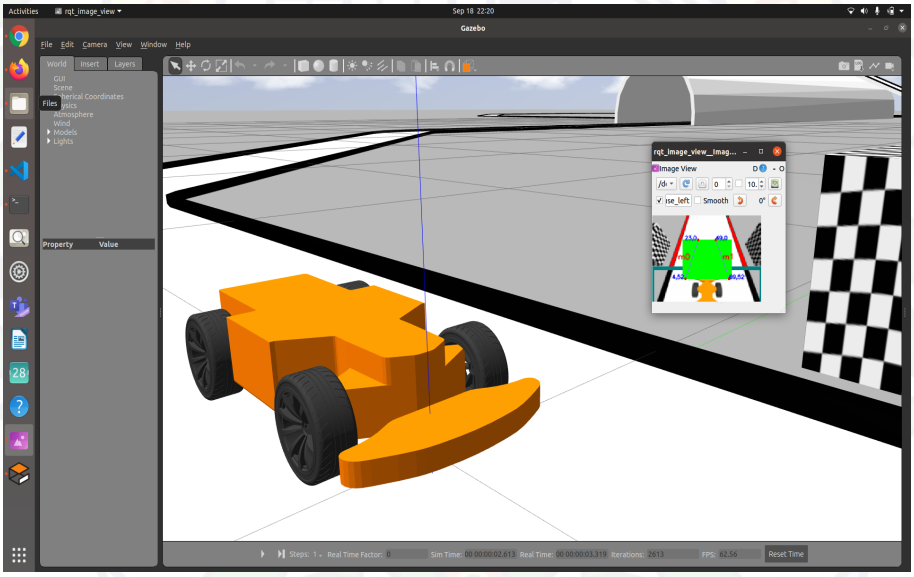
10 Correct codes of all the problems

Codes for all the problems for teleop and all other problems including the changes made in sdf files(model) to change wheels and apply colour on mesh.

https://docs.google.com/document/d/1gMmatMRsok7cL3cR5VxfWBCdyMLZToxLAGy7K-eV6Nw/edit?usp=drivesdk

For changing the looks of the car, Changes to be done in jinja files-
jinja file location - ros2ws/robo_gazebo/models/robo_car/robo_car.sdf.jinja

https://docs.google.com/document/d/15v4OdHQrLJ72K1REMO92OLmSEumnnH0if*r**LapFSQfk/ed**sharing*



11 Differences of ROS2 from ROS 1

Several changes and refactoring were done to gazebo_ros_pkgs when migrating from ROS 1 to ROS 2.

Some goals of the refactoring were:

1. Take advantage of new ROS 2 features, such as masterless discovery.
2. Remove code which duplicates functionality already present in Gazebo.
3. Reduce duplication by standardizing common functionality, such as how to set ROS namespaces, parameters and topic remapping.
4. Modernize the codebase, making use of the latest SDFormat, Gazebo and Ignition APIs, as well as ROS 2's style guidelines and linters.
5. Add tests and demos for all ported functionality.

Detailed migration guides for each plugin can be found on the gazebo_ros_pkgs wiki. See some general highlights below.

Init

The ROS 1 integration required that Gazebo be launched with the gazebo_ros_api_plugin system plugin, which would initialize ROS. There's no such requirement with ROS 2. Gazebo can be started without any plugins and ROS-2-enabled plugins can be added at runtime.

Node

In ROS 1, each plugin typically had one or more ros::NodeHandle instances to interact with ROS.

In ROS 2, plugins use gazebo_ros::Node instead, which allows each plugin to exist as its own node in the ROS graph, with its own parameters, namespace, loggers, etc. Plugins also don't need to worry about spinning the node or keeping callback queues - gazebo_ros handles all that internally.

SDF parsing

There are several configurations which Gazebo ROS plugins commonly want to set through SDF, and in the ROS 1 implementation, there was a lot of duplicate code on

plugins parsing the same things, sometimes following loose conventions. In ROS 2, common configurations like namespace, ROS parameters and topic remapping rules are parsed by gazebo_ros::Node, so there's no need for plugins to reimplement them every time.

gazebo_ros_api_plugin

In ROS 1, gazebo_ros_api_plugin was a large plugin which offered a lot of functionality in a bundle, giving users little flexibility to opt-in/out of features.

In ROS 2, this plugin is being split into smaller, more focused plugins. You can read the migration details on ROS 2 Migration: gazebo_ros_api_plugin.

Features	ROS	ROS2
Platforms	Tested on Ubuntu Maintained on other Linux flavors as well as OS X	ROS2 is currently being CI tested and supported on Ubuntu Xenial, OS X El Capitan as well as Windows 10
C++	C++03 // don't useC++11 features in its API	Mainly uses C++11 Start and plan to use C++14 & C++17
Python	Target Python 2	>= Python 3.5

Middleware	Custom serialization format (transport protocol + central discovery mechanism)	Currently all implementations of this interface are based on the DDS standard.
Unify duration and time types	The duration and time types are defined in the client libraries, they are in C++ and Python	In ROS2 these types are defined as messages and therefore are consistent across languages.
Components with life cycle	In ROS every node usually has its own main function.	The life cycle can be used by tools like roslaunch to start a system composed of many components in a deterministic way.
Threading model	In ROS the developer can only choose between single-threaded execution or multi-threaded execution.	In ROS2 more granular execution models are available and custom executors can be implemented easily.
Multiple nodes	In ROS it is not possible to create more than one node in a process.	In ROS2 it is possible to create multiple nodes in a process.
roslaunch	In ROS roslaunch files are defined in XML with very limited capabilities.	In ROS2 launch files are written in Python which enables the use of more complex logic like conditionals etc.

Simulators other than gazebo are LGSVL simulator
Why we use gazebo for simulation place of other simulators

With Gazebo you are able to create a 3D scenario on your computer with robots, obstacles and many other objects. Gazebo also uses a physical engine for illumination, gravity, inertia, etc. Further you can evaluate and test your robot in difficult or dangerous scenarios without doing any real harm to your robot. A well-designed simulator makes it possible to rapidly test algorithms, design robots, perform regression testing, and train AI systems using realistic scenarios. Gazebo offers the ability to accurately and efficiently simulate populations of robots in complex indoor and outdoor environments.

Available for MacOS, Linux and Windows. A binary package is only available for Linux Debian. Gazebo is installed via the command line using third-party package managers on other systems.

A scene is saved as an XML file. This makes it possible to create a bash script that changes the scene and then runs a simulation.
A less diverse library of default **robots**, that mostly includes wheeled and flying robots. Third-party robot models are available.
The default models are fairly simple and are therefore more appropriate for computationally complex simulations.
It also includes code and a scene editor.
No mesh manipulation is available.
Meshes are imported as single objects. Models that contain multiple sub-components have to be assembled in Gazebo from multiple DAE files, each corresponding to one sub-component.
Scene objects can be fully interacted with (e.g., moved or added) by the user during simulation. The world does not return to its original state when the simulation is reset.
Outputs include simulation log files, video frames as pictures and text files. A fairly comprehensive documentation, step-by-step tutorials and a large user community are available. Gazebo is likely to be supported in the future, since a development road map is available on the website.

12 Timeline

Aug 20, 2021 -We started our project.

Aug 24, 2021 - The team is done with Ubuntu installation. And started work on installing ROS2.

Aug 25, 2021 - Everyone installed ROS2. Now ready to explore the package.

Aug 29, 2021 - We are done with learning basic python OOP along with the creating our first package which contain publisher and subscriber. Till now we learnt about nodes and topic and other related things.

Sept 01, 2021 - We were given a task to create a system of multiple publishers and subscribers. We completed that. Now we start installing GAZEBO.

Sept 04, 2021 - We completed the GAZEBO installation and started to explore Gazebo files.

Sept 09, 2021 - We explore Gazebo files and learn how Gazebo works. After exploration we start implement teleoperation on car.

Sept 15, 2021 - We made code by our self and succesfully implement teleoperation on our car and finally we completed our aim.

Sept 16, 2021 - We more explore SDF file and learn our car was design.

Sept 17, 2021 - We implement different colour on car and we change visual of the wheels.