A Final Year Project Report on

Study of locomotion of multiple robots, both wheeled and flying robots in Gazebo, using ROS.

Under The Supervision Of

CHINTAN KUMAR MANDAL

Assistant Professor,

Computer Science and Engineering Department

Faculty of Engineering and Technology

Jadavpur University,

Submitted By

Md Sajjad Ansari

Class Roll. No. : 001710501031

Exam Roll no. : CSE218025

Registration no. : 139874 of 2017-2018

JADAVPUR UNIVERSITY

Department of Computer Science & Engineering

ACKNOWLEDGEMENT

My sincere and gratitude thanks to our Project Supervisor Assistant Professor Chintan Kumar Mandal, Computer Science and Engineering Department, Faculty of Engineering and Technology, Jadavpur University, who gave us this golden opportunityto do this project under his supervision and guidance. Without his motivational wordings we wouldn't be able to set off this project. His guidance gave us a lot of confidence.

And finally, a very-very thanks to our parents who cooperated with us a lot in bringingthis project up to the final destination and gave a feeling of being beatitude.

(Md Sajjad Ansari)

Content

1. OBJECTIVE

Motive of this Project is to Study and understand the locomotion of multiple robots, both wheeled and flying robots in Gazebo, using ROS. For the purpose of this I have building a world in gazebo first using the gazebo database and then build a model (i.e. wheeled Robot) and put it into the constructed world and understand it's motion and locomotive nature.

2. What is Gazebo

Gazebo is a 3D dynamic simulator with the ability to accurately and efficiently simulate populations of robots in complex indoor and outdoor environments. While similar to game engines, Gazebo offers physics simulation at a much higher degree of fidelity, a suite of sensors, and interfaces for both users and programs.

Typical uses of Gazebo include:

- testing robotics algorithms,
- designing robots,
- performing regression testing with realistic scenarios

A few key features of Gazebo include:

- multiple physics engines,
- a rich library of robot models and environments,
- a wide variety of sensors,
- convenient programmatic and graphical interfaces

2.1 Why Gazebo?

Robot simulation is an essential tool in every roboticist's toolbox. A well-designed simulator makes it possible to rapidly test algorithms, design robots, perform regression testing, and train AI system using realistic scenarios. Gazebo offers the ability to accurately and efficiently simulate populations of robots in complex indoor and outdoor environments. At your fingertips is a robust physics engine, high-quality graphics, and convenient programmatic and graphical interfaces. Best of all, Gazebo is free with a vibrant community.

2.2 System requirements

Gazebo is currently best used on Ubuntu, a flavor of Linux. You will need a computer that has:

- A dedicated GPU,
    - Nvidia cards tend to work well in Ubuntu
- A CPU that is at least an Intel I5, or equivalent,
- At least 500MB of free disk space,
- Ubuntu Trusty or later installed.

3. What is ROS?

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. It also provides tools and libraries for obtaining, building, writing, and running code across multiple computers. ROS is similar in some respects to 'robot frameworks,' such as Player, YARP, Orocos, CARMEN, Orca, MOOS, and Microsoft Robotics Studio.

The ROS runtime "graph" is a peer-to-peer network of processes (potentially distributed across machines) that are loosely coupled using the ROS communication infrastructure. ROS implements several different styles of communication, including synchronous RPC-style communication over services, asynchronous streaming of data over topics, and storage of data on a Parameter Server. These are explained in greater detail in our Conceptual Overview.

ROS is not a realtime framework, though it is possible to integrate ROS with realtime code. The Willow Garage PR2 robot uses a system called pr2_etherCAT, which transports ROS messages in and out of a realtime process. ROS also has seamless integration with the Orocos Real-time Toolkit.

3.1 Goals

A lot of people ask, "How is ROS different from X?" where X is another robotics software platform. It's a difficult question to answer as the goal of ROS is *not* to be a framework with the most features. Instead, the primary goal of ROS is to support code *reuse* in robotics research and development. ROS is a distributed framework of processes (aka *Nodes*) that enables executables to be individually designed and loosely coupled at runtime. These processes can be grouped into *Packages* and *Stacks*, which can be easily shared and distributed. ROS also supports a federated system of code *Repositories* that enable collaboration to be distributed as well. This design, from the filesystem level to the community level, enables independent decisions about development and implementation, but all can be brought together with ROS infrastructure tools.

In support of this primary goal of sharing and collaboration, there are several other goals of the ROS framework:

- Thin: ROS is designed to be as thin as possible -- we won't wrap your main() -- so that code written for ROS can be used with other robot software frameworks. A corollary to this is that ROS is easy to integrate with other robot software frameworks: ROS has already been integrated with OpenRAVE, Orocos, and Player.
- ROS-agnostic libraries: the preferred development model is to write ROS-agnostic libraries with clean functional interfaces.
- Language independence: the ROS framework is easy to implement in any modern programming language. We have already implemented it in Python, C++, and Lisp, and we have experimental libraries in Java and Lua.
- Easy testing: ROS has a builtin unit/integration test framework called rostest that makes it easy to bring up and tear down test fixtures.
- Scaling: ROS is appropriate for large runtime systems and for large development processes

3.2 Operating Systems

ROS currently only runs on Unix-based platforms. Software for ROS is primarily tested on Ubuntu and Mac OS X systems, though the ROS community has been contributing support for Fedora, Gentoo, Arch Linux and other Linux platforms.

While a port to Microsoft Windows for ROS is possible, it has not yet been fully explored.

3.3 Releases

The core ROS system, along with useful tools and libraries are regularly released as a ROS Distribution. This distribution is similar to a Linux distribution and provides a set of compatible software for others to use and build upon.

3.4 Contributing

As ROS is open source, we hope that you will consider contributing to ROS or libraries that are compatible with ROS. Please see our section on Contributing for more information on how you can participate in the ROS community.

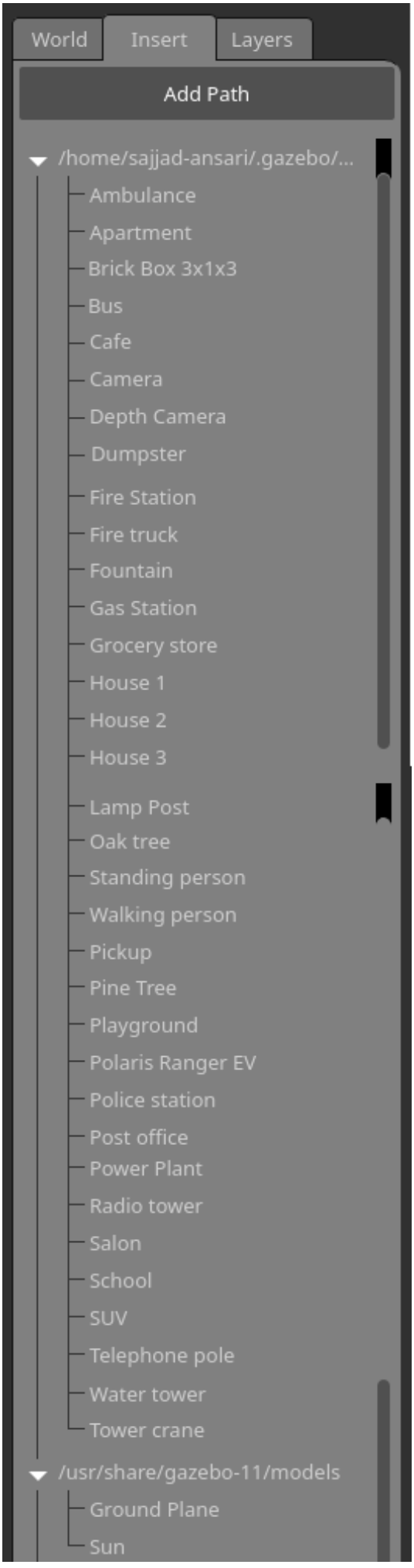
4. Building a Gazebo World

Gazebo relies on a database to store and maintain models available for use within simulation. The model database is a community-supported resource, so People upload and maintain models that you create and use.

Gazebo is able to dynamically load models into simulation either programmatically or through the GUI. Models exist on your computer, after they have been downloaded or created by you. This tutorial describes Gazebo's model directory structure, and the necessary files within a model directory.

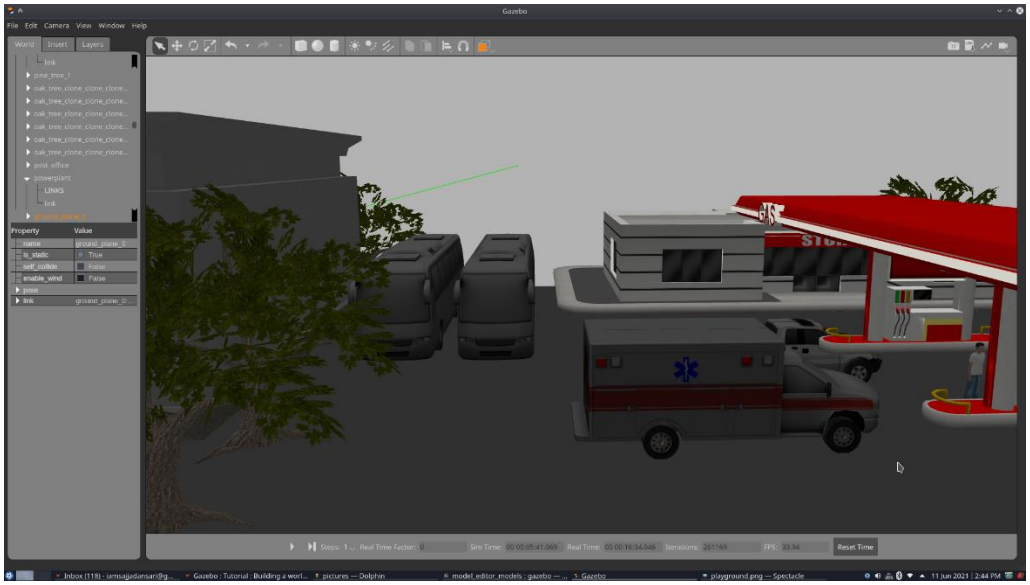
Models in Gazebo define a physical entity with dynamic, kinematic, and visual properties. In addition, a model may have one or more plugins, which affect the model's behavior. A model can represent anything from a simple shape to a complex robot; even the ground is a model.

Here is the list of Models I have used from Gazebo Database to build virtual world.

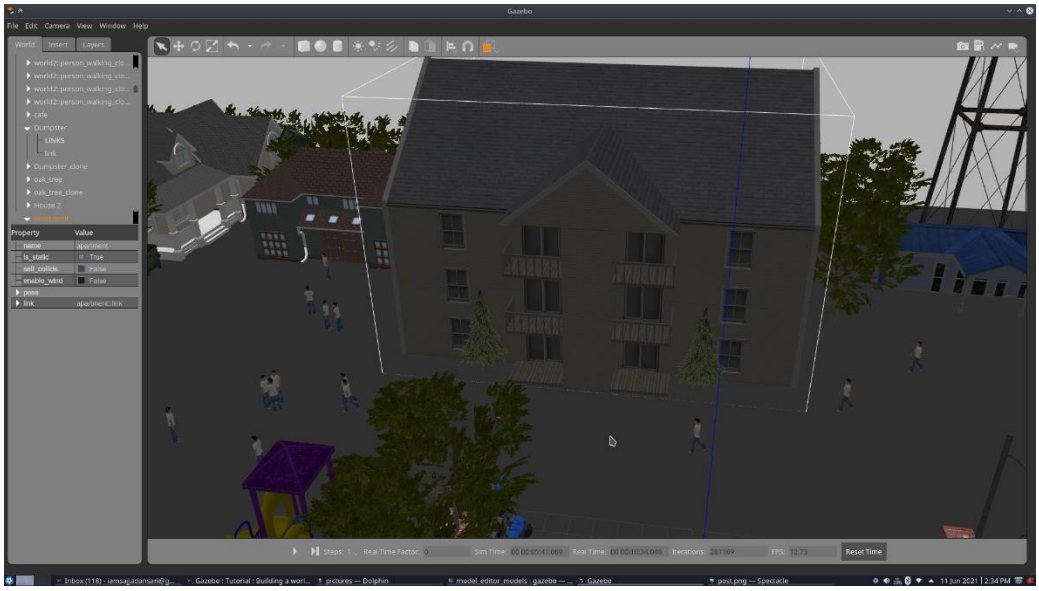


4.1 World Elements and Models

Ambulance:

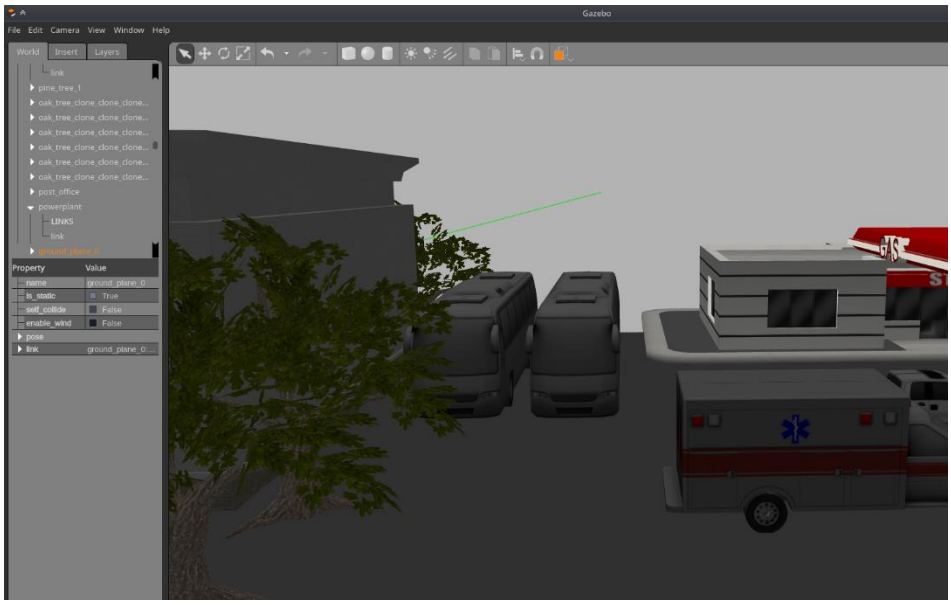


Apartment:

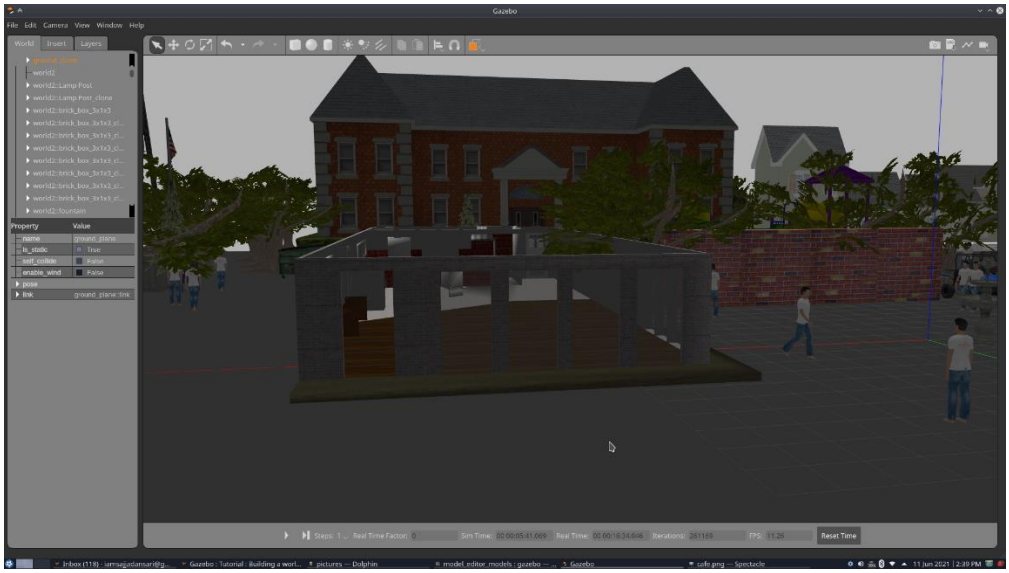


Brick Box:

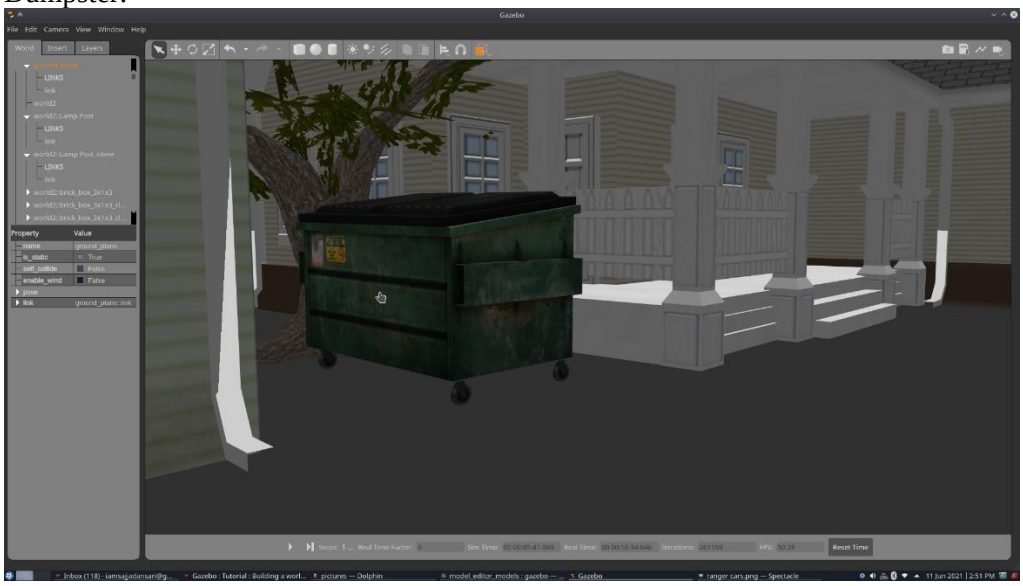
Bus:



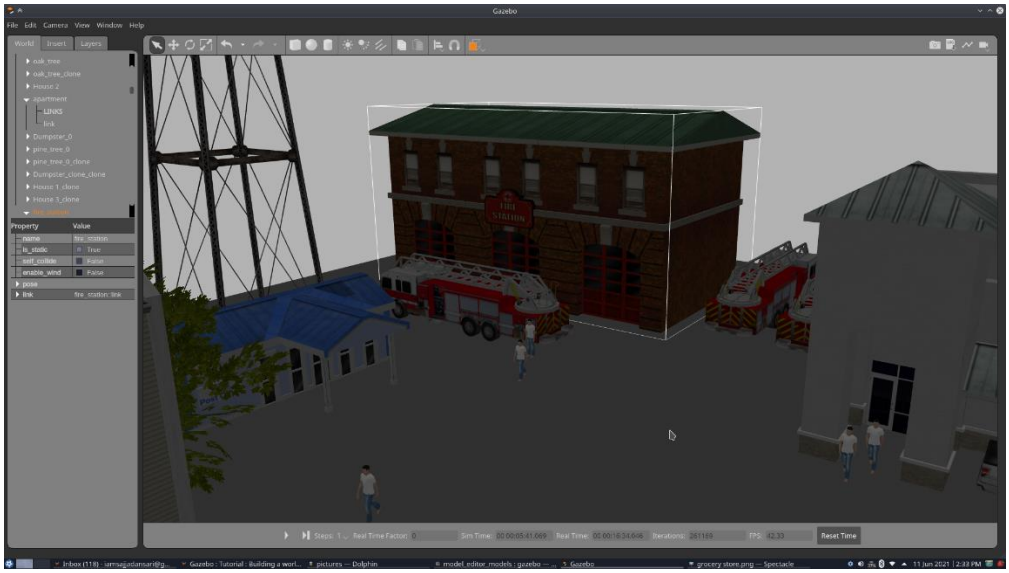
Café:



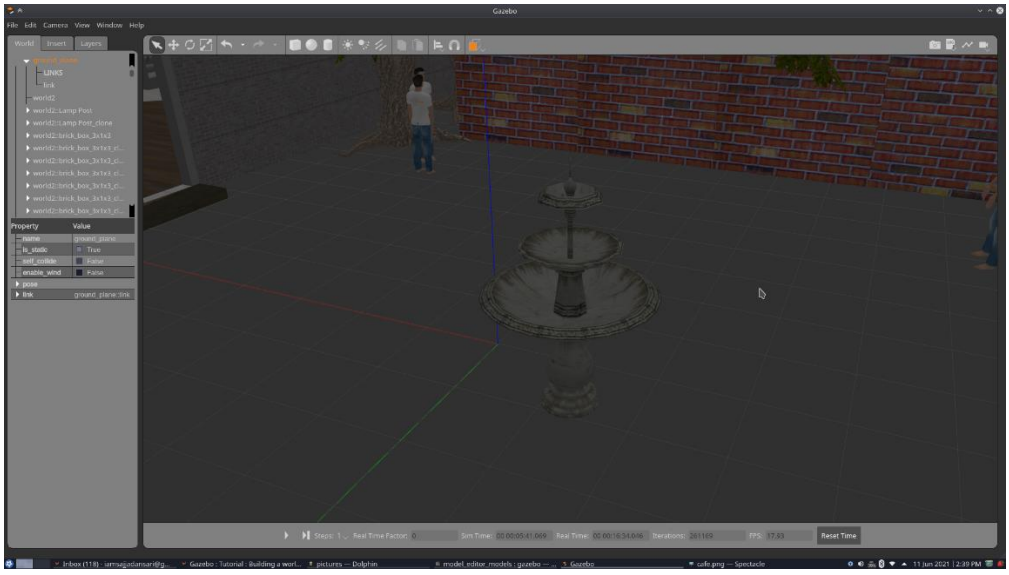
Dumpster:



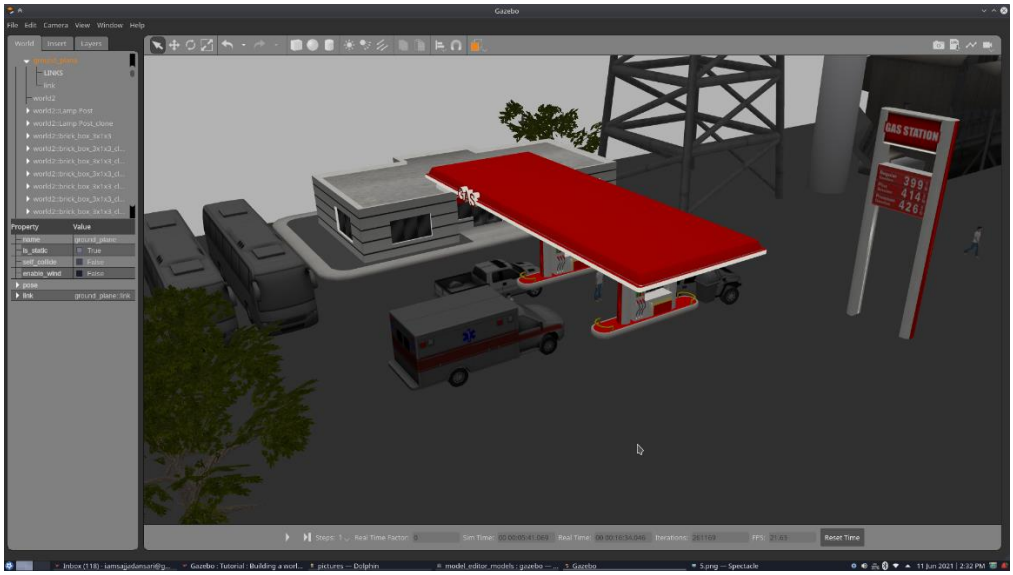
Fire Station and Fire Truck:



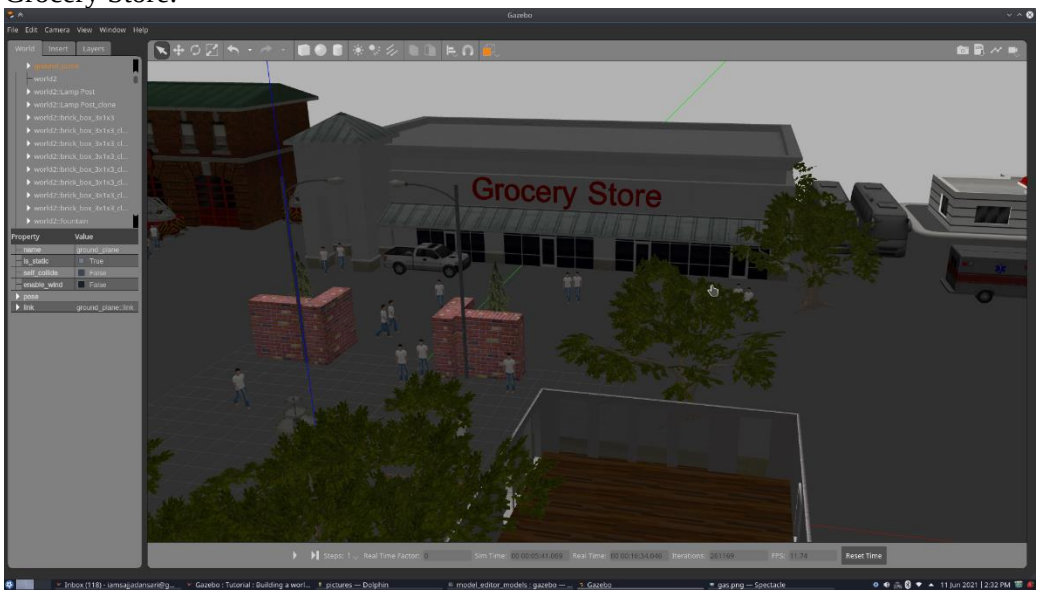
Fountain:



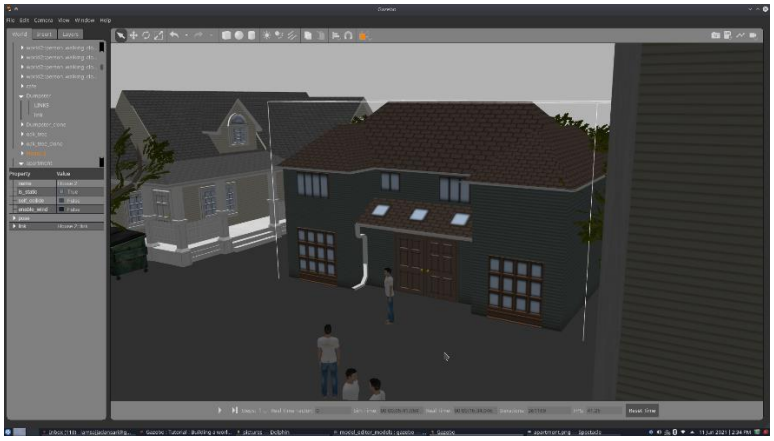
Gas Station:



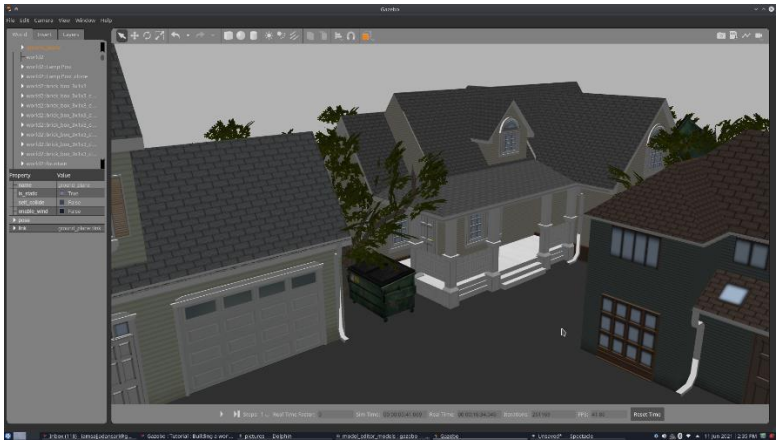
Grocery Store:



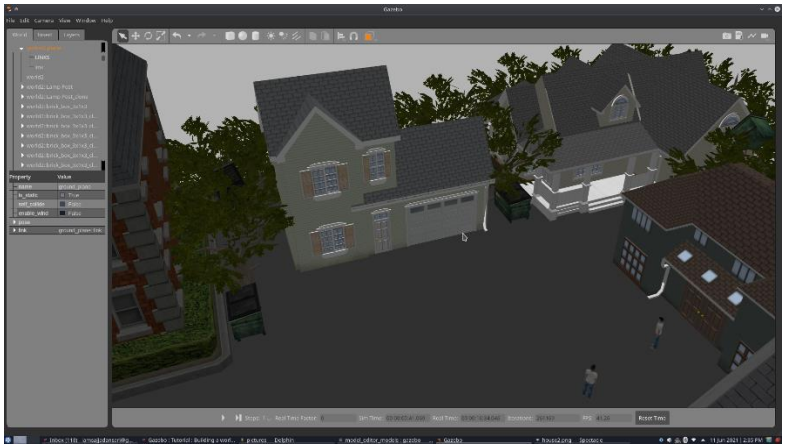
House 1 :



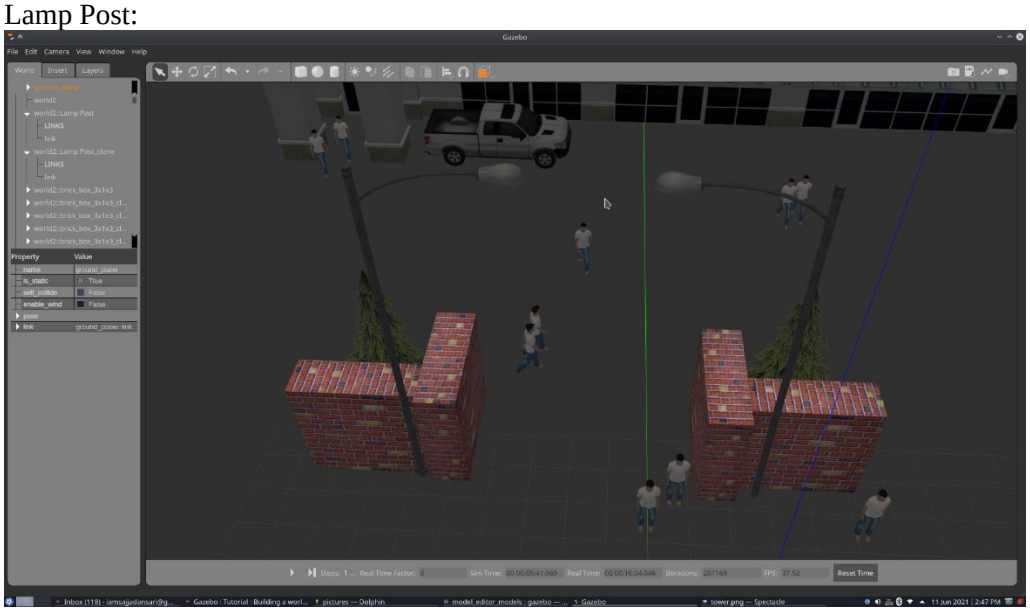
House 2 :



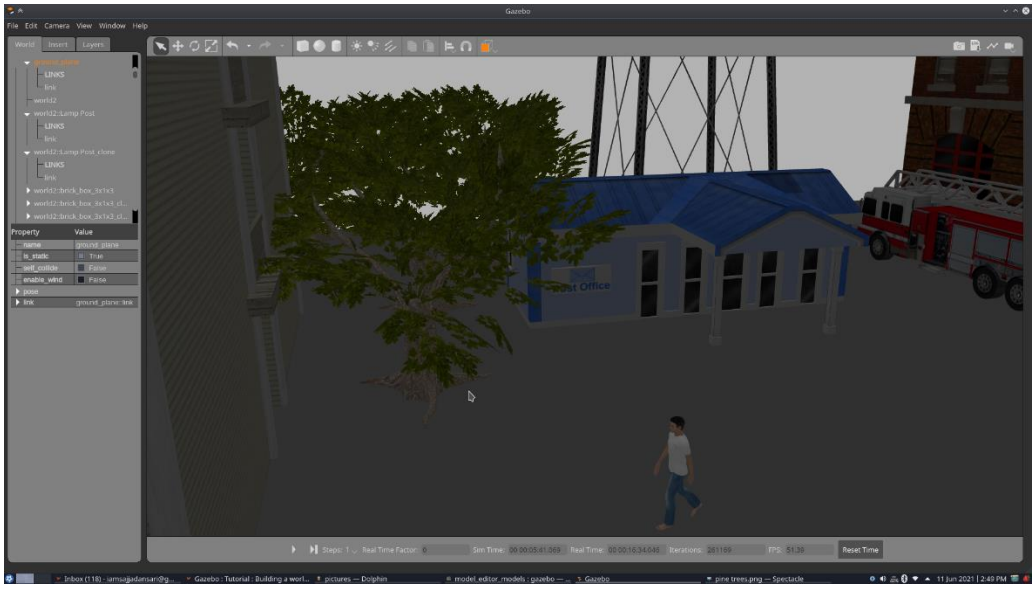
House 3 :



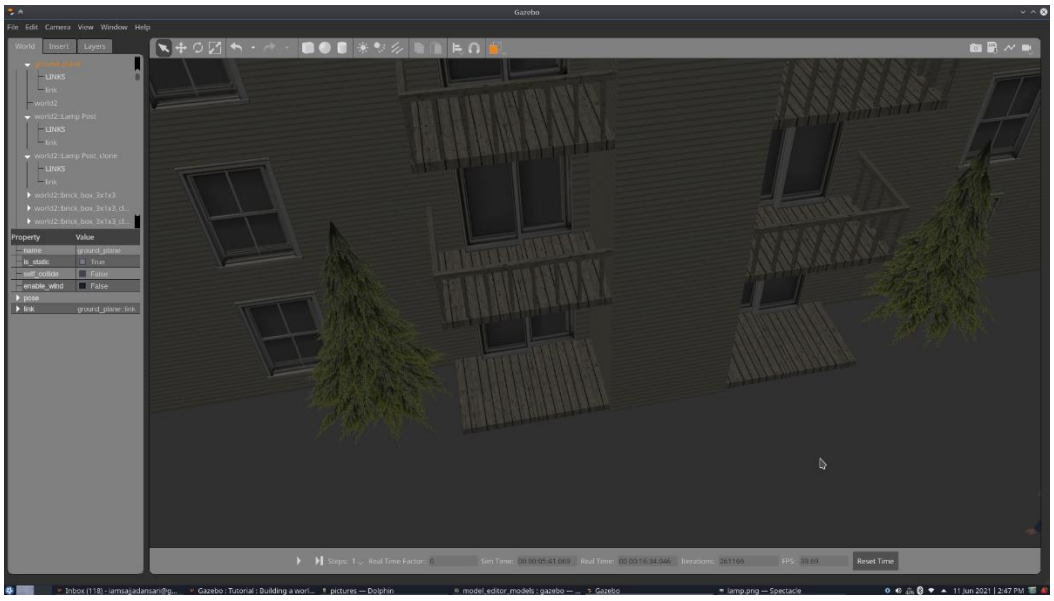
Lamp Post:



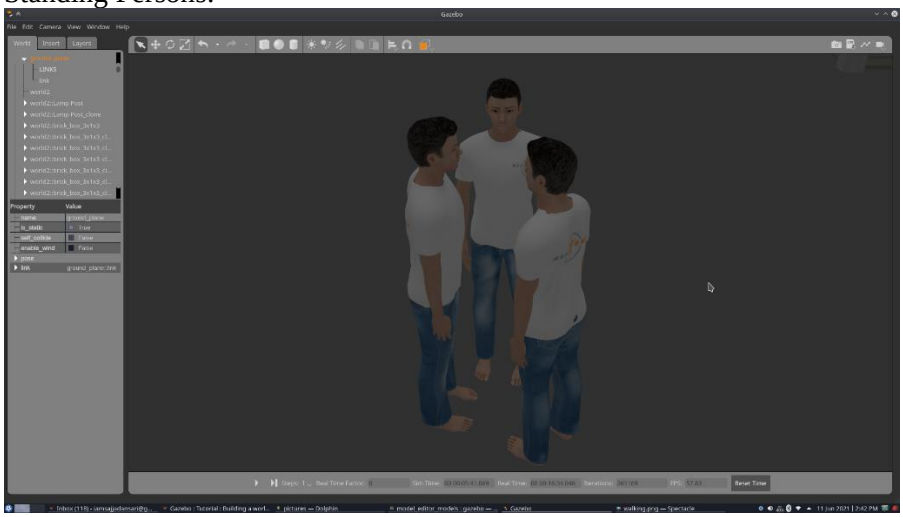
Oak Tree:



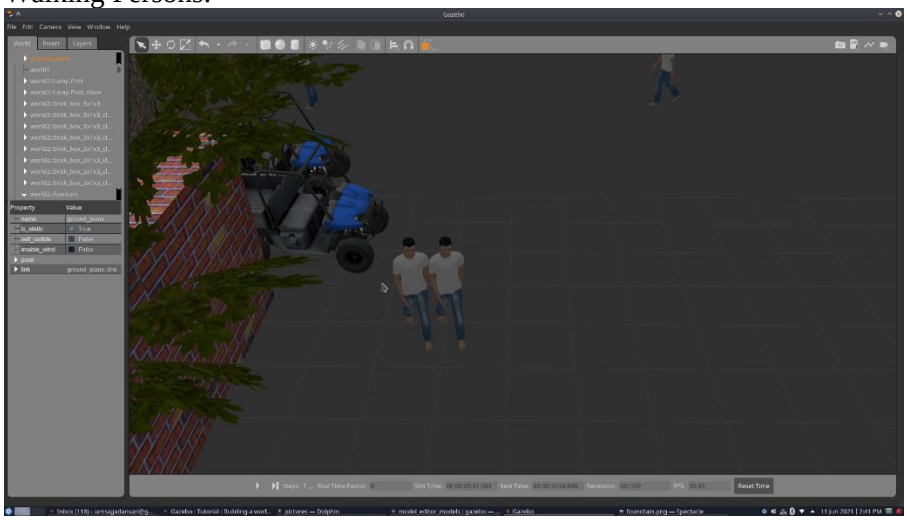
Pine Tree:



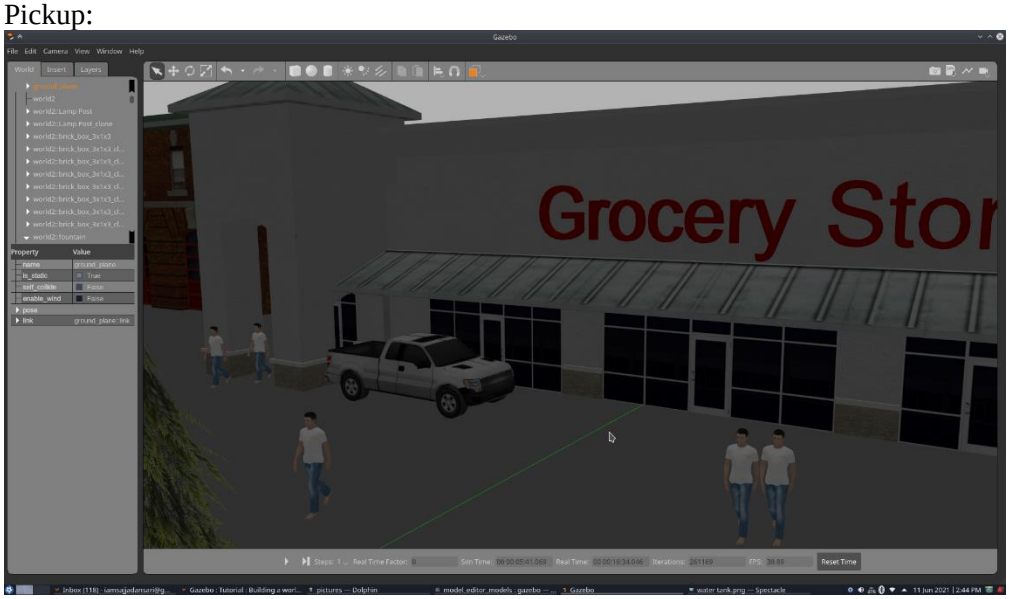
Standing Persons:



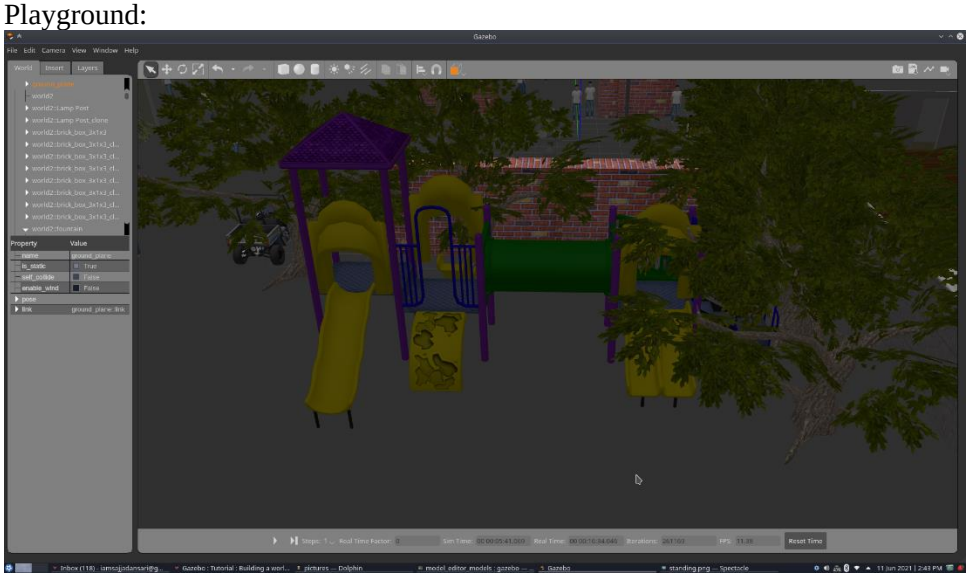
Walking Persons:



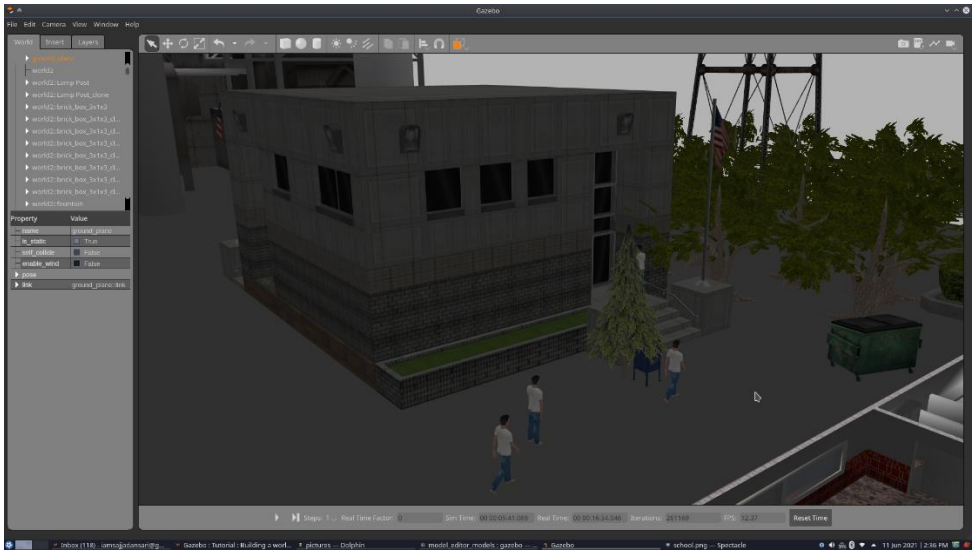
Pickup:



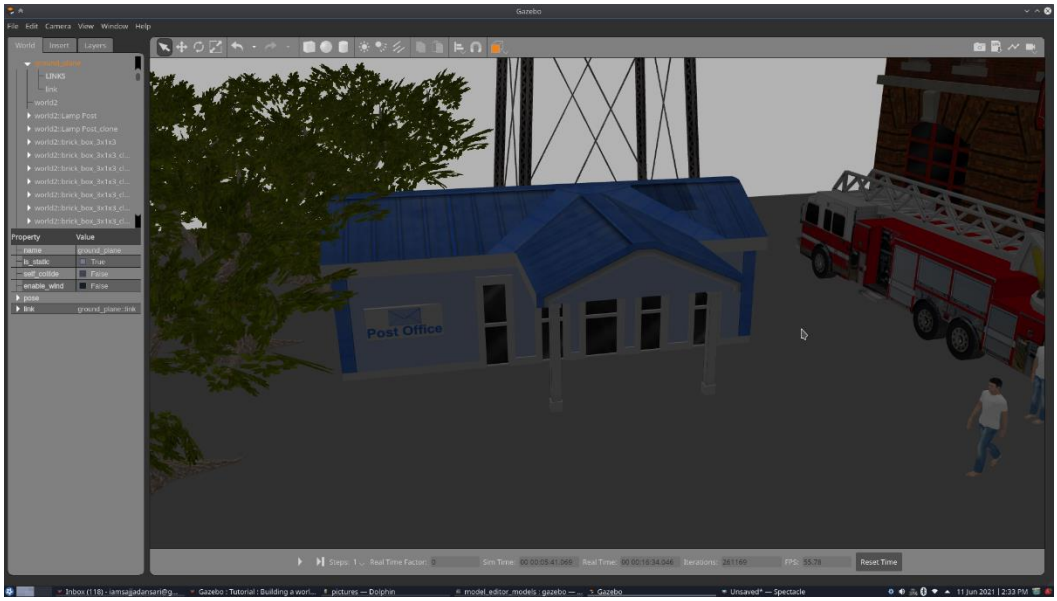
Playground:



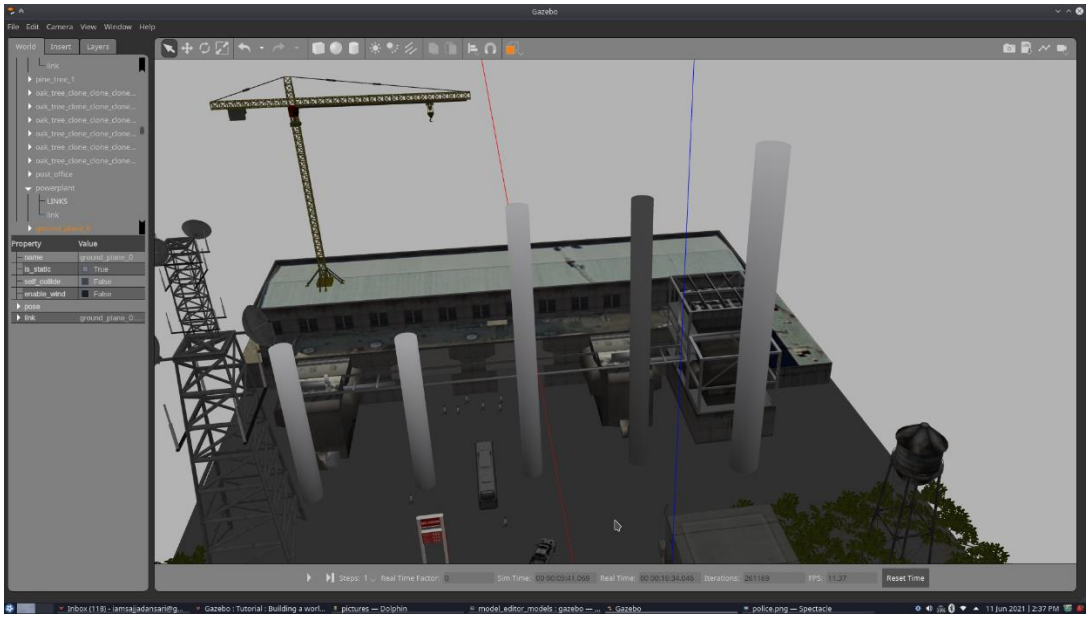
Police Station:



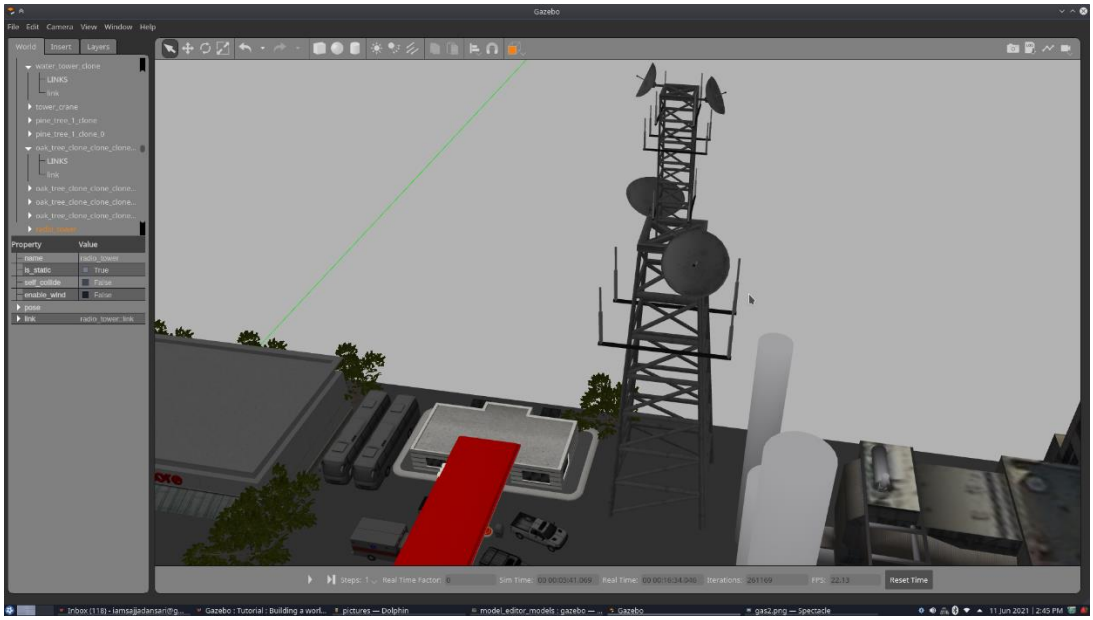
Post Office:



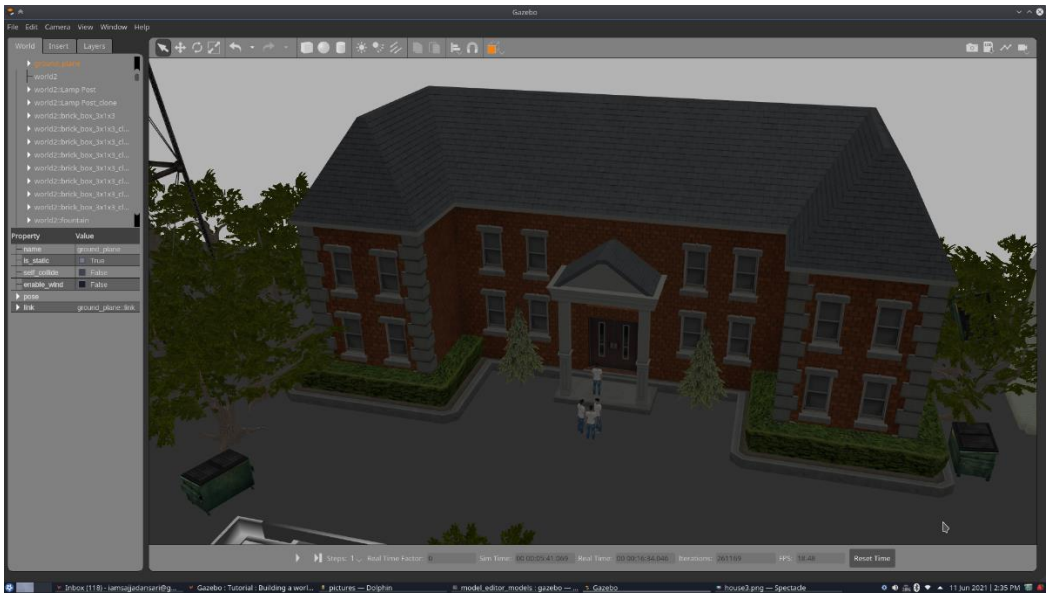
Power Plant:



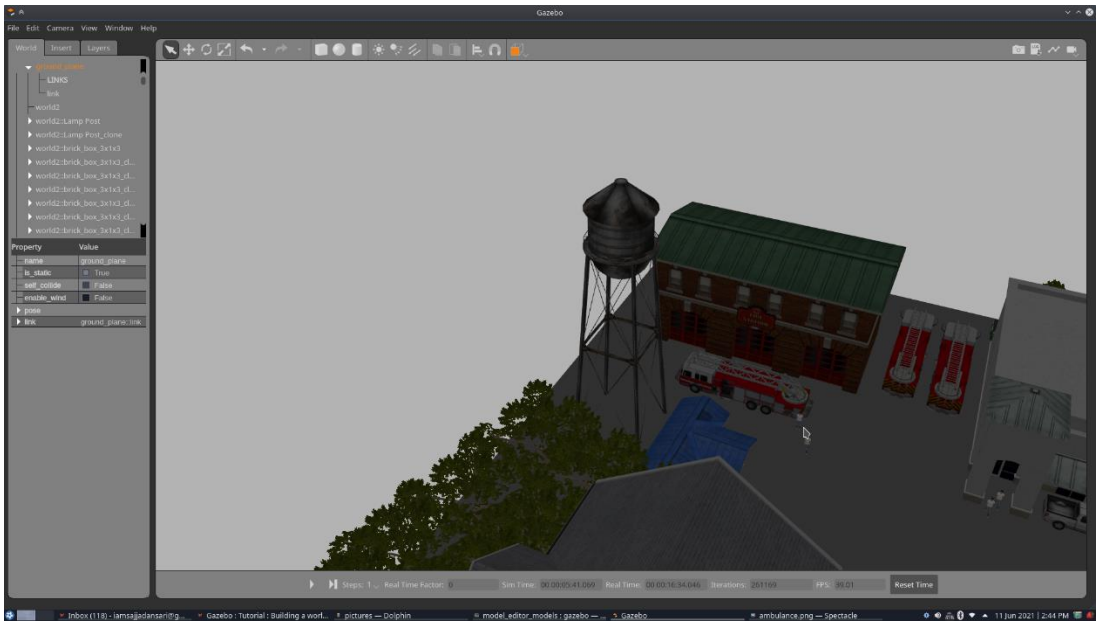
Radio Tower:



School:

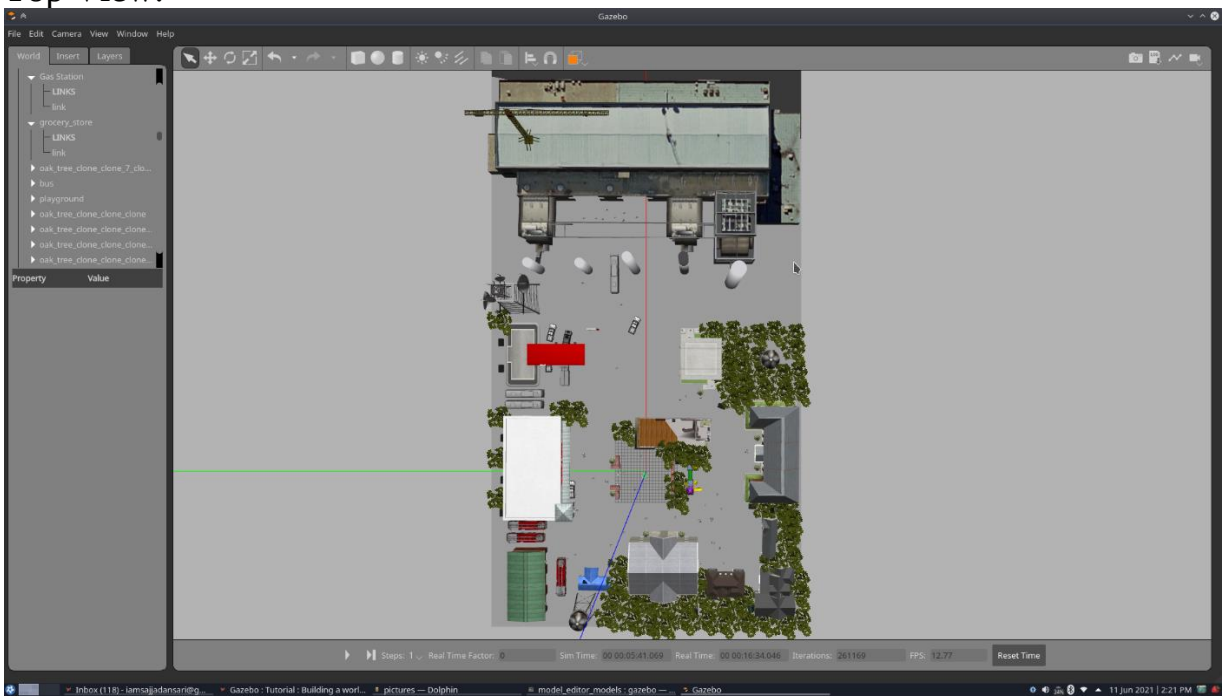


Water Tower:

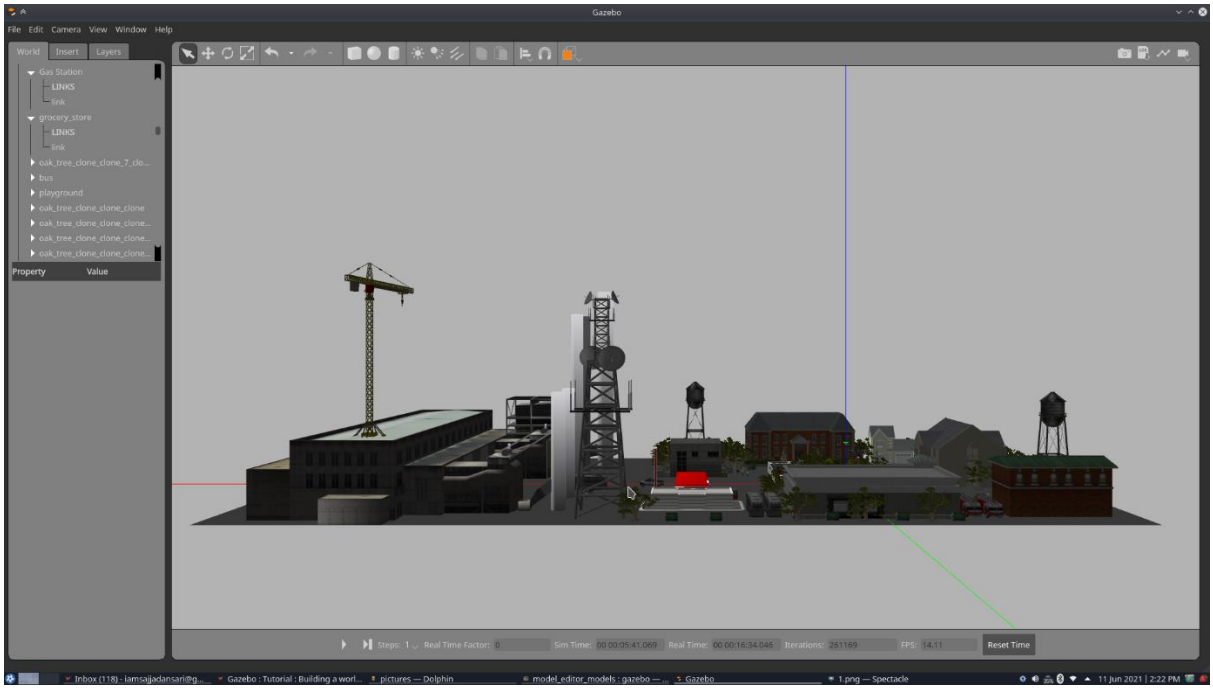


4.2 Complete World

Top View:



Side View:



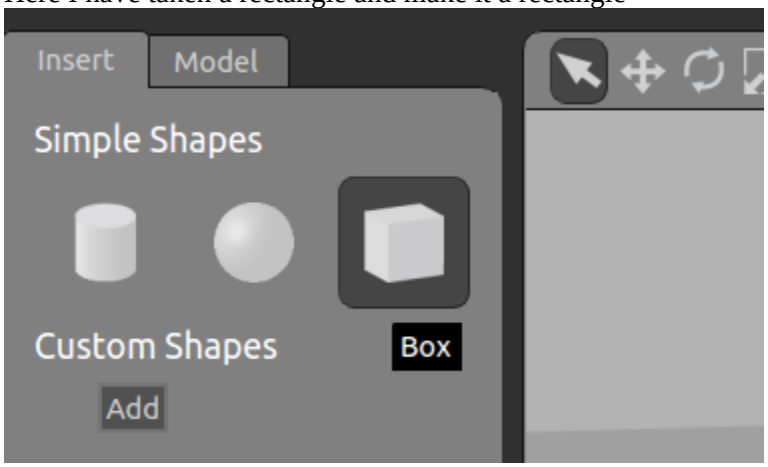
Code File of World Available here:

https://drive.google.com/file/d/1FzhT8VM-Ne0JZNq3nNQOTDaZ8E1DpckK/view?usp=sharing

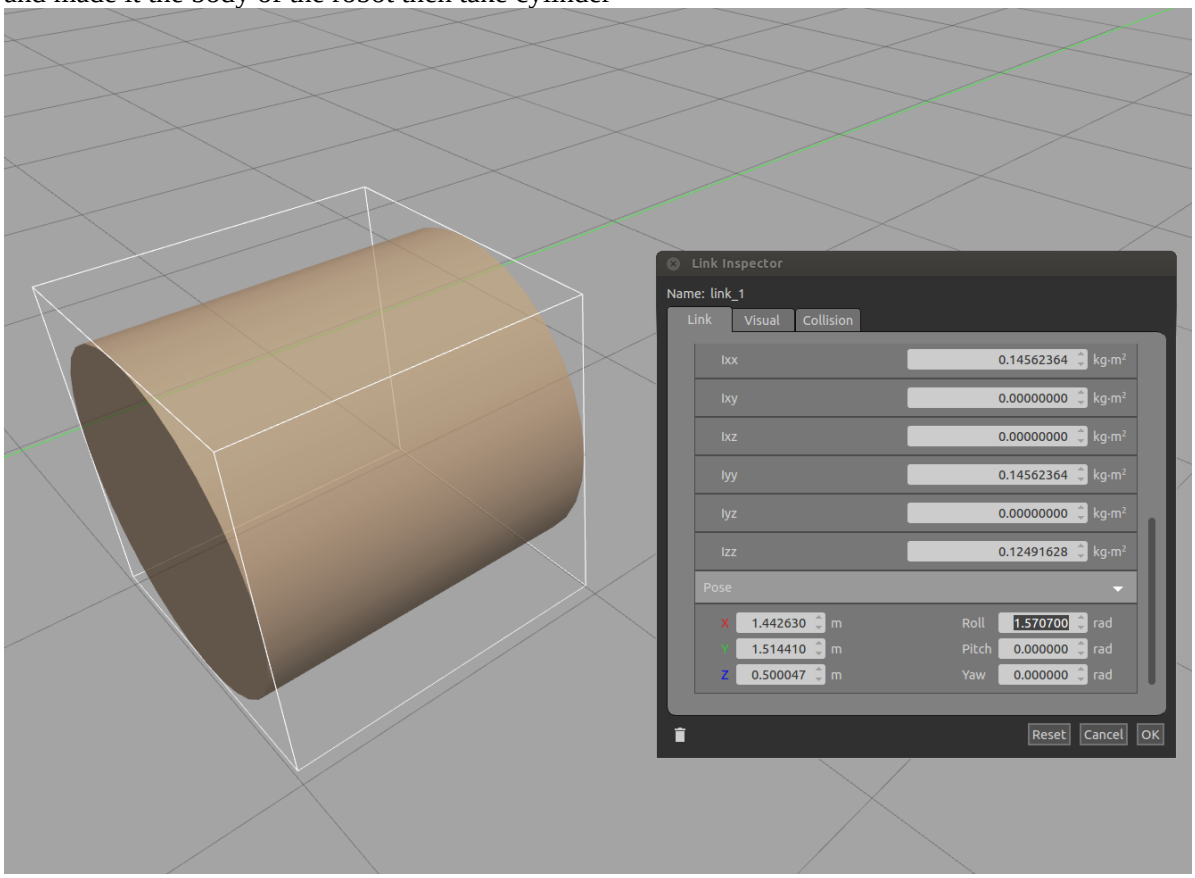
5. Budling a Robot with attached Sensors

Now we'll construct our simple robot. We'll make a wheeled Robot and add sensors.
The Model Editor lets us construct simple models right in the Graphical User Interface (GUI).

Here I have taken a rectangle and make it a rectangle

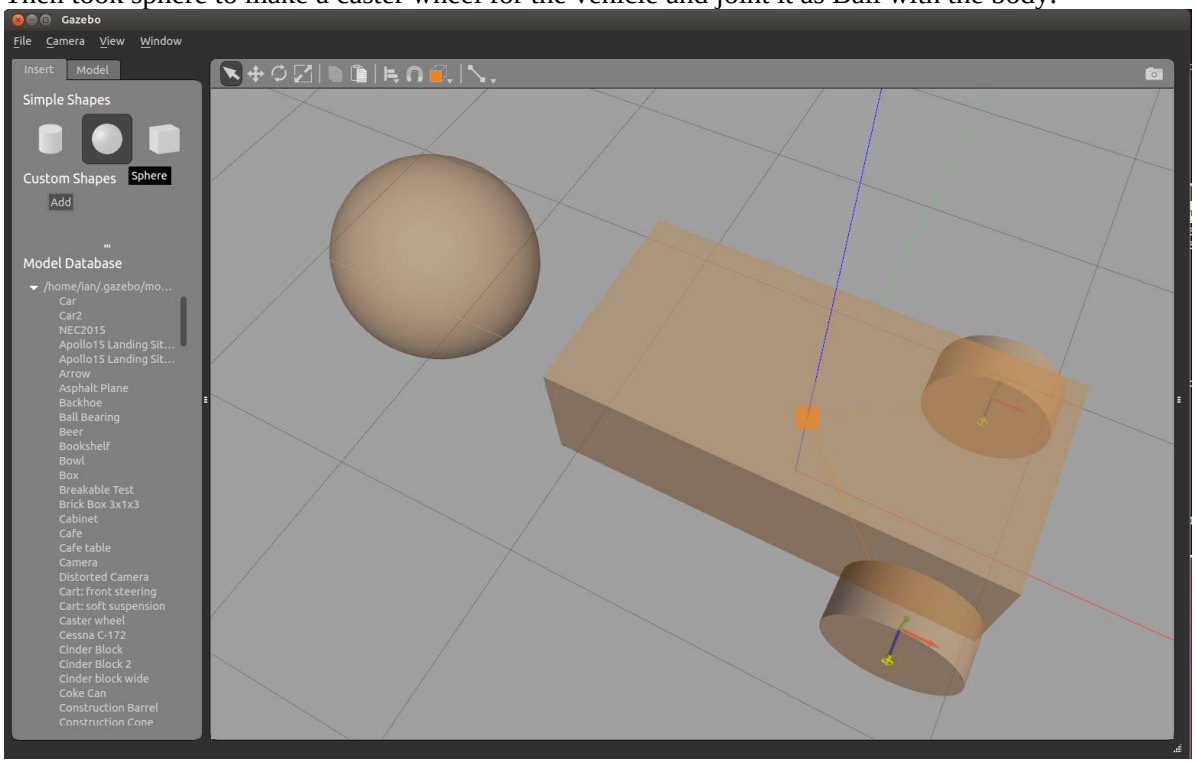


and made it the body of the robot then take cylinder



and resize it to make two wheels
then joint them as Revolving with the body where the body is parent and cylinders are the child

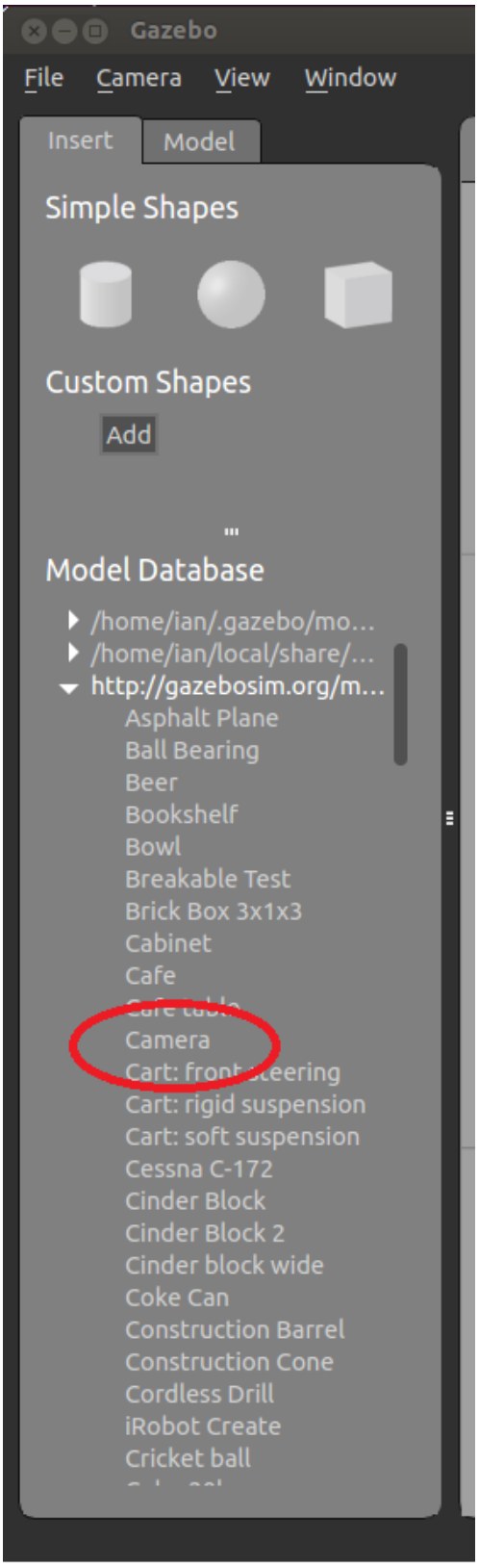
Then took sphere to make a caster wheel for the vehicle and joint it as Ball with the body.



5.2 Adding sensor

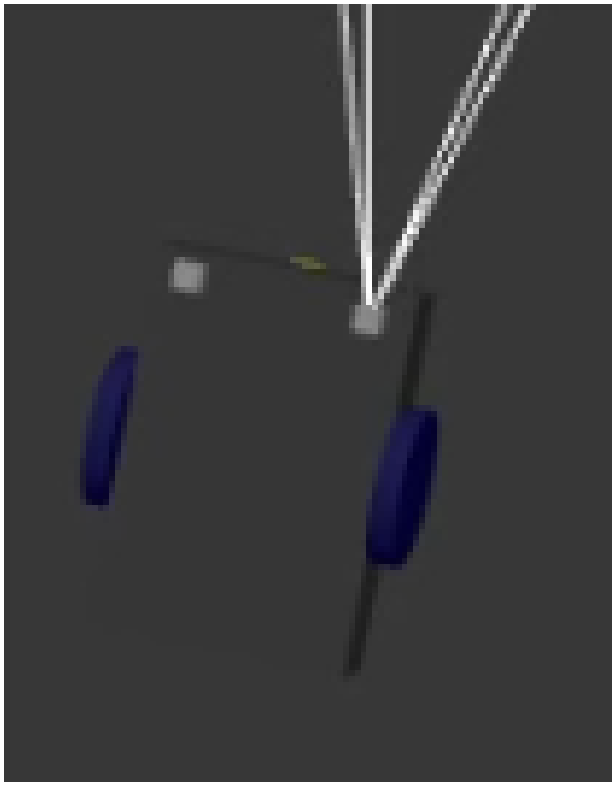
The sensor we will add to the car is a camera and depth camera sensor which is going to help us detect objects in front of the car. we will insert an existing sensor model from the model database.

And place them on the robot and joint it as a Fixed.

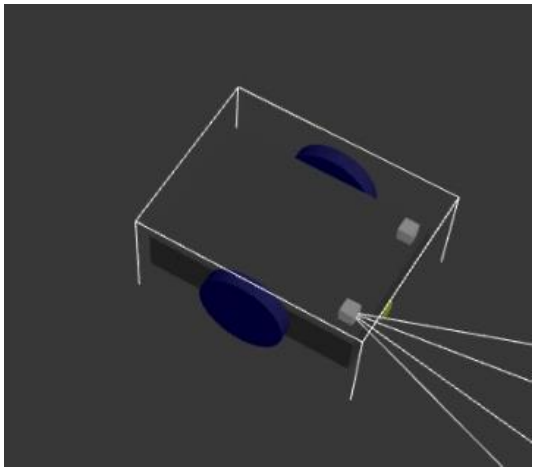


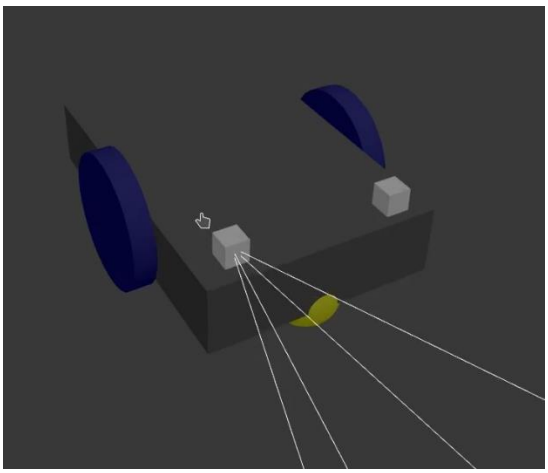
5.3 Complete Robot

Top View:

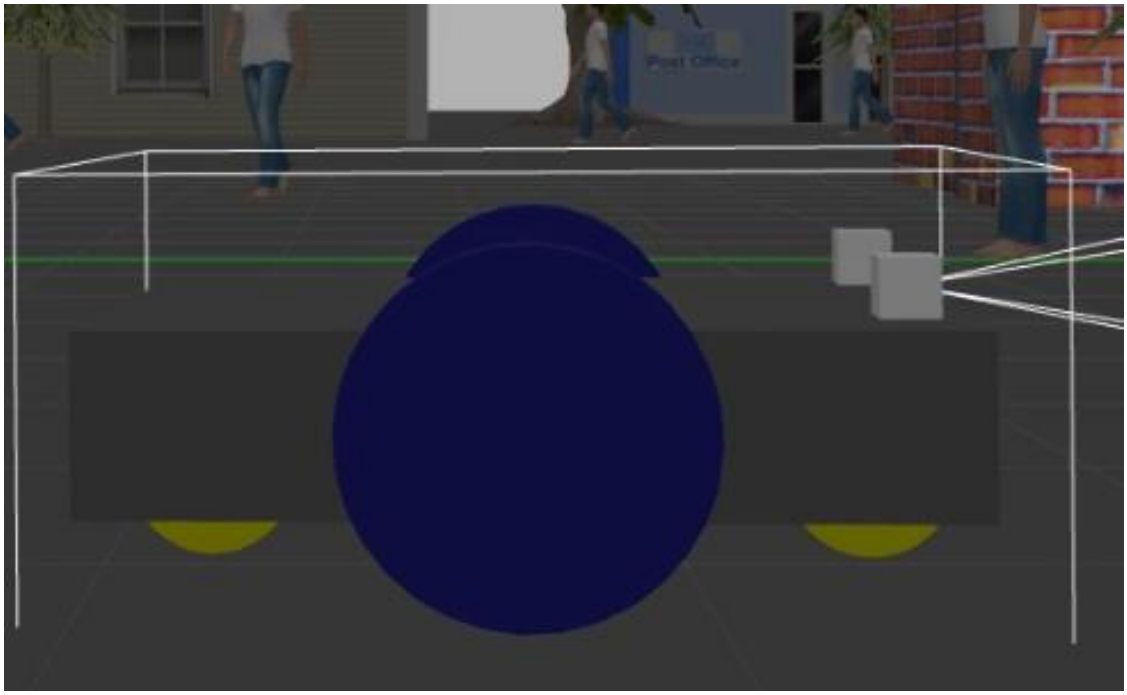


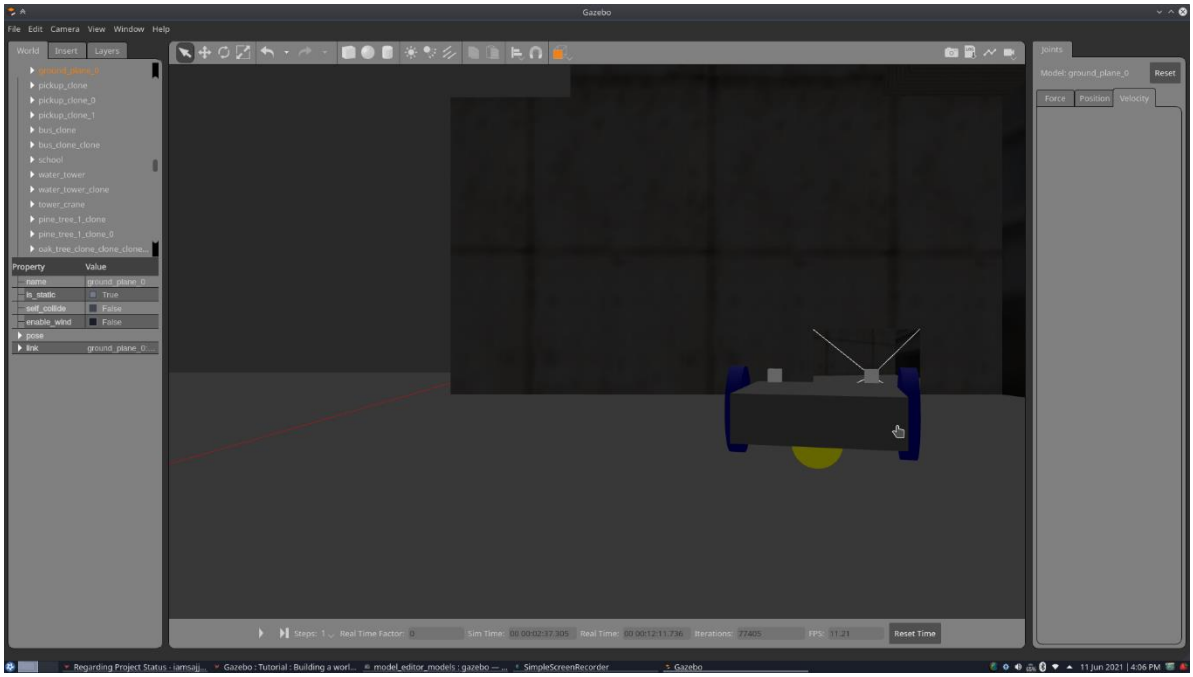
Angle Views:





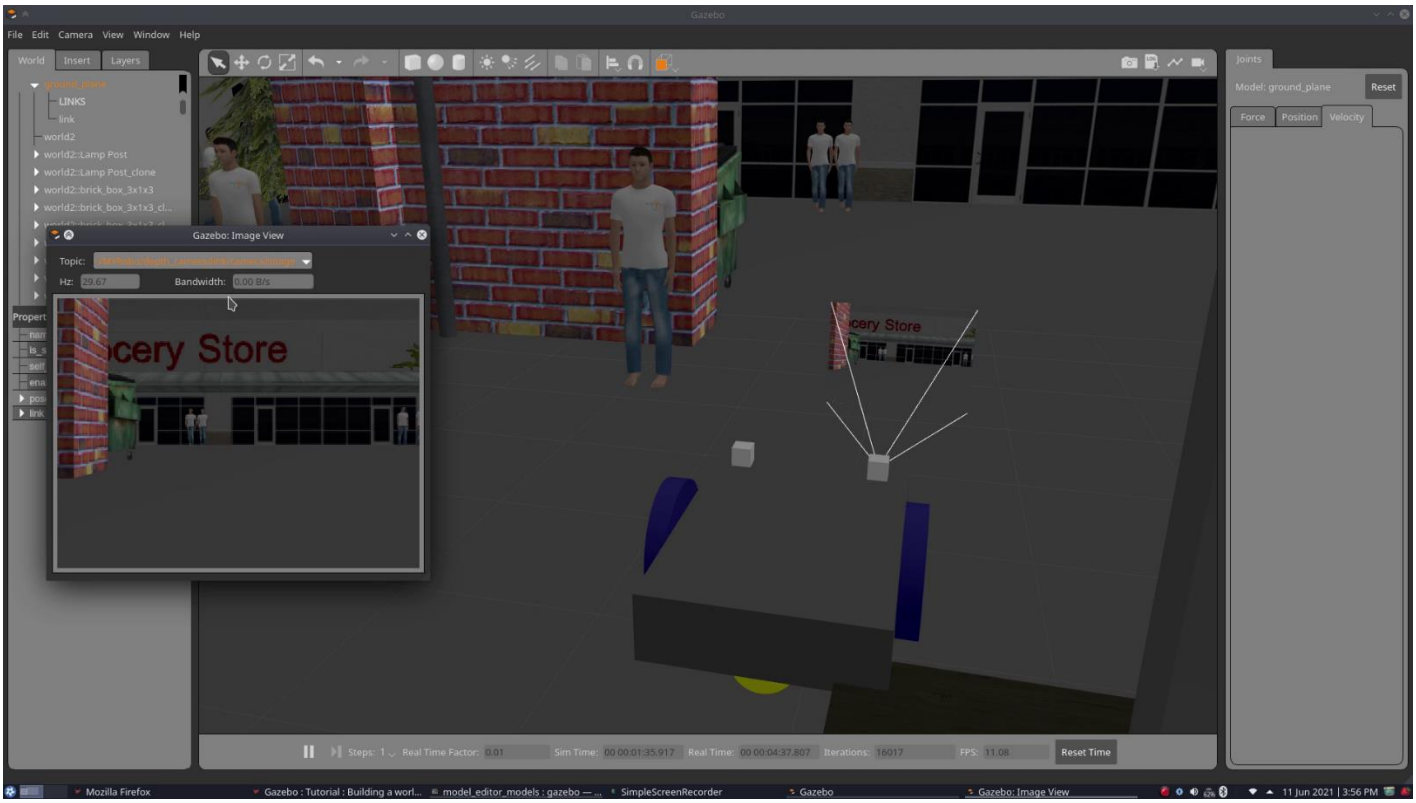
Side View:



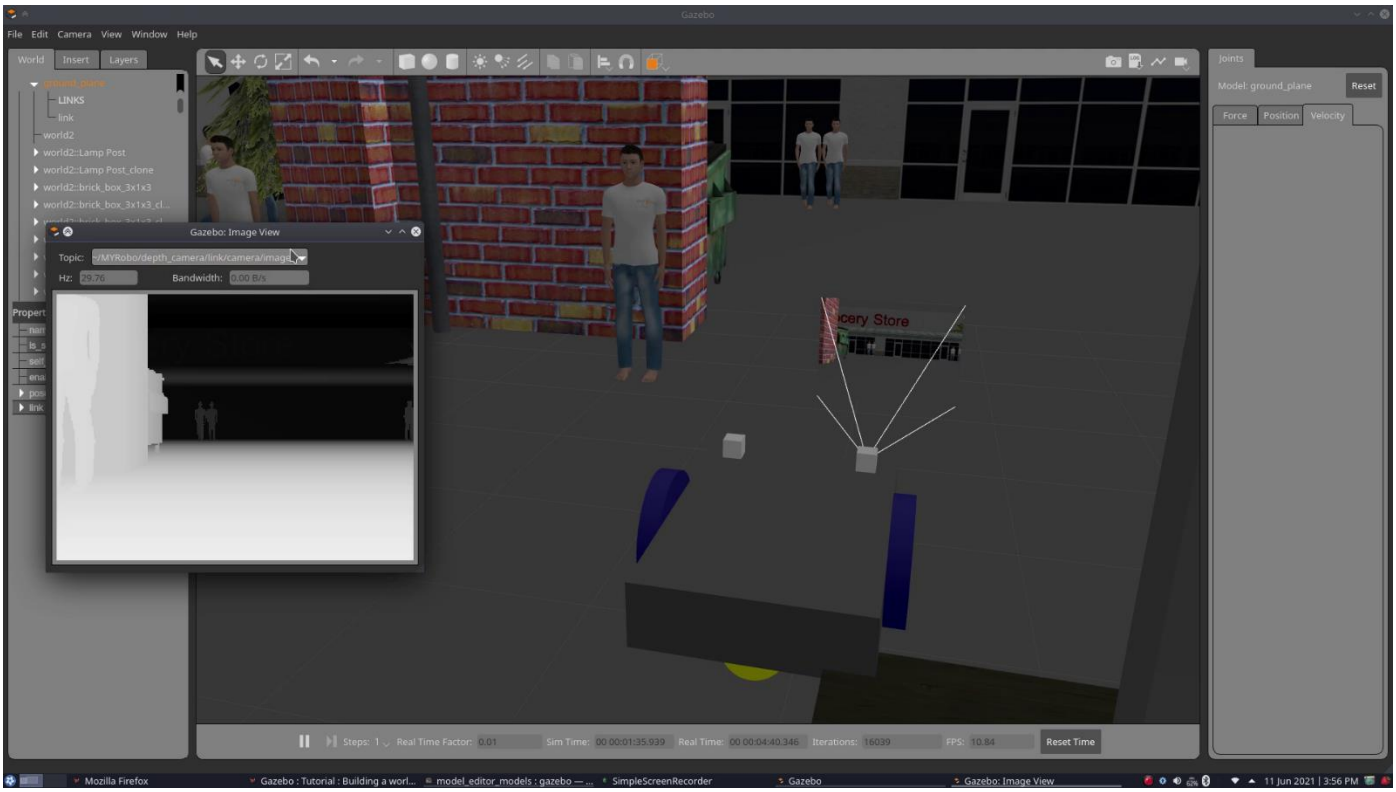

Gazebo
File Edit Camera View Window Help
World Insert Layers
Joints
Model: ground_plane_0
Reset
Force Position Velocity
Property Value
Reset Time

5.4 Sensors Output

Camera Sensor output:



Depth Camera Output:



Code Files of Robot Available here:

https://drive.google.com/drive/folders/1XJSa36Ce8CmWLc21I5NHewpdizACYaTi?usp=sharing

6. Conclusion

It was a wonderful learning experience for me while working on this project. This project took me through the various phases of Gazebo and ROS development and gave me real insight into the world of Robotics. The joy of working and the thrill involved while tackling the various problems and challenges gave me a feel of the developers' industry.

The Complete Demonstrated Video is Available here :
https://drive.google.com/file/d/1Q3_tBIvjUiuYDM5rcCLWorqpYX9E_aOD/view?usp=sharing