# ENPM690_HW3

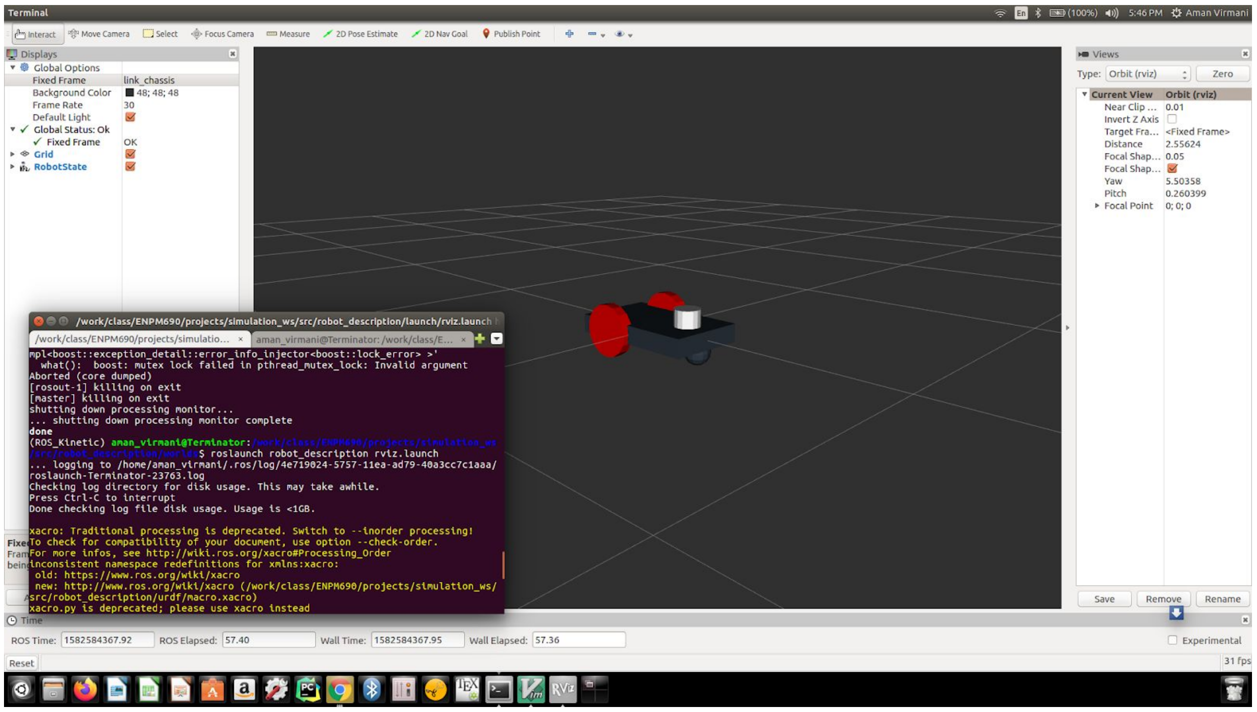
**Aman Virmani**

**Source Code:** https://github.com/AmanVirmani/mapBot

**Problem Statement : Program a simple robot vehicle in a simulated environment (robot simulation tools and libraries may be used). Your simulated robot should exhibit at least one sensor input (e.g., forward-looking range sensor that returns the distance to the nearest obstacle) and two control outputs (e.g., left and right wheels, or speed and direction of vehicle motion). Show that you can drive your robot around through mouse or keyboard inputs.**

**Procedure:**

**Step 1:** Create the URDF file. The robot that I have created uses a robot.xacro file. It also includes two other xacro files: materials.xacro and macros.xacro.
Robot.xacro : Main URDF file to describe the robot structure.
Materials.xacro : Xacro file to describe the material properties used to create the robot.
Macros.xacro : Xacro file to define the macros which are called from robot.xacro file.



| | |
|---|---|
| Global Options | |
| Fixed Frame | link_chassis |
| Background Color | ■ 48; 48; 48 |
| Frame Rate | 30 |
| Default Light | ✓ |
| Global Status: Ok | |
| Fixed Frame | OK |
| Grid | ✓ |
| RobotState | ✓ |

| | |
|---|---|
| Type: | Orbit (rviz) |
| Current View | Orbit (rviz) |
| Near Clip ... | 0.01 |
| Invert Z Axis | ☐ |
| Target Fra... | <Fixed Frame> |
| Distance | 2.55624 |
| Focal Shap... | 0.05 |
| Focal Shap... | ✓ |
| Yaw | 5.50358 |
| Pitch | 0.260399 |
| Focal Point | 0; 0; 0 |

```
/work/class/ENPM690/projects/simulation_ws/src/robot_description/launch/rviz.launch
mpl<boost::exception_detail::error_info_injector<boost::lock_error> >'
  what():  boost: mutex lock failed in pthread_mutex_lock: Invalid argument
Aborted (core dumped)
[rosout-1] killing on exit
[master] killing on exit
shutting down processing monitor...
... shutting down processing monitor complete
done
(ROS_Kinetic) aman_virmani@Terminator:/work/class/ENPM690/projects/simulation_ws
/src/robot_description/worlds$ roslaunch robot_description rviz.launch
... logging to /home/aman_virmani/.ros/log/4e719024-5757-11ea-ad79-40a3cc7c1aaa/
roslaunch-Terminator-23763.log
Checking log directory for disk usage. This may take awhile.
Press Ctrl-C to interrupt
Done checking log file disk usage. Usage is <1GB.

xacro: Traditional processing is deprecated. Switch to --inorder processing!
To check for compatibility of your document, use option --check-order.
For more infos, see http://wiki.ros.org/xacro#Processing_Order
inconsistent namespace redefinitions for xmlns:xacro:
 old: https://www.ros.org/wiki/xacro
 new: http://www.ros.org/wiki/xacro (/work/class/ENPM690/projects/simulation_ws/
src/robot_description/urdf/macro.xacro)
xacro.py is deprecated; please use xacro instead
```

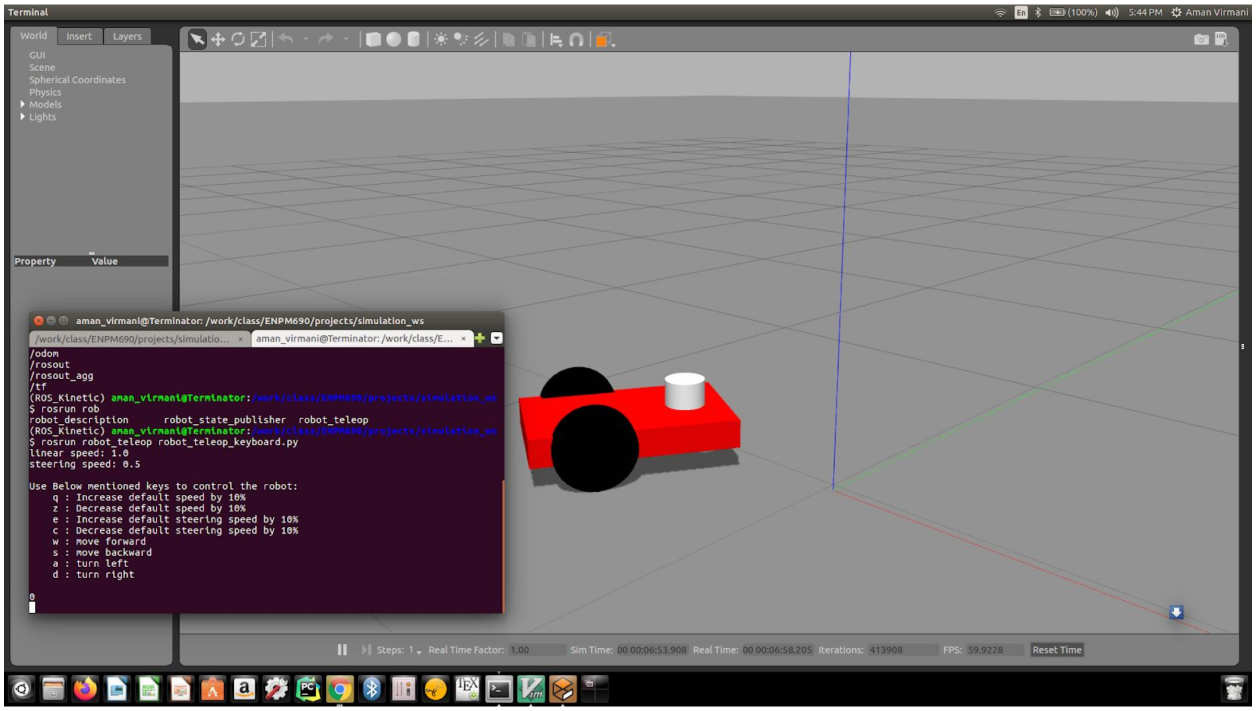
ROS Time: 1582584367.92 ROS Elapsed: 57.40 Wall Time: 1582584367.95 Wall Elapsed: 57.36

**Fig. 1 : Robot Model Visualize in Rviz**

**Step 2:** Create launch files to visualise the robot in rviz and gazebo.

**Step 3 :** Add a differential drive component to the robot to operate it using teleop. I have used the **gazebo plugin : libgazebo_ros_diff_drive.so** for the same. Velocity commands can then be published to the topic cmd_vel to control the robot

**Step 4 :** Create the robot_teleop_keyboard.py file to publish commands to control the robot in gazebo world. For testing purposes, I have used empty world file in gazebo. I have defined key mappings similar to the one used in most computer games.



```
aman_virmani@Terminator: /work/class/ENPM690/projects/simulation_ws
/odom
/rosout
/rosout_agg
/tf
(ROS_Kinetic) aman_virmani@Terminator:/work/class/ENPM690/projects/simulation_ws
$ rosrun rob
robot_description      robot_state_publisher  robot_teleop
(ROS_Kinetic) aman_virmani@Terminator:/work/class/ENPM690/projects/simulation_ws
$ rosrun robot_teleop robot_teleop_keyboard.py
linear speed: 1.0
steering speed: 0.5

Use Below mentioned keys to control the robot:
    q : Increase default speed by 10%
    z : Decrease default speed by 10%
    e : Increase default steering speed by 10%
    c : Decrease default steering speed by 10%
    w : move forward
    s : move backward
    a : turn left
    d : turn right

0
```

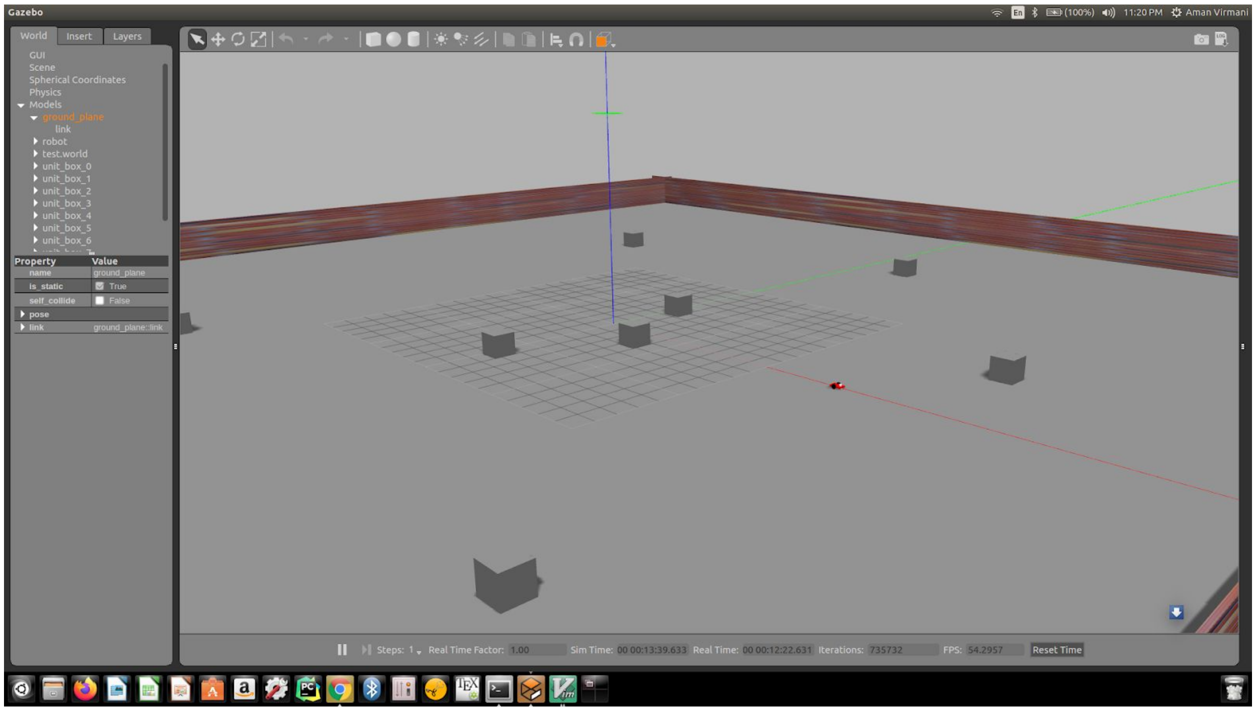
**Fig. 2 : Robot Model Visualize in Gazebo with teleop activated in terminal**

**Step 5 :** Add a laser scan sensor as input to sense the world around the robot. To get sensor data in **gazebo, libgazebo_ros_laser.so plugin** has been used. The output is published to the topic : **/mapBot/laser/scan**.

**Problem Statement: Add a programmed behavior to your robot, such as following (or avoiding) a light, or wandering, while avoiding collisions with obstacles.**

**Step 6 :** Write a python script to read laser scan data from the topic : **/mapBot/laser/scan** in the gazebo world and publish commands to **cmd_vel topic** to steer clear of potential obstacles. Logic here is simple to go straight and change path only when object is within a specific proximity threshold to avoid collision with the objects.

**Step 7 :** Create a gazebo world to test the next application of the robot, which is to map the environment using laser scan data while avoiding obstacles simultaneously.

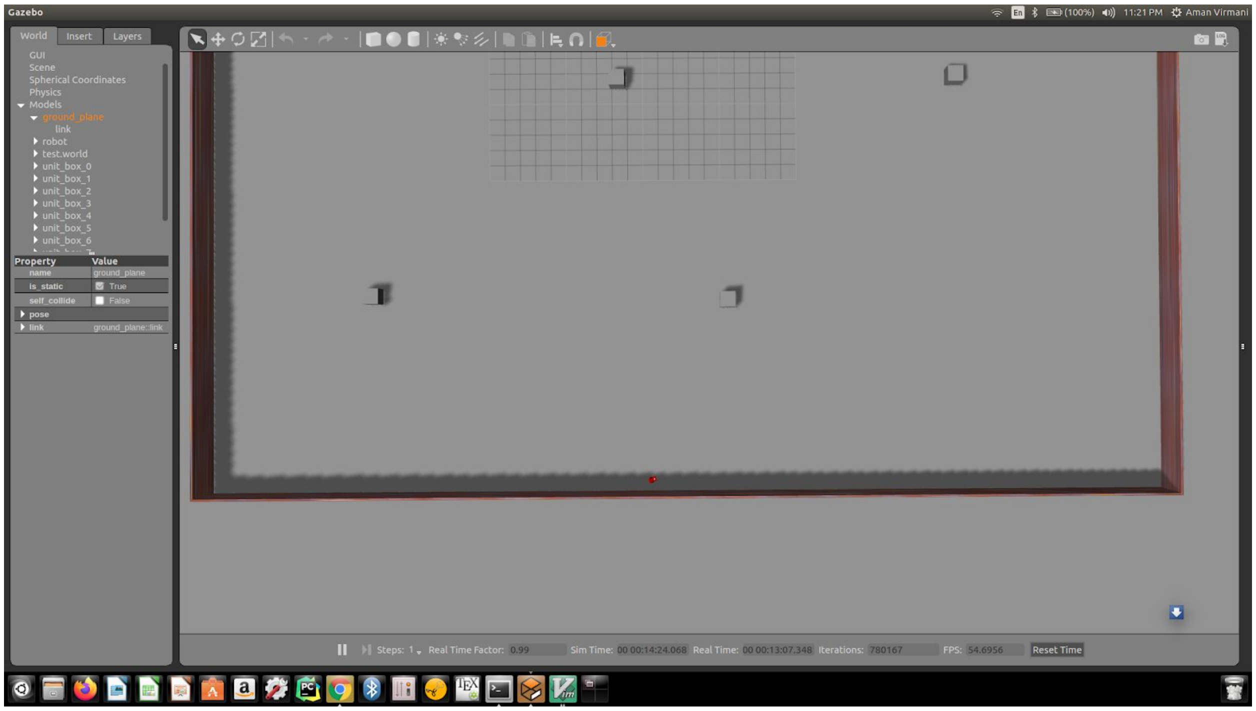


| Property | Value |
|---|---|
| name | ground_plane |
| is_static | True |
| self_collide | False |
| pose | |
| link | ground_plane::link |

Steps: 1 Real Time Factor: 1.00 Sim Time: 00 00:13:39.633 Real Time: 00 00:12:22.631 Iterations: 735732 FPS: 54.2957

**Fig. 3 : Robot Moving straight till it detects obstacles to steer away from**

**Step 8 :** Use the script in Step 6 to explore the environment created in Step 7, only this time we have added a behaviour to detect and follow the wall. The robot will move in an anticlockwise direction while moving forward till it detects a wall. Once the robot is positioned such that its front and front-left path is clear while its front-right is obstructed the robot goes into the following wall state. In this state this function is executed and this function makes the robot follow a straight line.



| Property | Value |
|---|---|
| name | ground_plane |
| is_static | ✓ True |
| self_collide | False |
| pose | |
| link | ground_plane::link |

```
Steps: 1  Real Time Factor: 0.99  Sim Time: 00 00:14:24.068  Real Time: 00 00:13:07.348  Iterations: 780167  FPS: 54.6956
```

Fig. 4 : Robot Model(in red color) following the wall closely