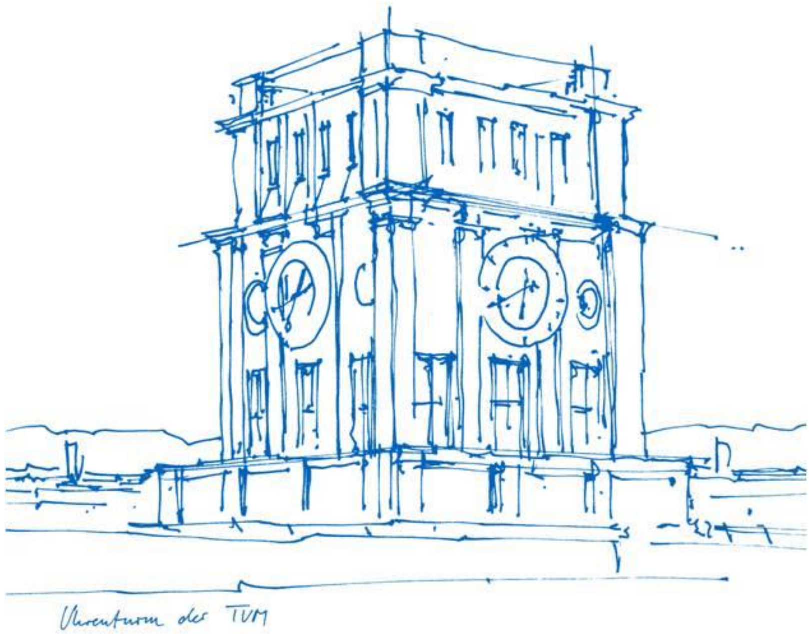


MathWorks®

Accelerating the pace of engineering and science

AUTOMATED DRIVING SYSTEMS CO-SIMULATING CARLA WITH SIMULINK USING ROS

Contents

List of Figures

Introduction

Carla is an open source simulator for autonomous driving research, providing development, training and validation of autonomous urban driving systems and in order to keep this experience similar to reality, it provides urban layouts, buildings and pedestrians. This document will explain thoroughly how to create the Carla-Ros interface and being capable of connecting Carla to MATLAB and Simulink- fig (1) and will go through the installation steps of the required softwares.

Figure 1: Connecting Carla to MATLAB & Simulink through ROS

Specifications

Operating system

- Linux-Ubuntu 16.04 LTS (Xenial Xerus)

Hardware Specifications

- 64-bit PC
- Intel Core i7-7700 hq (7th Generation Processor)
- Nvidia GTX-1050 4GB (10th Generation Graphics Card)
- 16 GB RAM

Installation Guidelines

In order to build Carla, it requires Ubuntu 16.04 or later versions. The first Step is

Build tools and dependencies installation

(1)
```
sudo apt-get update
```

(2)
```
sudo apt-get install wget software-properties-common
```

(3)
```
sudo add-apt-repository ppa:ubuntu-toolchain-r/test
```

(4)
```
wget -O - https://apt.llvm.org/llvm-snapshot.gpg.key|sudo apt-key add -
```

(5)
```
sudo apt-add-repository "deb http://apt.llvm.org/xenial/ llvm-toolchain-xenial-7 main"
```

(6)
```
sudo apt-get update
```

(7)
```
sudo apt-get install build-essential clang-7 lld-7 g++-7 cmake ninja-build libvulkan1 python
python-pip python-dev python3-dev python3-pip libpng16-dev libtiff5-dev libjpeg-dev tzdata
sed curl unzip autoconf libtool rsync
```

(8)
```
pip2 install --user setuptools
```

(9)
```
pip3 install --user setuptools
```

Unreal Engine Installation

After installing the dependencies, the second step is to install Unreal Engine, but in order to but in order to avoid compatibility issues between its dependencies and Carla's, the ideal scenario is to compile everything with the same compiler version and C++ runtime library

(1)

```
sudo update-alternatives --install /usr/bin/clang++ clang++ /usr/lib/llvm-7/bin/clang++ 170
```

(2)

```
sudo update-alternatives --install /usr/bin/clang clang /usr/lib/llvm-7/bin/clang 170
```

Adding GitHub Account

In order to be capable of cloning Unreal Engine from the repository, which is set to private, adding the GitHub account is necessary when signing up on Unreal Engine:

- ➢ Sign in using Unreal Engine Account

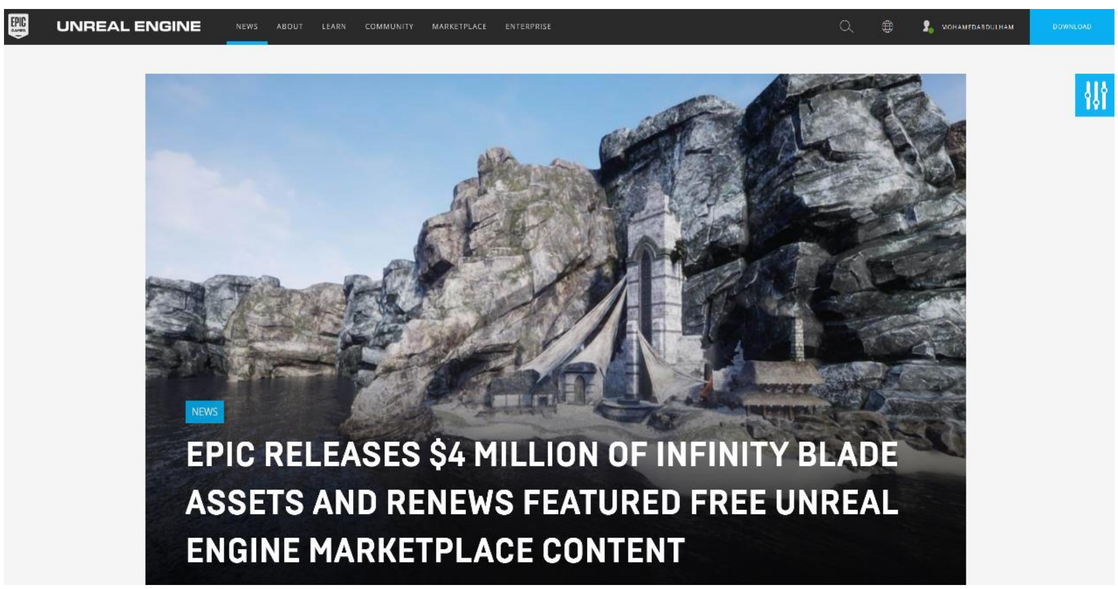


Figure 2:Unreal Engine Log-in Page

- Go to Personal
- Choose connected Accounts

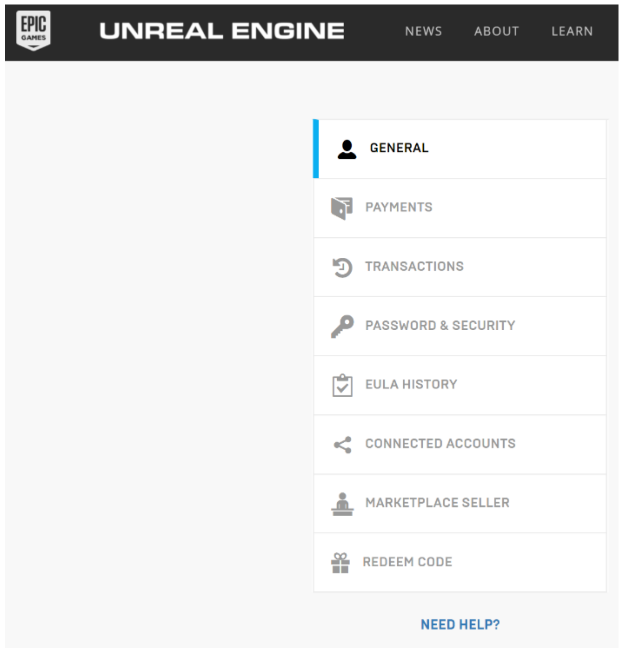


Figure 3:Connected Accounts

- Finally, connect to GitHub Account

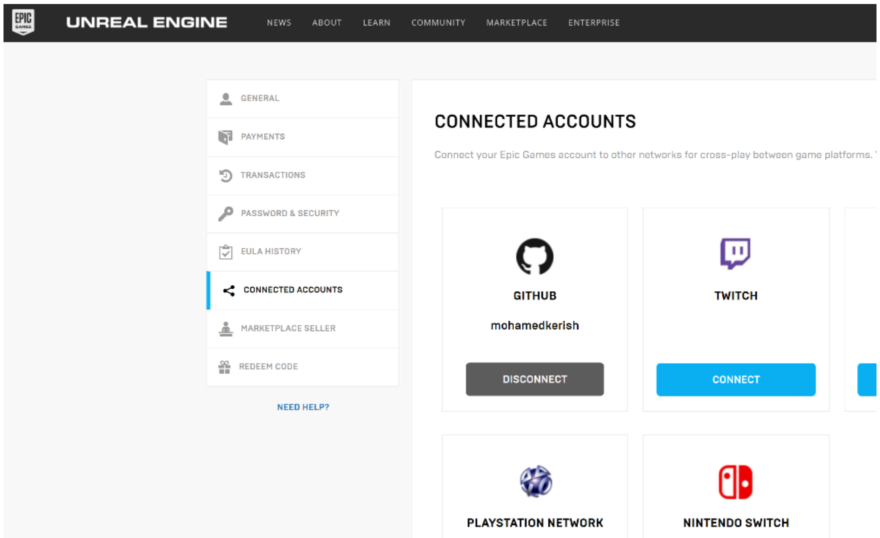


Figure 4: Connect to GitHub Account

Downloading and Compiling Unreal Engine version 4.22

Installing Unreal Engine requires a minimum of 8 GB of RAM and here are the corresponding commands

(1)

```
git clone --depth=1 -b 4.22 https://github.com/EpicGames/UnrealEngine.git ~/UnrealEngine_4.22
```

(2)

```
cd ~/UnrealEngine_4.22
```

(3)

```
./Setup.sh && ./GenerateProjectFiles.sh && make
```

ROS Installation

The ROS version that would be installed is ROS-Kinetic, which only supports Ubuntu 16.04 Xenial. The installation and compilation steps are as follows:

Setting-up your computer to accept software from packages.ros.org.

(1)

```
sudo sh -c 'echo "deb http://packages.ros.org/ros/ubuntu $(lsb_release -sc) main" > /etc/apt/sources.list.d/ros-latest.list'
```

Setup your keys

(2)

```
sudo apt-key adv --keyserver 'hkp://keyserver.ubuntu.com:80' --recv-key C1CF6E31E6BADE8868B172B4F42ED6FBAB17C654
```

Installation

(3)

```
sudo apt-get update
```

Desktop-Full Install

This one is recommended, as it includes ROS, rqt, rviz, robot-generic libraries, 2D/3D simulators, navigation and 2D/3D perception

(4)

```
sudo apt-get install ros-kinetic-desktop-full
```

ROSdep Initialization

This would enable easy installation of system dependencies and to run some core components in ROS

(5)

```
sudo rosdep init
```

(6)

```
rosdep update
```

Environment Setup

This would automatically add the ROS environment variables to the bash session every time a new shell is launched

(7)

```
echo "source /opt/ros/kinetic/setup.bash" >> ~/.bashrc
```

(8)

```
source ~/.bashrc
```

Extra Dependencies Installation

Until now, the needed dependencies were installed to run the Core ROS packages, yet to create and manage your own workspaces, various tools and requirements that are distributed separately need to be installed, so to install these dependencies

(9)

```
sudo apt install python-rosinstall python-rosinstall-generator python-wstool build-essential
```

Carla-Ros Bridge package Installation

The installed ROS package aims at providing a simple ROS-bridge for Carla Simulator and the setup must be done through creating a catkin Workspace and installing Carla-Ros Bridge package. It is advisable to do the whole compilation inside the **Unreal Engine_4.22** file

Setup folder structure

(1)
```
mkdir -p ~/carla-ros-bridge/catkin_ws/src
```

(2)
```
cd ~/carla-ros-bridge
```

(3)
```
git clone https://github.com/carla-simulator/ros-bridge.git
```

(4)
```
cd catkin_ws/src
```

(5)
```
ln -s ../../ros-bridge
```

(6)
```
source /opt/ros/kinetic/setup.bash
```

(7)
```
cd ..
```

Install Dependencies

Make sure one more time that all the required ROS dependencies were installed

(8)
```
rosdep update
```

(9)
```
rosdep install --from-paths src --ignore-src -r
```

Build

(10)

```
catkin_make
```

Carla Installation

The final procedure is to clone Carla from GitHub and install it. Doing the whole compilation inside the Unreal Engine_4.22 file is necessary

Clone from Repository

(1)

```
git clone https://github.com/carla-simulator/carla
```

(2)

```
sudo apt-get install aria2
```

The following command would take so much time, so it is advisable to run this line- *command (2)*- at first.

Update Carla

The next step is to open the cloned Carla file from terminal and run this command

(3)

```
./Update.sh
```

Set Environment Variable

It must be set in order for Carla to find Unreal Engine’s installation folder

(4)

```
export UE4_ROOT=~/UnrealEngine_4.22
```

Compile the simulator and launch Unreal Engine's Editor

(5) make launch

If the Carla-Unreal Editor starts to initialize and suddenly, this error shows up- fig (5)

Figure 5: Incompatible Vulcan Driver Error

it is advisable in this case to go to **Additional Drivers,** and select the Nvidia Proprietary Driver instead of the Nouveau open-source Driver, as shown in fig (6)

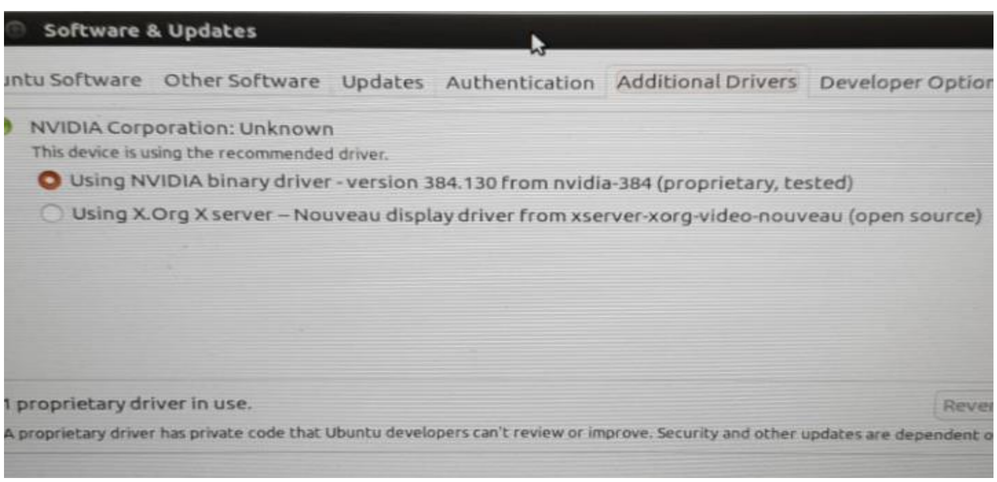


Figure 6: Additional Drivers in Ubuntu 16.04 LTS

Then type in the ***make launch*** command again, and the Carla-Unreal Editor will start the Initialization (it will take an hour from this process to finish)

Compile Python API

Run the following command to compile the Python API module responsible for running the Python examples

(6) make PythonAPI

Create a packaged version

The last step and the most important one is to compile everything and create a packaged version able to run without UE4 Editor

(7) make package

This command will take up to two hours to finish and in order to make sure that everything was compiled successfully, this should be the final message, fig (7)

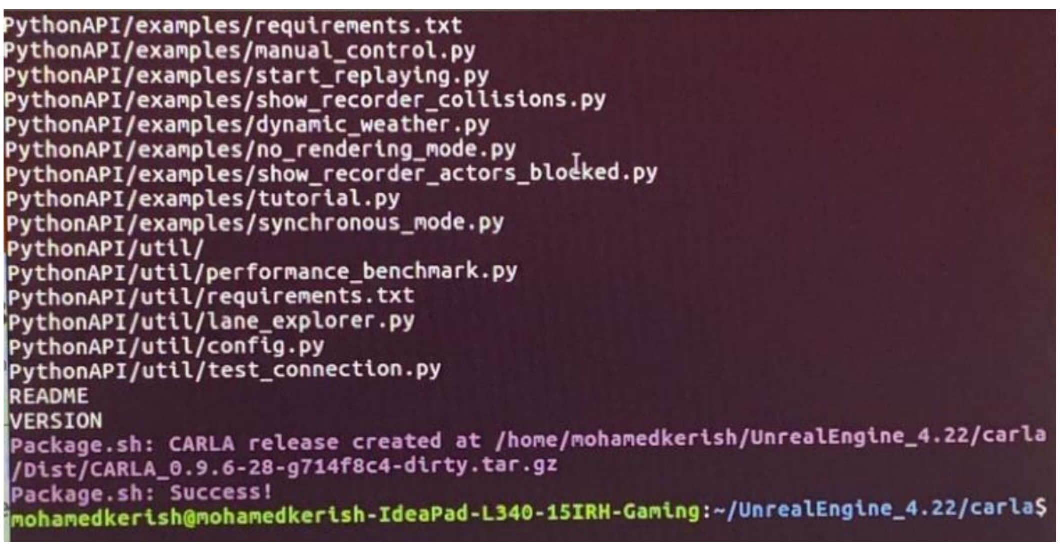


Figure 7: Package. sh: Success

Carla-ROS Bridge

Start the ROS Bridge

Run the Simulator

- Open the Unreal Engine File → Carla → Unreal → CarlaUE4 → LinuxNoEditor

- Open the terminal from the LinuxNoEditor and run the simulator inside it using the following command

(1)

```
./CarlaUE4.sh -windowed -ResX=320 -ResY=240
```

Where ResX and ResY represents the dimensions of the window, fig (8)

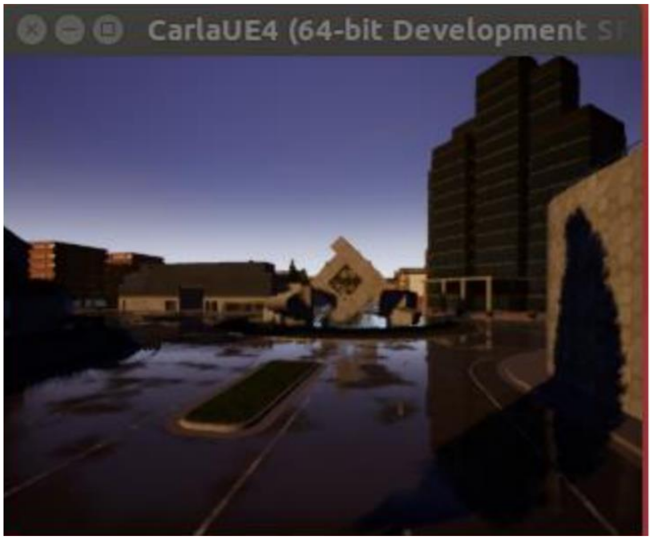


Figure 8: Carla UE4 Window

Egg-File Location

Complete path to the egg-file including the egg-file itself, it is located inside the ***dist file*** inside ***Python API***

Unreal Engine File → Carla → Unreal → Python API → Carla → dist

(2)

```
export PYTHONPATH=$PYTHONPATH: <path/to/carla/>/PythonAPI/<your_egg_file>
```

(3)

```
source ~/carla-ros-bridge/catkin_ws/devel/setup.bash
```

ROS Options

(a) Start the ROS-bridge

```
roslaunch carla_ros_bridge carla_ros_bridge.launch
```

(b) Start the ROS-bridge together with RVIZ

```
roslaunch carla_ros_bridge carla_ros_bridge_with_rviz.launch
```

(c) Start the ROS-bridge together with an example ego vehicle

```
roslaunch carla_ros_bridge carla_ros_bridge_with_example_ego_vehicle.launch
```

For ROS installation verification purposes, the ROS-bridge together with RVIZ (second option) was chosen to carry-out the upcoming example. After making the ROS connection, the RVIZ window will open up and inside, camera images, sensor data and odometry of information of every vehicle can be reached and can be compared with the carla simulator in terms of accuracy.

Verification Case

In order to create a real case simulation, Python API examples are used to add sensors and spawn characters, for example, vehicles, and pedestrians. From Examples, open terminal:

Unreal Engine File → Carla → Unreal → Python API → Examples

(4)

```
./spawn_npc.py -n (number of spawned characters)
```

n 80 was used in this example.

If the pygame dependency did not work out, run the following command

(5)

```
sudo apt-get install python-pygame
```

Then, run command (4) again

Python API Example

Finally, run any of the python API examples inside a new terminal and manual control was chosen in this case

(6)

```
./manual_control.py
```

Finally, the pygame window will appear, which allows for the change of sensors and manual control by the user and on the top the Frames per Second for the server and the client are shown, fig (9)

Figure 9: Pygame Window

```
Use ARROWS or WASD keys for control.

    W            : throttle
    S            : brake
    AD           : steer
    Q            : toggle reverse
    Space        : hand-brake
    P            : toggle autopilot
    M            : toggle manual transmission
    ,/.          : gear up/down

    TAB          : change sensor position
    `            : next sensor
    [1-9]        : change to sensor [1-9]
    C            : change weather (Shift+C reverse)
    Backspace    : change vehicle
```

Figure 10: Manual Control and Sensor change Keys

As shown in fig (10), the autopilot mode (Autonomous Driving) can be activated through pressing P and accordingly, camera images are accessible on RVIZ window and accordingly, the Carla-ROS connection is obtained, fig (11)

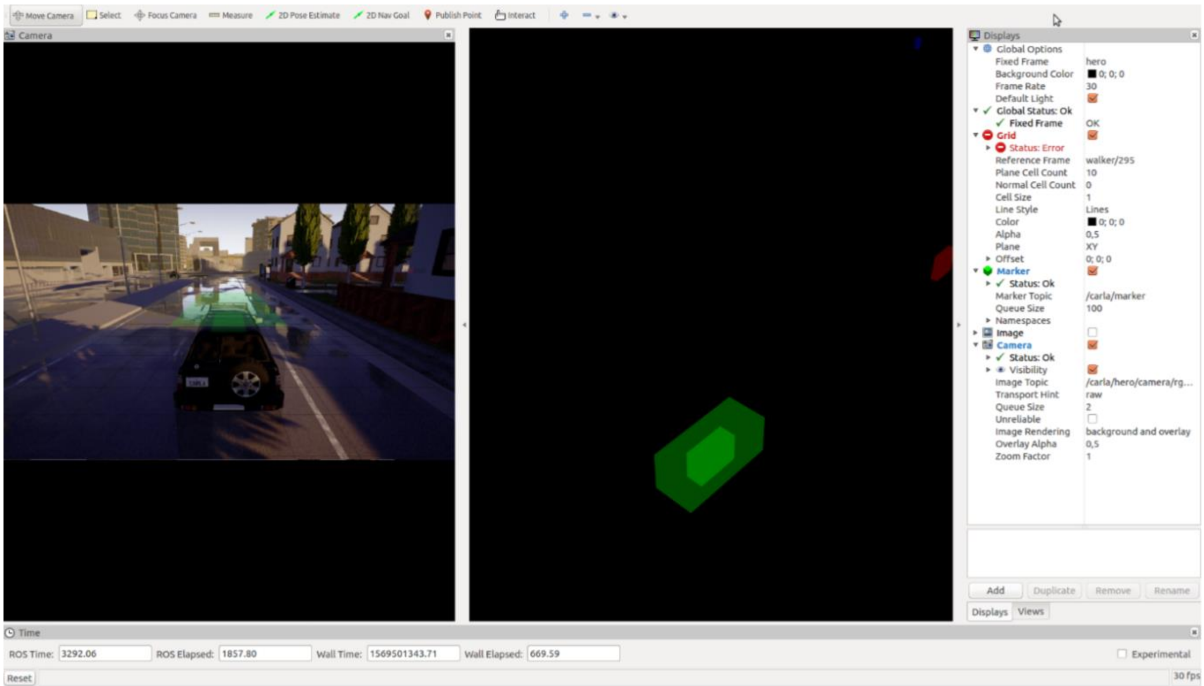


Figure 11: RVIZ Window with Camera Images from Carla

MATLAB-ROS

In the preceding part, the bridge was there to connect between the Carla world and ROS. In the upcoming part, the focus would be on creating a connection between MATLAB and ROS in order to send data to Simulink using the ROS Custom Messages Interface.

Required add-ons Installation

ROS Toolbox

For MATLAB and Simulink users of version R2019B, a new version of ROS Toolbox has been created, providing an interface connecting MATLAB and Simulink with ROS, enabling the user to create a network of ROS nodes.

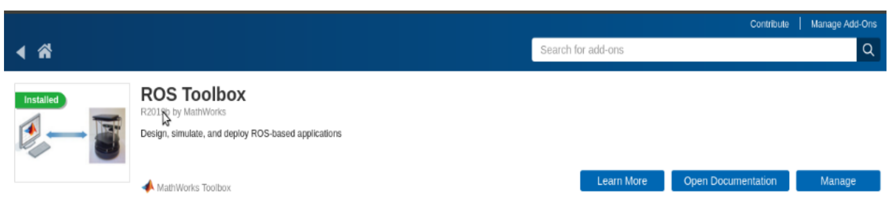


Figure 12: ROS Toolbox Installation

ROS Toolbox interface for ROS Custom Messages

The Custom Messages Interface allows the user to define his own custom ROS message within MATLAB and Simulink to communicate with other nodes in the ROS network.

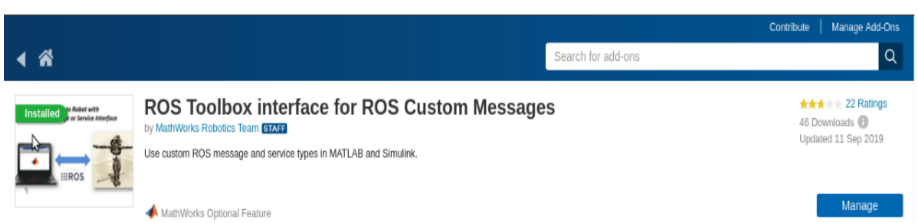


Figure 13: ROS Toolbox interface for ROS Custom Messages

Create Custom Messages for Carla

Install necessary files to create Custom Messages

- Clone the Carla_msgs file from Gitlab repository and unzip it inside ROS-bridge folder

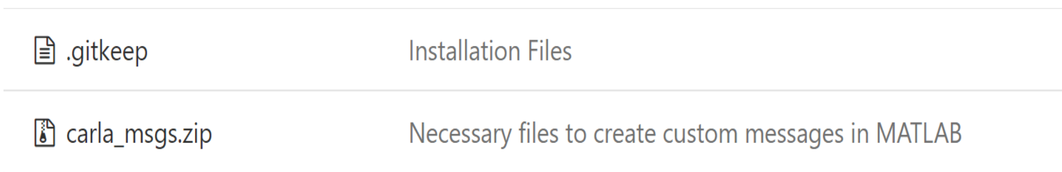


Figure 14: Clone necessary files from Gitlab

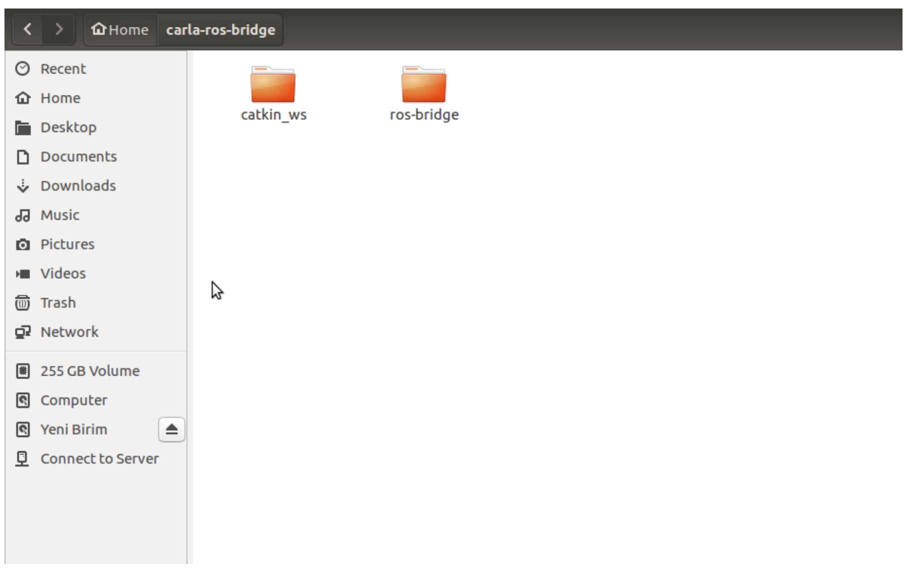


Figure 15: ros-bridge file

- In order to create custom messages for Carla, the *rosgenmsg* command, which is a function of ROS toolbox interface for ROS custom messages and inside the command type in the address of the ROS bridge directory and accordingly, ***matlab_gen*** file is created inside ROS bridge folder

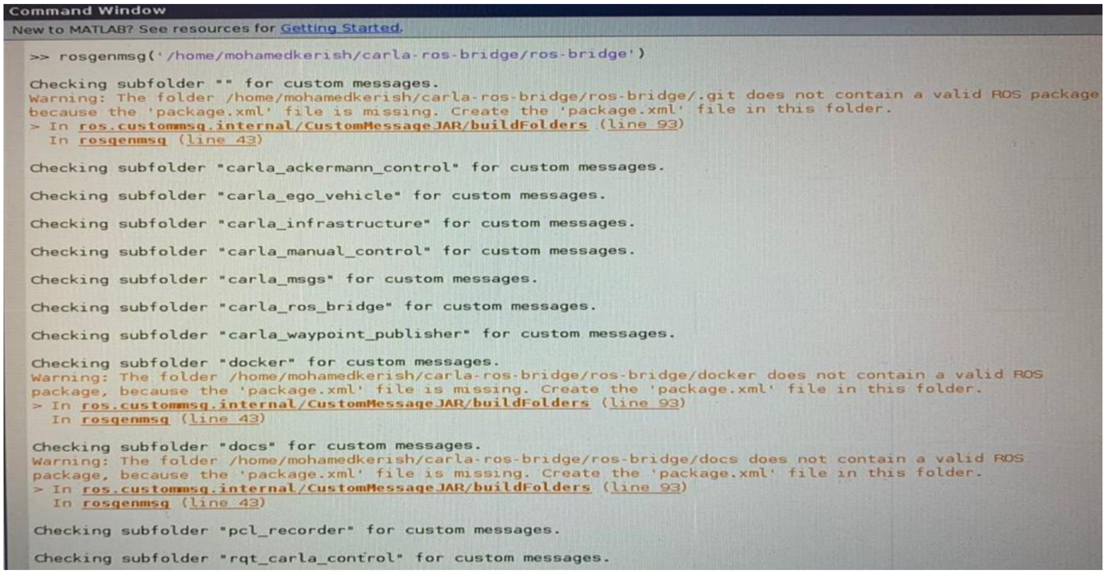


Figure 16: rosgenmsg command in MATLAB

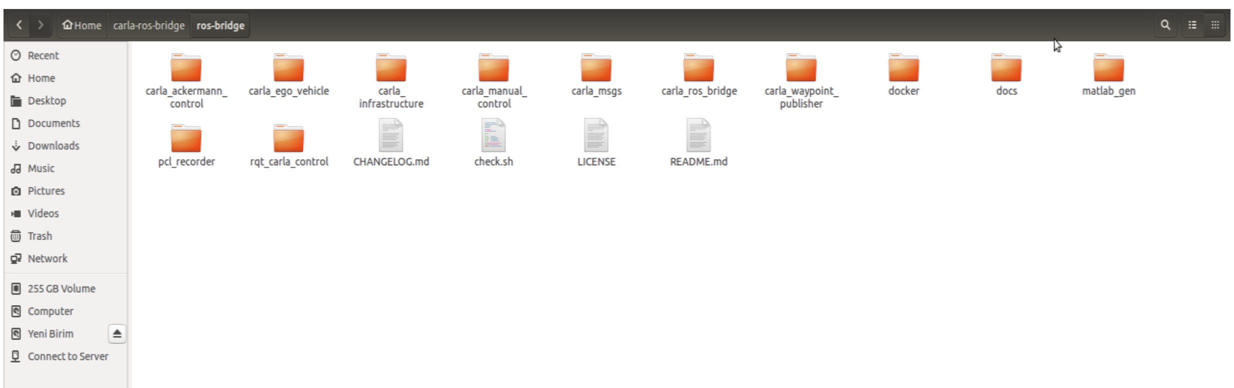
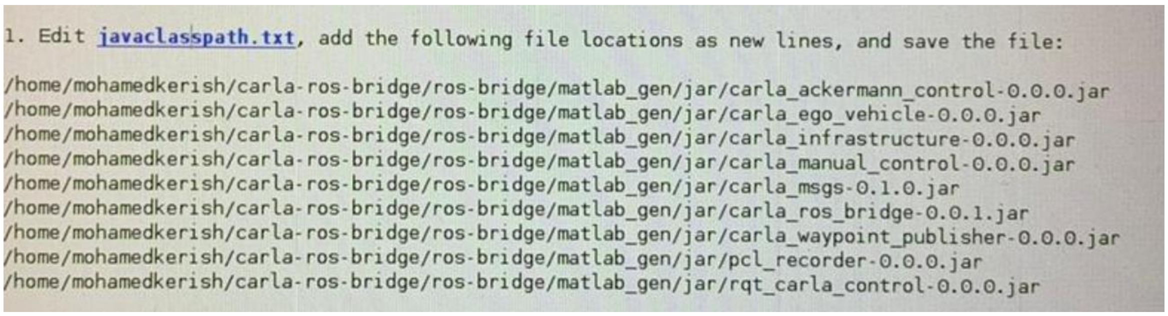


Figure 17: matlab_gen file

Steps to use Custom Messages

- Edit ***javaclasspath.txt*** and add the following file locations as new lines and save the file in the end



1. Edit javaclasspath.txt, add the following file locations as new lines, and save the file:

```
/home/mohamedkerish/carla-ros-bridge/ros-bridge/matlab_gen/jar/carla_ackermann_control-0.0.0.jar
/home/mohamedkerish/carla-ros-bridge/ros-bridge/matlab_gen/jar/carla_ego_vehicle-0.0.0.jar
/home/mohamedkerish/carla-ros-bridge/ros-bridge/matlab_gen/jar/carla_infrastructure-0.0.0.jar
/home/mohamedkerish/carla-ros-bridge/ros-bridge/matlab_gen/jar/carla_manual_control-0.0.0.jar
/home/mohamedkerish/carla-ros-bridge/ros-bridge/matlab_gen/jar/carla_msgs-0.1.0.jar
/home/mohamedkerish/carla-ros-bridge/ros-bridge/matlab_gen/jar/carla_ros_bridge-0.0.1.jar
/home/mohamedkerish/carla-ros-bridge/ros-bridge/matlab_gen/jar/carla_waypoint_publisher-0.0.0.jar
/home/mohamedkerish/carla-ros-bridge/ros-bridge/matlab_gen/jar/pcl_recorder-0.0.0.jar
/home/mohamedkerish/carla-ros-bridge/ros-bridge/matlab_gen/jar/rqt_carla_control-0.0.0.jar
```

Figure 18: Edit javaclasspath.txt

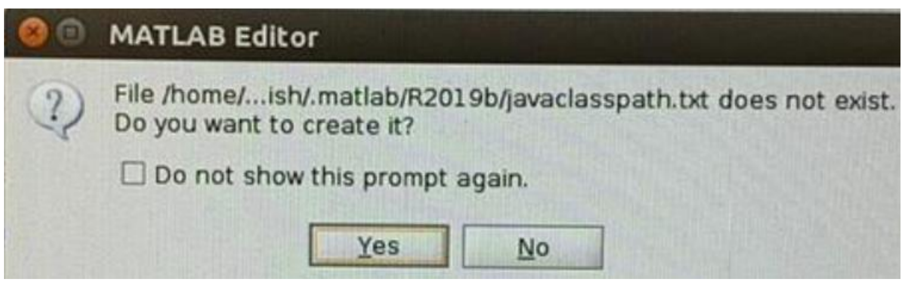


Figure 19: Create new file

Editor - /home/mohamedkerish/.matlab/R2019b/javaclasspath.txt*

javaclasspath.txt*

```
/home/mohamedkerish/carla-ros-bridge/ros-bridge/matlab_gen/jar/carla_ackermann_control-0.0.0.jar
/home/mohamedkerish/carla-ros-bridge/ros-bridge/matlab_gen/jar/carla_ego_vehicle-0.0.0.jar
/home/mohamedkerish/carla-ros-bridge/ros-bridge/matlab_gen/jar/carla_infrastructure-0.0.0.jar
/home/mohamedkerish/carla-ros-bridge/ros-bridge/matlab_gen/jar/carla_manual_control-0.0.0.jar
/home/mohamedkerish/carla-ros-bridge/ros-bridge/matlab_gen/jar/carla_msgs-0.1.0.jar
/home/mohamedkerish/carla-ros-bridge/ros-bridge/matlab_gen/jar/carla_ros_bridge-0.0.1.jar
/home/mohamedkerish/carla-ros-bridge/ros-bridge/matlab_gen/jar/carla_waypoint_publisher-0.0.0.jar
/home/mohamedkerish/carla-ros-bridge/ros-bridge/matlab_gen/jar/pcl_recorder-0.0.0.jar
/home/mohamedkerish/carla-ros-bridge/ros-bridge/matlab_gen/jar/rqt_carla_control-0.0.0.jar
```

Figure 20: Added file locations

- The next step is to add the Custom Message folder to the MATLAB path through executing the following command lines in MATLAB, as shown in fig (21)

```
>> addpath('/home/mohamedkerish/carla-ros-bridge/ros-bridge/matlab_gen/msggen')
savepath
```

Figure 21: Addedpath command

However, while executing this command, an error message showed up, fig (22)

```
Warning: Unable to save path to file '/usr/local/MATLAB/R2019b/toolbox/local/pathdef.m'. You can save your
path to a different location by calling SAVEPATH with an input argument that specifies the full path. For
MATLAB to use that path in future sessions, save the path to 'pathdef.m' in your MATLAB startup folder.
> In savepath (line 176)
```

Figure 22: Error message

So, in order to avoid such an error, the user has to the properties of the ***pathdef.m*** file through using this command inside the terminal

(1) sudo nautilus

which will allow the user to open a file explorer with root access, giving him the permission to change the properties of the file from read only to read and write and the steps are as follows:

```
mohamedkerish@mohamedkerish-IdeaPad-L340-15IRH-Gaming:~$ sudo nautilus
[sudo] password for mohamedkerish:
```

Figure 23: Nautilus command in Terminal

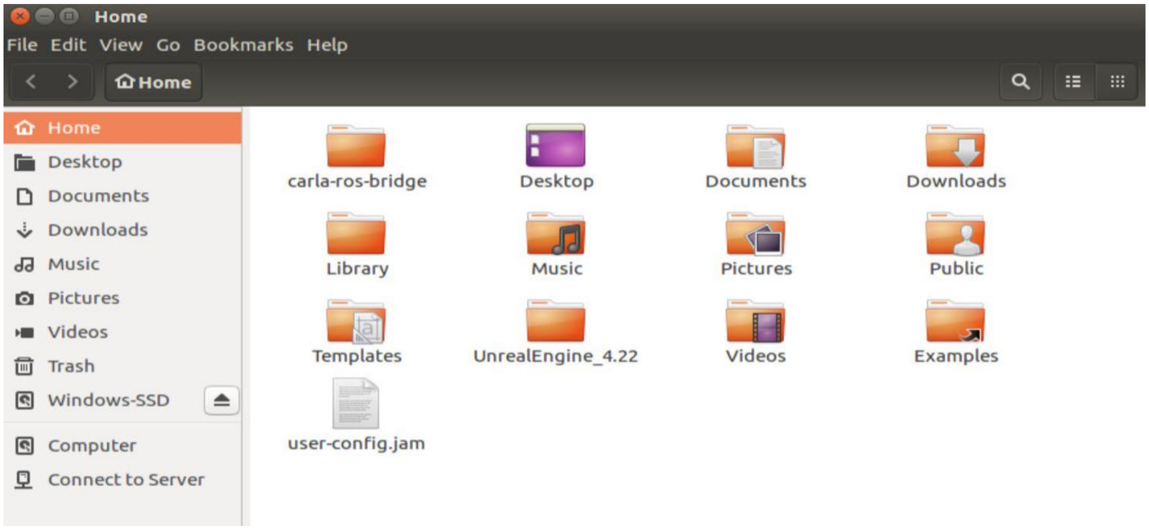


Figure 24: New file explorer

usr → local → MATLAB → R2019b → toolbox → local → pathdef.m

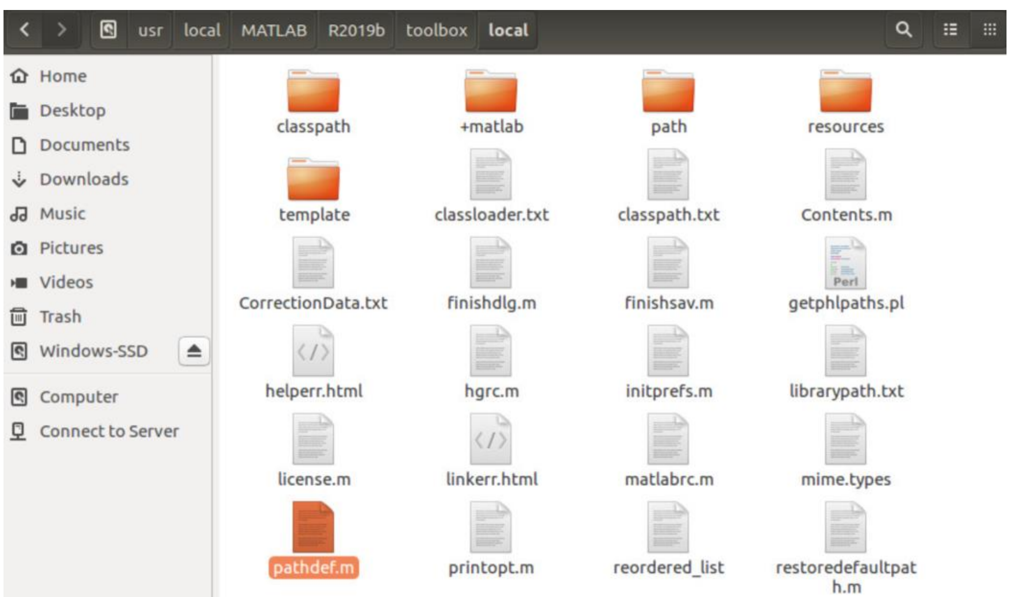


Figure 25: Choose the pathdef.m file

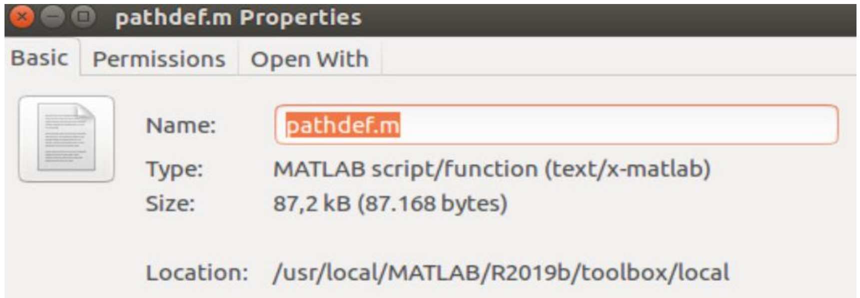


Figure 26: Choose Permissions

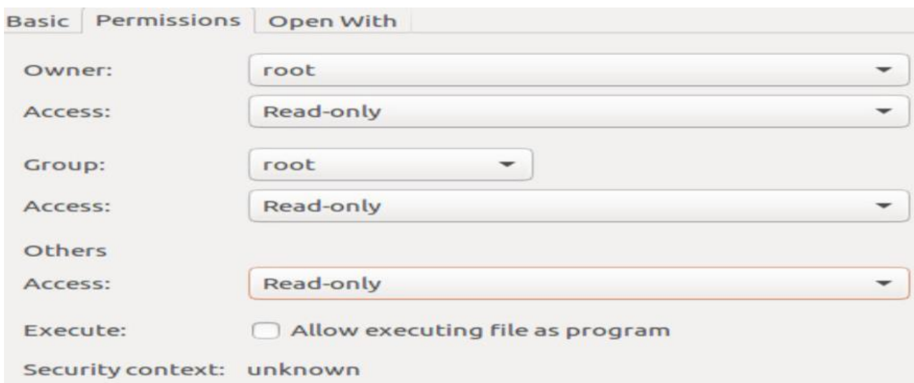


Figure 27: Read-only acccess

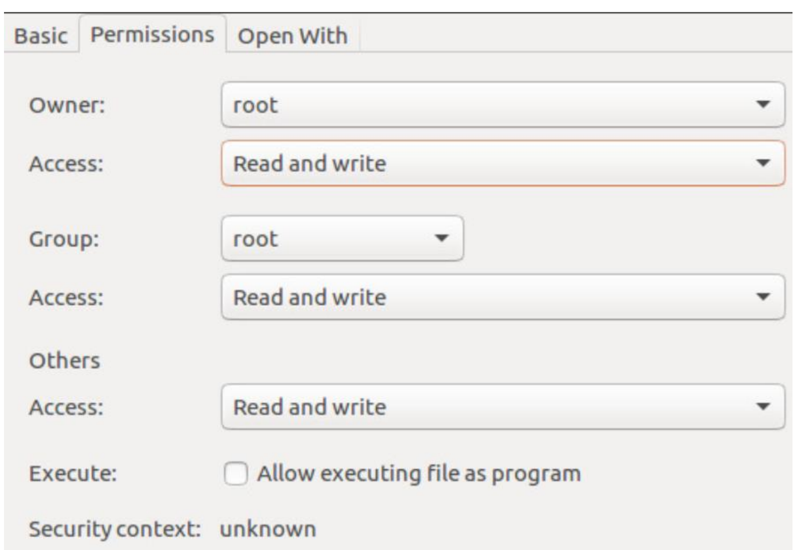


Figure 28: Read and Write access

After executing the previous steps, use the ***addpath*** command again and in order to make sure that the Custom Messages for Carla were added in MATLAB use the following command, fig (29)

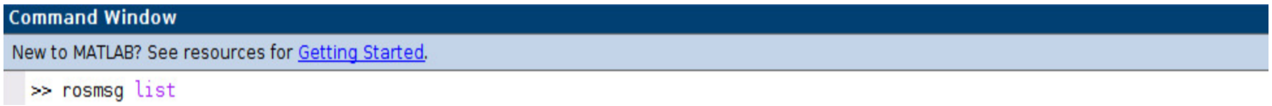


Figure 29: rosmsg list command

If Carla messages are seen in that list, user should see these messages in the Simulink ROS blocks.

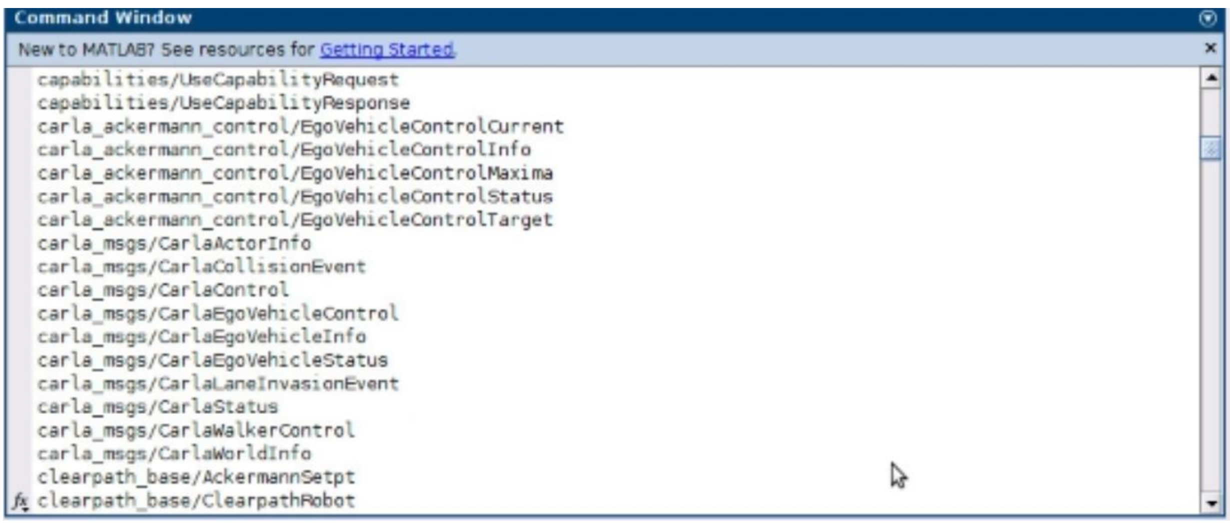


Figure 30: Carla messages in the list

The next step is to check the ROS blocks inside Simulink

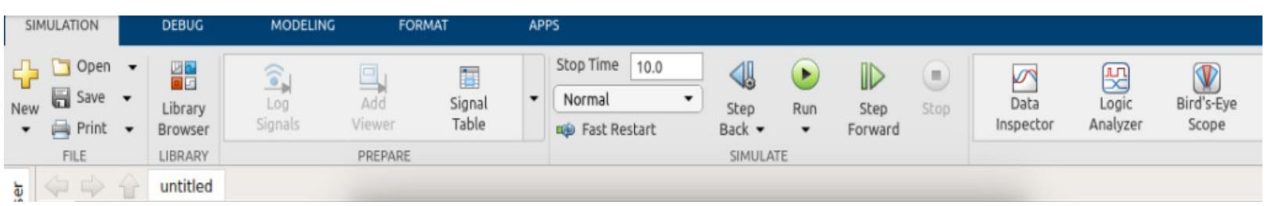


Figure 31: Choose Library Browser

And accordingly, the ROS blocks will show up and can be selected inside the Simulink project as shown in fig (32)

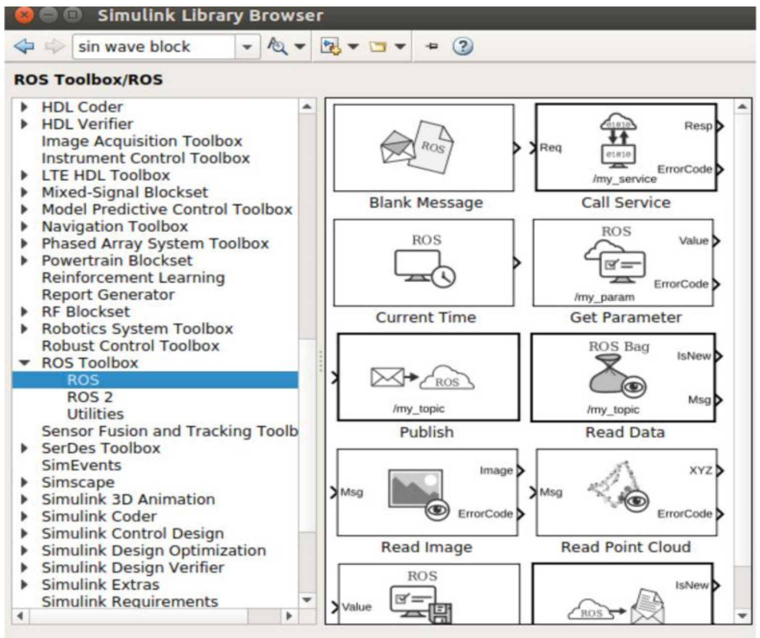


Figure 32: Select ROS

Types of Blocks

The following ROS blocks would be mainly used in order to extract Sensor Data and images from Carla and model them in Simulink

- Subscribe Block is mainly used to receive messages from ROS network

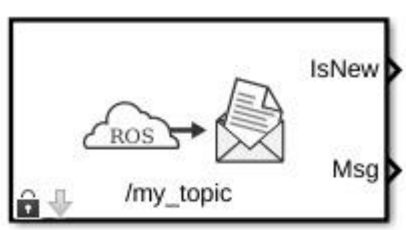


Figure 33: Subscribe block

- Read Point Cloud block is used to extract point cloud data from ROS point cloud message (LIDAR)

Figure 34: Read Point Cloud Block

- Publish block is used to send messages to a ROS network

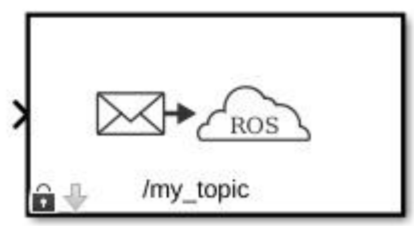


Figure 35: Publish Block

- Read Image block is used to extract image signal from ROS image messages (RGB, Semantic Segmentation and Depth)

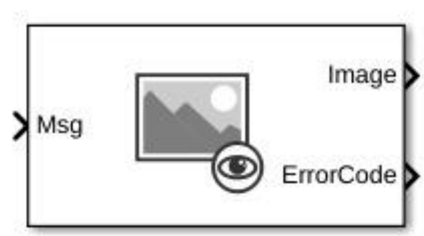


Figure 36: Read Image Block

Proof of Concept

This example is set in order to make sure that the interface between Carla and Simulink is plausible and Sensor Data can be exchanged and modelled on Simulink using the ROS bridge via ROS Custom Messages. In this example, most of the procedures that were written in the ***Carla-ROS Bridge*** section would be executed and the ROS Blocks, shown in the ***Create Custom Messages for Carla*** section will be used in order to create a model on Simulink.

At first, the steps done in the ***Start ROS Bridge*** part have to be done:

- Run the Simulator
- Locate the Egg-file
- Choose one of the ROS options and for this proof of concept, the third option would be chosen *'Start the ROS-bridge together with an example ego vehicle'*
- Directly, after following these steps, the Carla world would be initialized with the Ego-Vehicle being there, as shown in fig (37)

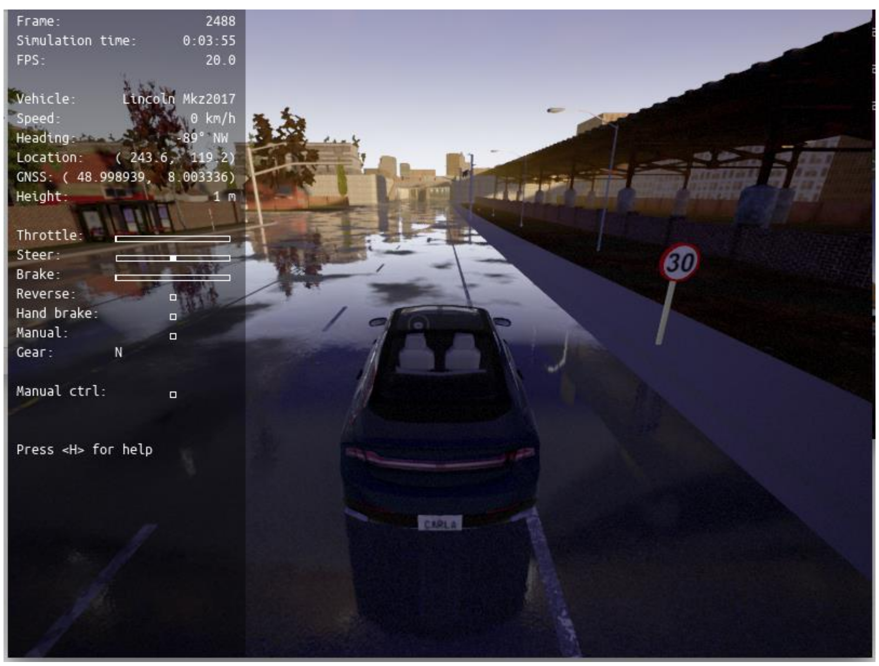


Figure 37: Carla-ROS window

- Spawn Vehicles and characters to have more of a dynamic environment

- Customization of the number of sensors, their types and their locations on the vehicle from the ***Sensors.json*** file

Carla-ros-bridge ⟶ ros-bridge ⟶ carla_ego_vehicle ⟶ config ⟶ sensors.json


```
    "sensors": [
        {
            "type": "sensor.camera.rgb",
            "id": "front",
            "x": 2.0, "y": 0.0, "z": 2.0, "roll": 0.0, "pitch": 0.0, "yaw": 0.0,
            "width": 800,
            "height": 600,
            "fov": 100,
            "sensor_tick": 0.05
        },
        {
            "type": "sensor.camera.rgb",
            "id": "view",
            "x": -4.5, "y": 0.0, "z": 2.8, "roll": 0.0, "pitch": -20.0, "yaw": 0.0,
            "width": 800,
            "height": 600,
            "fov": 100,
            "sensor_tick": 0.05
        },
        {
            "type": "sensor.lidar.ray_cast",
            "id": "lidar1",
            "x": 0.0, "y": 0.0, "z": 2.4, "roll": 0.0, "pitch": 0.0, "yaw": 0.0,
            "range": 5000,
            "channels": 32,
            "points_per_second": 320000,
            "upper_fov": 2.0,
            "lower_fov": -26.8,
            "rotation_frequency": 20,
            "sensor_tick": 0.05
        },
        {
            "type": "sensor.other.gnss",
            "id": "gnss1",
            "x": 1.0, "y": 0.0, "z": 2.0
        },
        {
            "type": "sensor.other.collision",
            "id": "collision1",
            "x": 0.0, "y": 0.0, "z": 0.0
        },
        {
            "type": "sensor.other.lane_invasion",
            "id": "laneinvasion1",
            "x": 0.0, "y": 0.0, "z": 0.0
        }
    ]
}
```


Figure 38: Types of sensors in Carla

After following these steps, the Carla world is set-up and now interfacing with the ROS and the next step is to connect this world to Simulink using the ROS blocks to create a co-simulation model in which sensor data and can obtained and visualized while the vehicle is moving autonomously.

- The first model created is the Vehicle Control model, fig (39), it consists of several separate building blocks that have several functionalities in order to obtain a certain output, for example, Point cloud data from Lidar, RGB images and Semantic Segmentation from Camera Sensor, while being capable of shifting between Manual and Automatic Control through enabling either Autopilot or Manual Control. In order, to understand each model separately, the user has to double click any of these models to see their building blocks and how the Carla-ROS custom messages are used as a networking platform between Simulink and Carla.

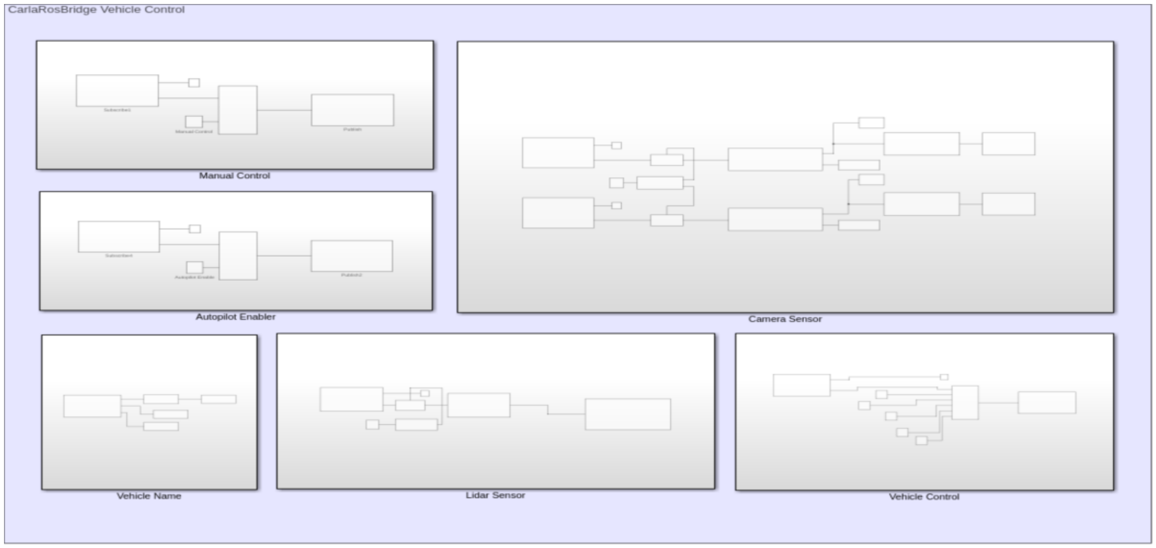


Figure 39: Vehicle Control Model

- The second model created in Simulink using the ROS blocks is capable of obtaining RGB images and Semantic Segmentation, fig (40), through obtaining a message from ROS network and forwarding it to the read image block, and also, the number of frames per second (FPS) can be displayed. Moreover, it is easy to choose between either displaying RGB images or Semantic Segmentation by just inputting a Boolean value, for example, 0 for case [0] to obtain RGB display and 1 for obtaining Semantic Segmentation

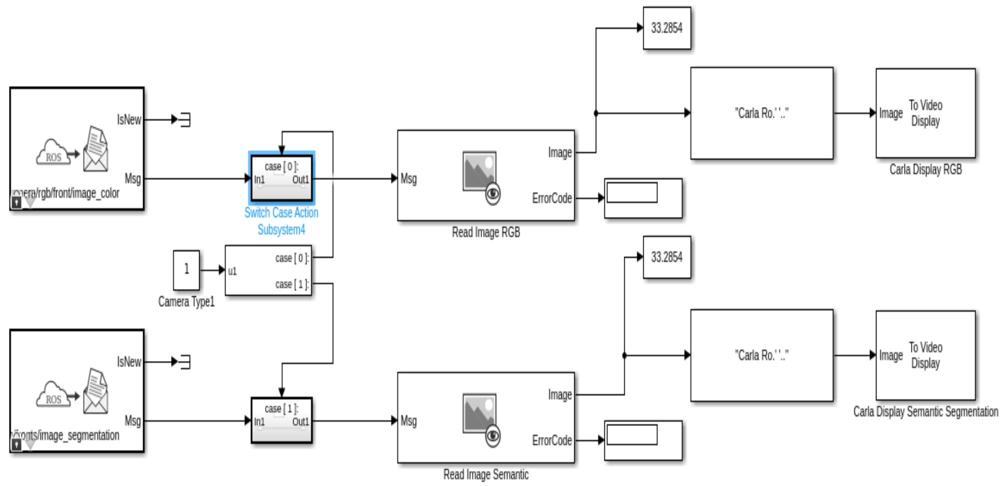


Figure 40: Camera Sensor Model

- Point Cloud Display model

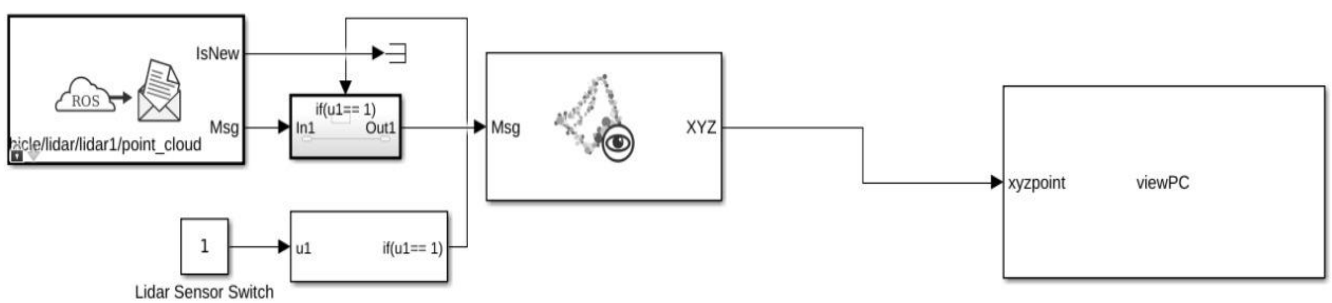


Figure 41: Point Cloud Display

- Enabling the visualizations sensors and shifting between Autopilot and manual control modes can be done using the Dashboard

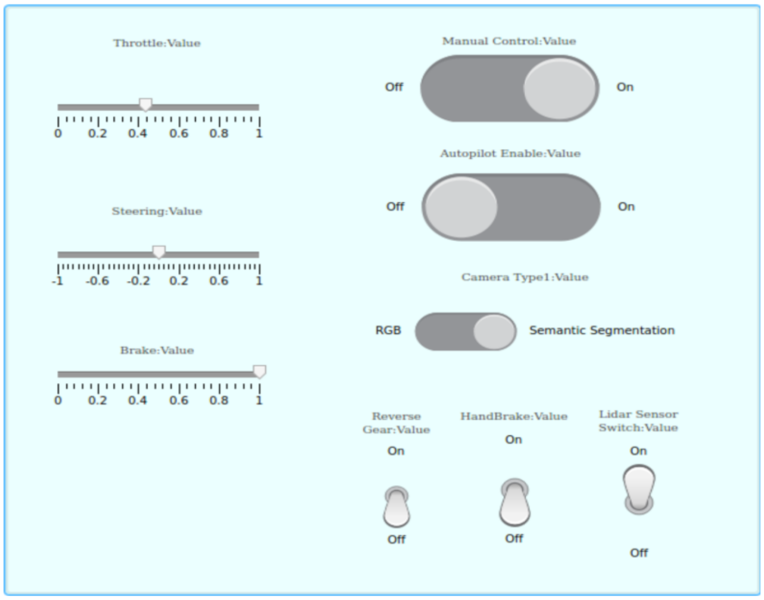


Figure 42: Dashboard

- The fourth model created was used in order to switch the vehicle from the autonomous mode to manual control inside Simulink through using the Publish block, which sends the enabling message to ROS network

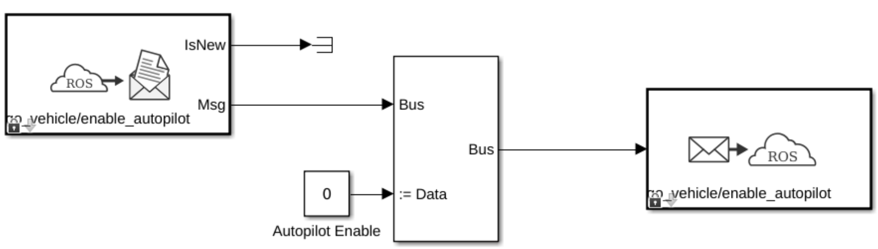


Figure 43: Disabling Autopilot

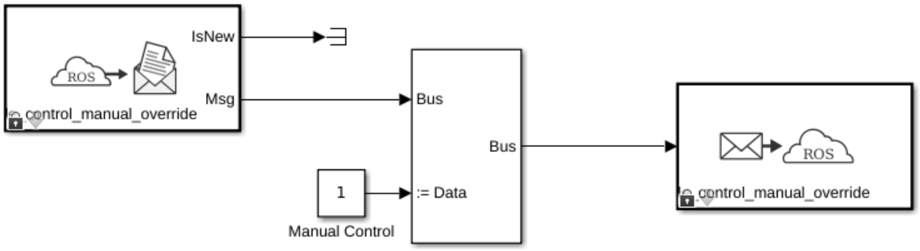


Figure 44: Manual Control Enabler

- If the manual control is enabled through choosing the value 1 inside the manual control block or by switching on the manual control switch located in the dashboard, the vehicle would be manually controlled through choosing certain values for throttling, steering and braking from the Dashboard

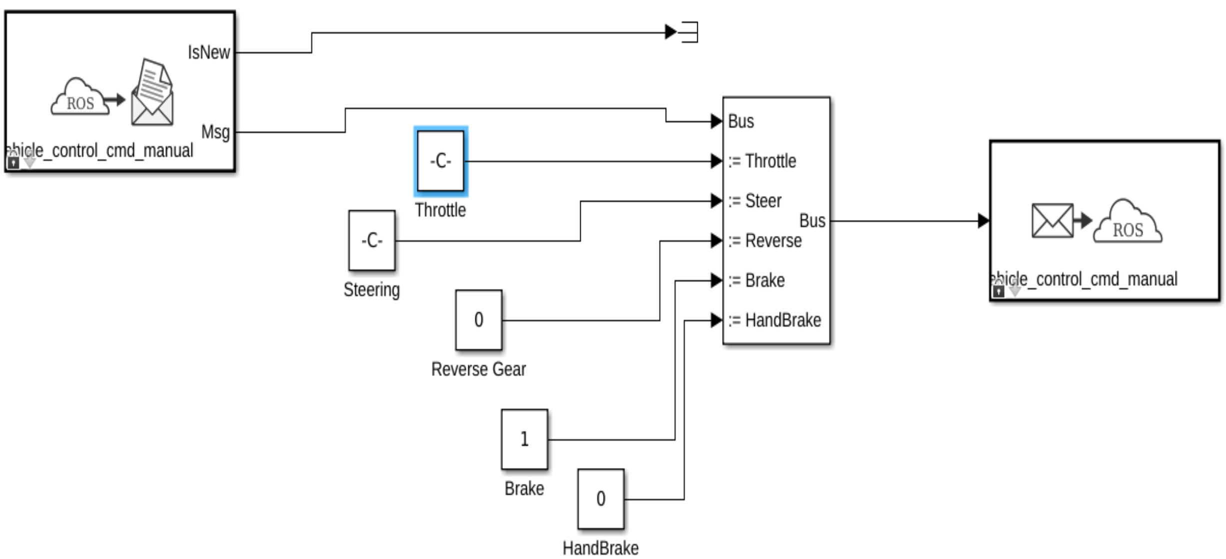


Figure 45: Vehicle Control

- Finally, after creating the models inside Simulink, for the co-simulation to take place, press on ***RUN*** button

Figure 46: Run the co-simulation

- These windows would be obtained in the end, Semantic Segmentation, RGB and Lidar

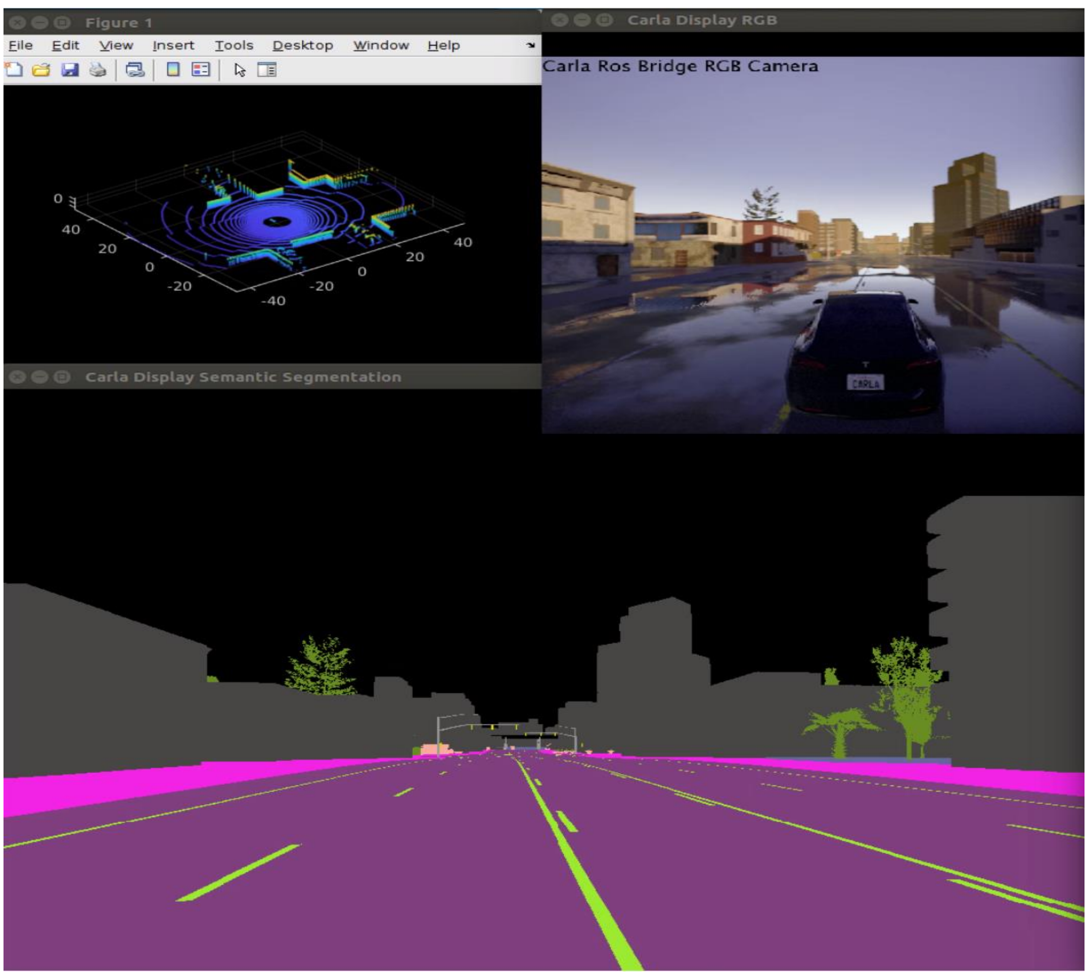


Figure 47: Final Display

Custom Examples

Lane Following

- After the Proof of concept, Lane Following case was created. The model shown in fig (48) uses Semantic Segmentation in order to detect lanes, also, as in the previous case of a Lidar, a new block named ***Lane Detection*** was created using ***Matlab.System*** class and a lane detection Algorithm was used. In the Algorithm, it was specified that whenever the vehicle detects a lane using semantic segmentation, a spline must be drawn on this lane

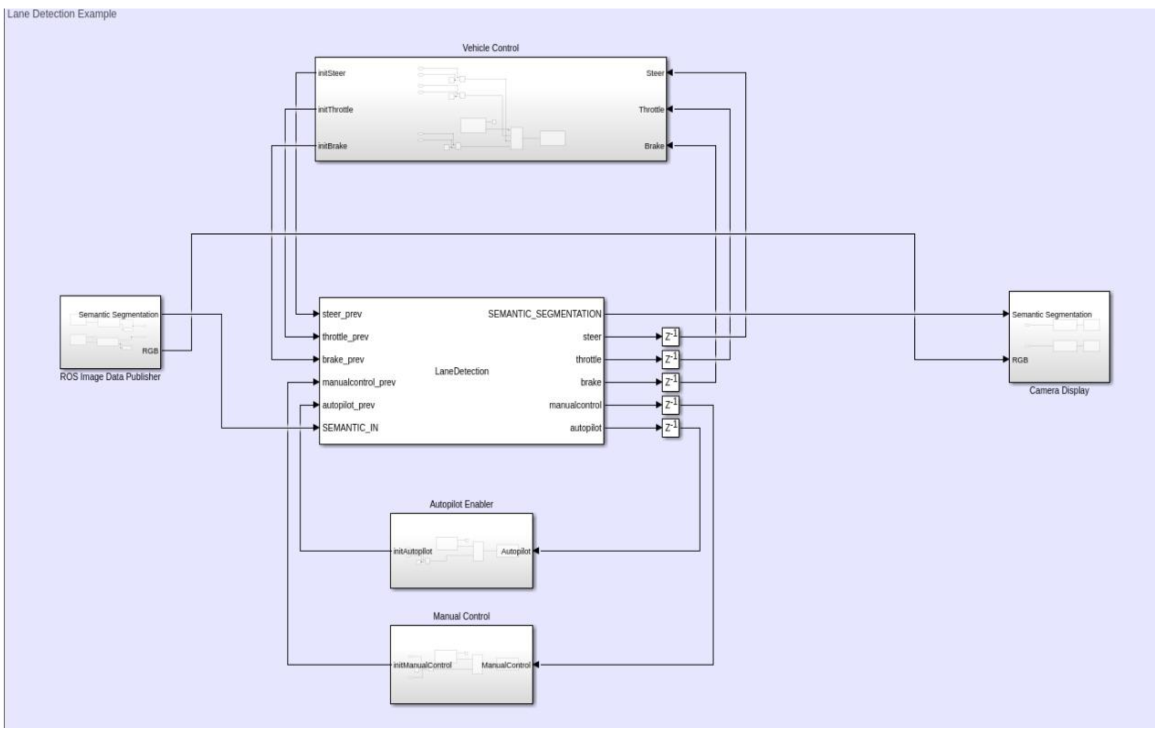


Figure 48: Lane Detection Model

```
% Update the spline history ~ every .2 seconds. A very high FPS
% would polute the spline fitting
if cputime - obj.push_back_time > .2

    %Delete the oldest element in the array
    obj.history_array(:,1) = [];

    %Most recent history weight
    weight = 1.25;

    %Average of the steer since the last update
    steer = (obj.steer_window_avg + steer)/(obj.frames_since_last_update + 1);

    %Add to the histroy
    obj.history_array = [obj.history_array, [cputime;steer]];

    %Pre-processing for the curve fitting
    [timeData, steerData] = prepareCurveData( obj.history_array(1,:),...
                                              obj.history_array(2,:));


    obj.history_array(end) = obj.history_array(end) * weight;

    %Fit a smoothing spline to the data
    obj.steer_spline = fit( timeData, steerData, 'smoothingspline', ...
                        'Normalize', 'on', 'SmoothingParam',0.95);

    obj.history_array(end) = obj.history_array(end) / weight;

    obj.steer_window_avg = 0;
    obj.frames_since_last_update = 0;

    obj.push_back_time = cputime;
```

Figure 49: Spline Addition

- The main focus of the lane detection model is to be able to control the vehicle and manuever it from inside Simulink, so the autopilot was initialized with zero, so after using the logic gate ***or*** to compare between the values, the autopilot would be disabled. On the contrary, the manual control would be initialized with a value of one, so the output of the logic gate would be one and the vehicle would be controlled manually

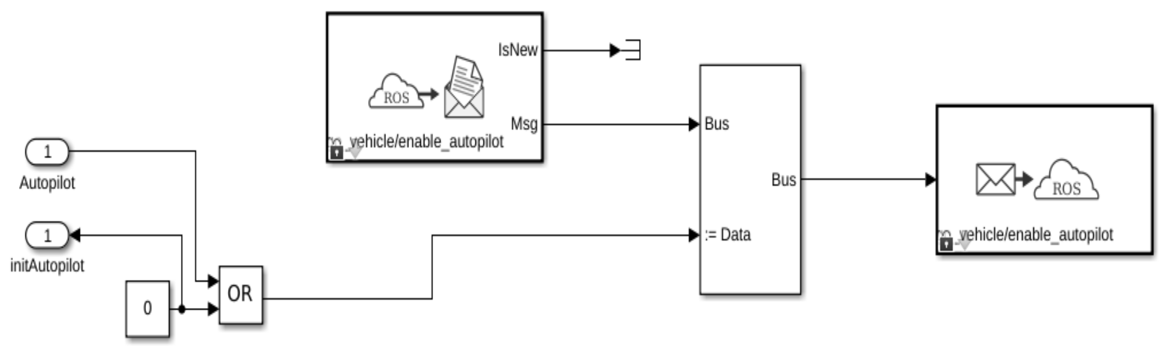


Figure 50: Autopilot Enabler

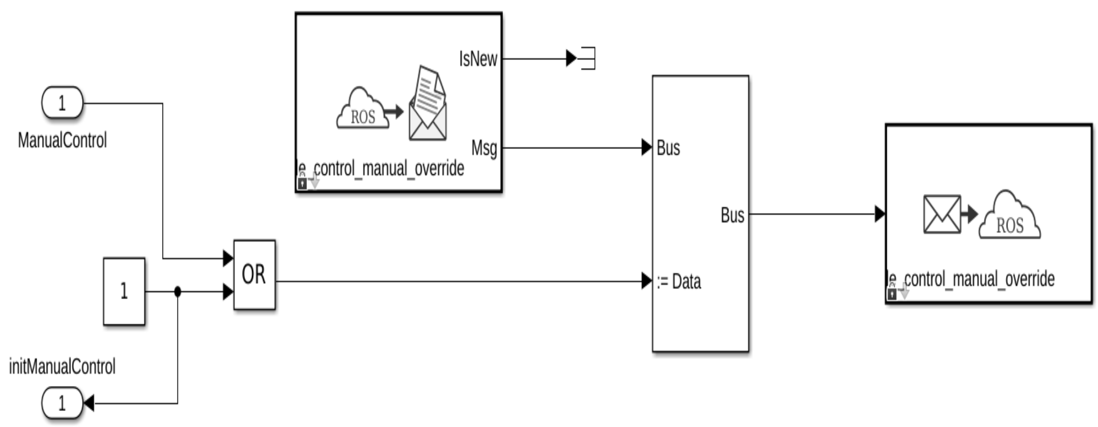


Figure 51: Manual Control Enabler

- While controlling the vehicle manually, according to its position with respect to the lanes, the steer, throttle and brake values would be updated by summing them up to their initial values and accordingly, the vehicle can change its position

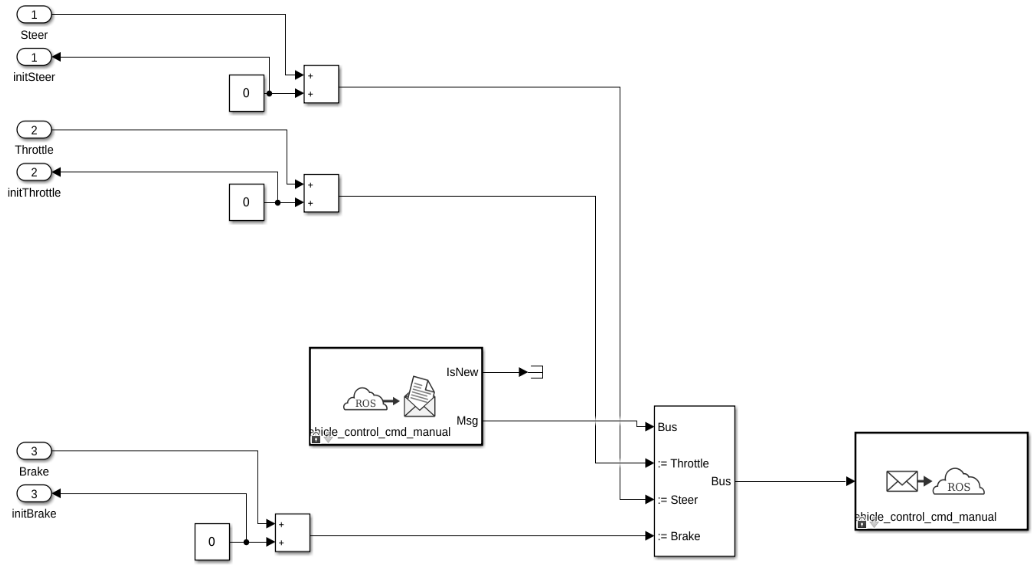


Figure 52: Vehicle Control

- After running this model on Simulink, these windows would be displayed, the first would be the RGB Camera Display, the second is Semantic Segmentation Display, and the last window is showing the update of the splines drawn in green over the white lanes after getting detected with Semantic Segmentation

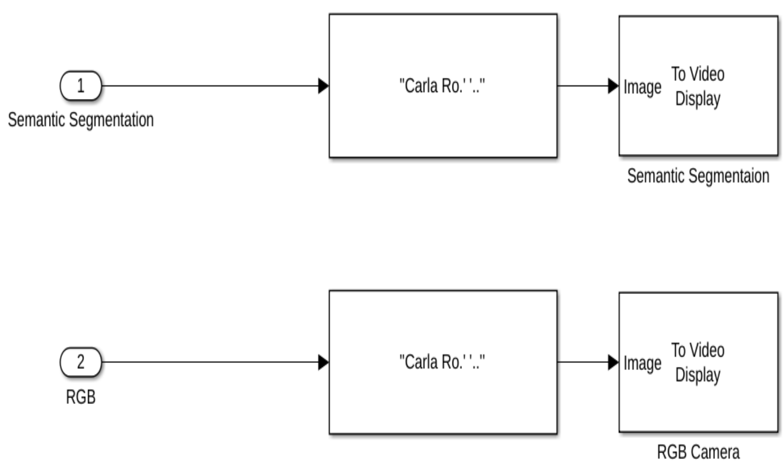


Figure 53: Sensor Display

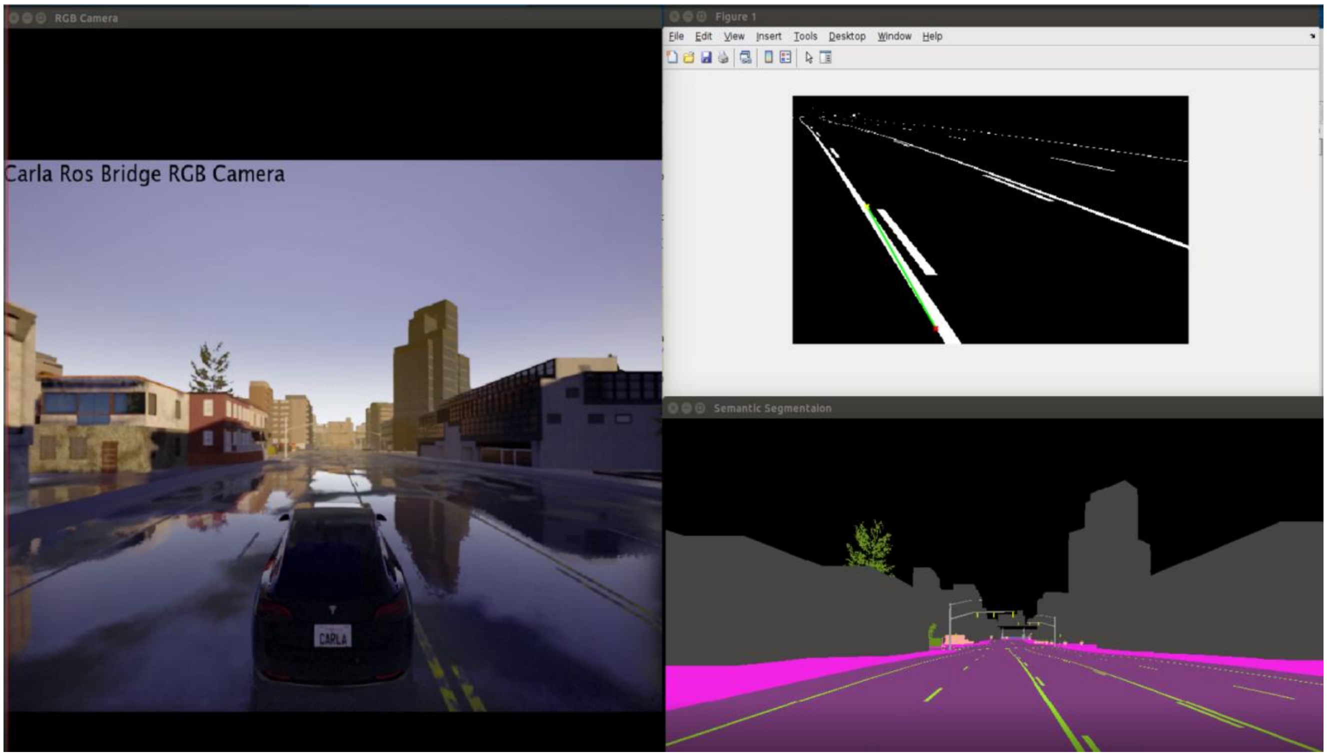


Figure 54: Output Window

Lidar Sensor

- The third model created was for Lidar sensor in order to obtain point cloud data and can be also enabled from the Dashboard that will be explained in the upcoming point. This model works as follows, the subscribe ROS block receives the message from the Carla environment then passed to the Read Point Cloud block to extract point cloud data, yet it was not possible to visualize these point cloud using the Display block in Simulink, so a new block- ***viewPC***- was created with ***Matlab.System class*** in order to write a MATLAB code to plot a 3-D point cloud data using the ***pcshow()*** built-in function

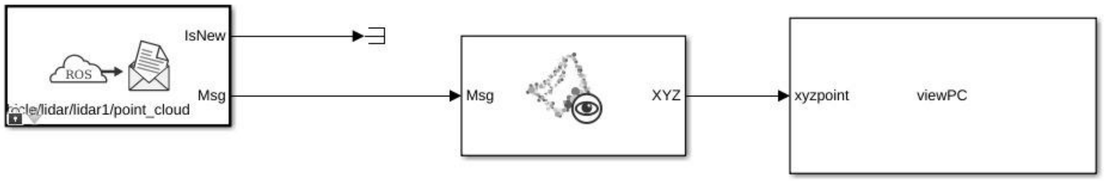


Figure 55: Lidar Sensor

Keyboard Control

- The last model created was a separate keyboard unit capable of controlling the vehicle with specific keys while enabling manual control mode

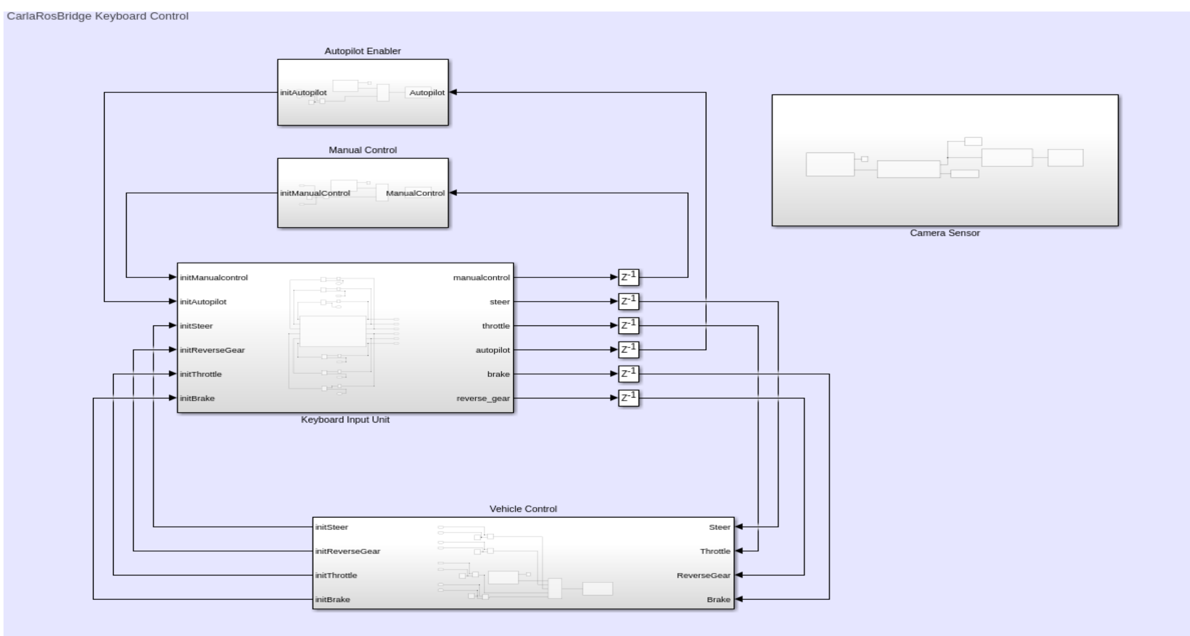


Figure 56: Keyboard Control Model

```
% Keys function
%
%    W          --->    Forward
%    S          --->    Brake
%    A          --->    Left
%    D          --->    Right
%    R          --->    Reverse gear
%    F          --->    Forward gear
%    E          --->    Enter auto control
%    Q          --->    Quit auto control
```

Figure 57: Input Keys

- Similar to the lane Detection model, so the autopilot was initialized with zero, so after using the logic gate ***or*** to compare between the values, the autopilot would be disabled. The Manual control was initialized with the value of 1 when ***Q*** is pressed to quit the auto control.

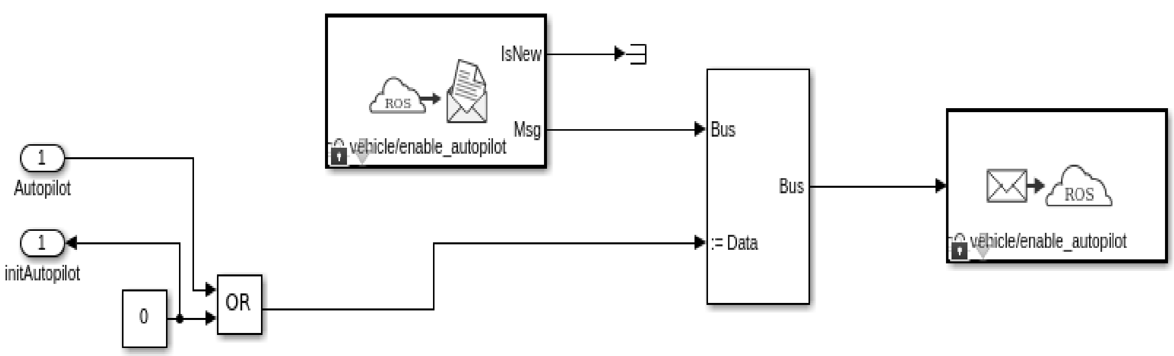


Figure 58: Autopilot Enabler

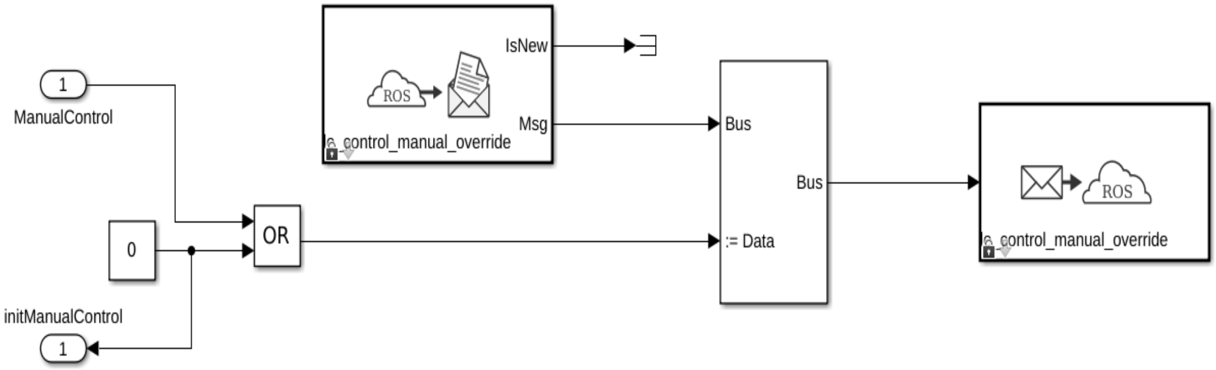


Figure 59: Manual Control Enabler

- The interaction between the vehicle control blocks and the keyboard input unit is a form of a loop updating in terms of updating the values for Throttling, steering, braking and for this keyboard model reverse gearing was included. For example, if the car is going to move forward from rest, the initial throttling value would be 0 and as long as the throttling key is pressed, ***W***, the throttling values would be updated in the vehicle control model and the new values would be taken from the keyboard input unit.

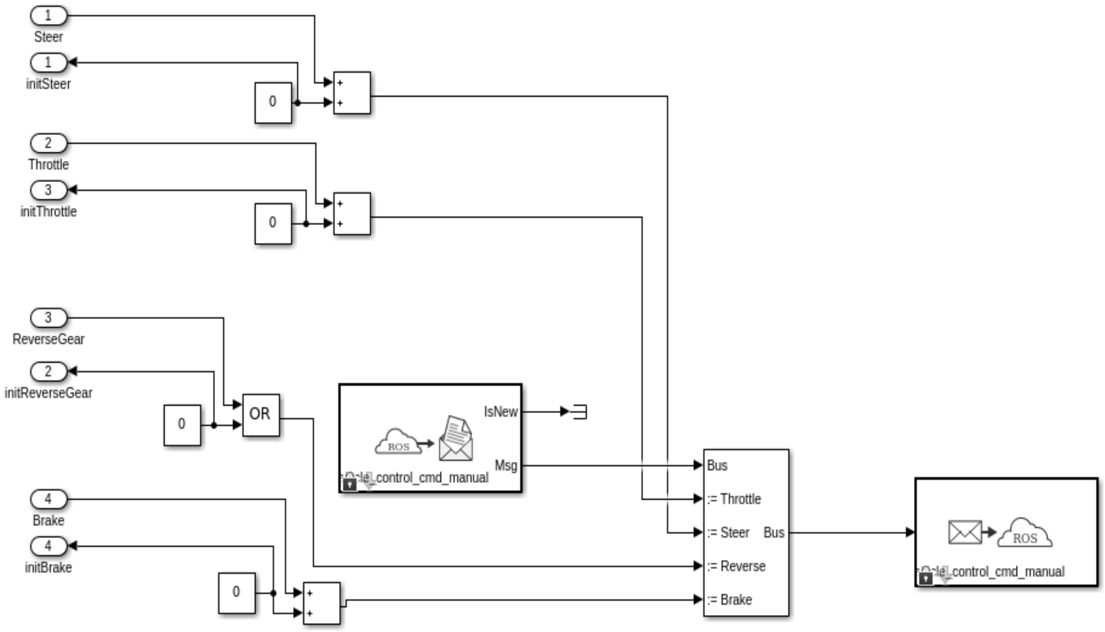


Figure 60: Vehicle Control

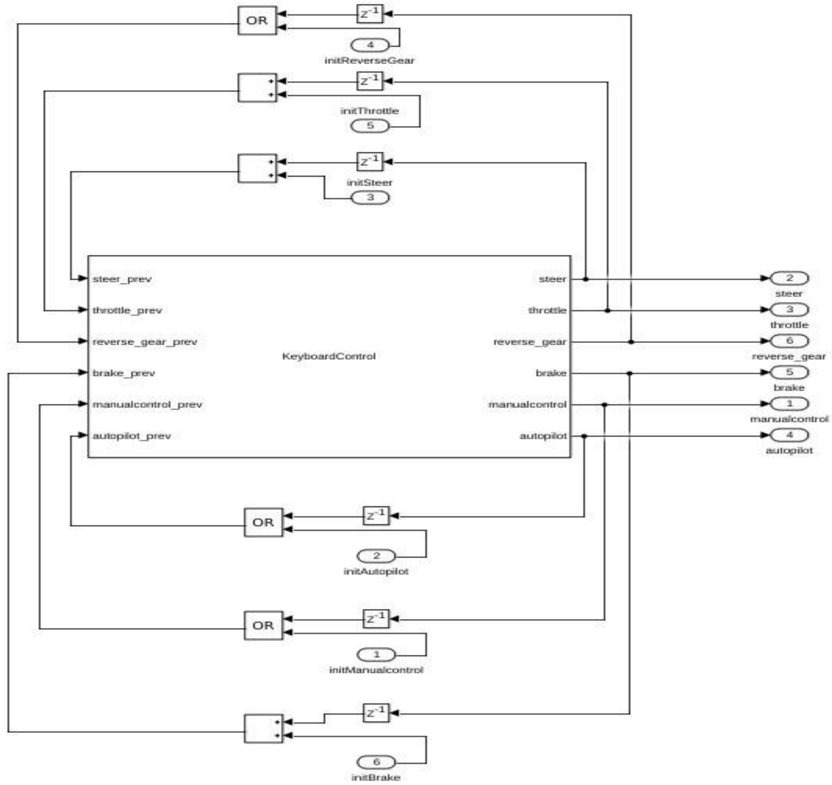


Figure 61: Keyboard Input Unit

- Finally, after running this model, these windows would be the output and in order to switch to the manual mode and run the Carla vehicle from inside Simulink, press the ***Q*** key

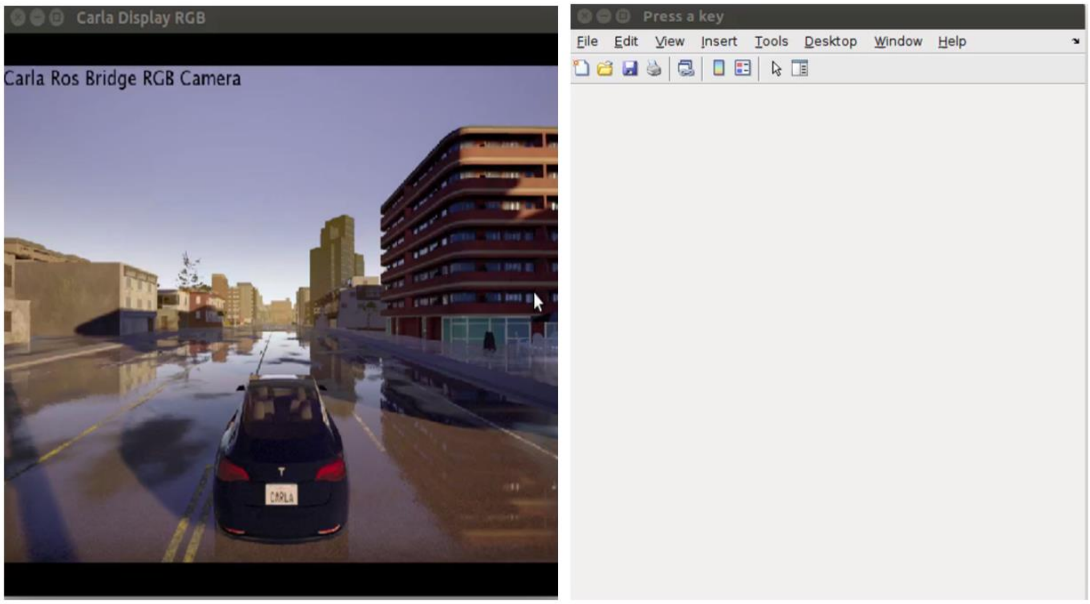


Figure 62: Keyboard Model output

APPENDIX

Testing Installation Process on a New Hardware

Specifications

Operating system

- Linux-Ubuntu 16.04 LTS (Xenial Xerus)

Hardware Specifications

- 64-bit PC
- Intel Core i7 (9th Generation Processor)
- Nvidia GTX-1650
- 8 GB RAM

- The installation process was as similar as the process of the old Hardware, but the first problem appeared while doing the initialization of Carla UE4 Editor, as the compatible Vulkan Driver was missing, so the old technique was tried in order to change to a Nvidia Proprietary, for GTX 1650, the 430 Driver package was only available to install using the following commands

(1)
```
sudo add-apt-repository ppa:graphics-
```

(2)
```
sudo apt update
```

(3)
```
sudo apt install nvidia-driver-430 nvidia-settings
```

Yet, for the last command, it was necessary to boot the whole system in insecure mode and after doing so, the driver was installed not as a proprietary, but as an open source as shown in fig (63)

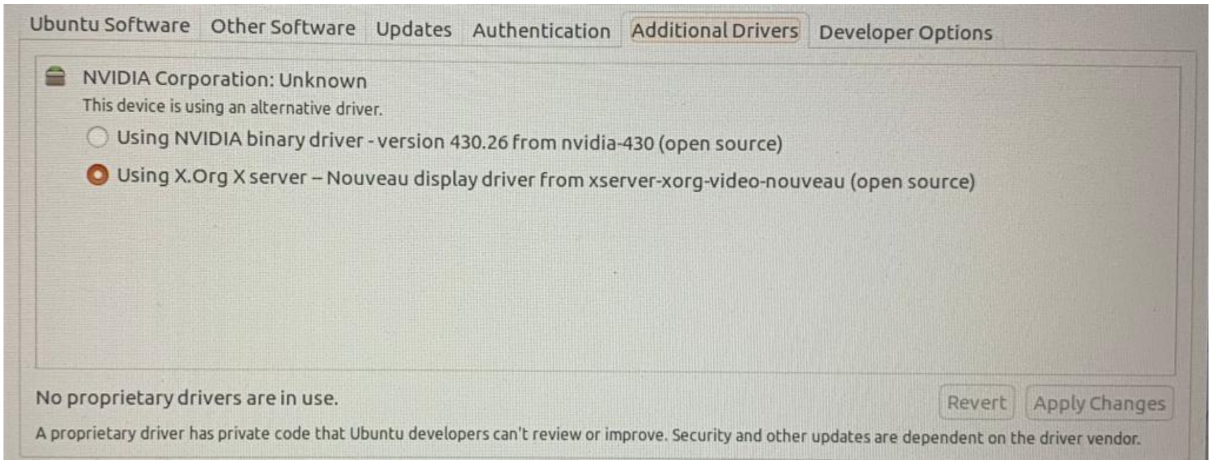


Figure 63: Additional Drivers in the new Hardware

- Therefore, installing a Vulkan-driver was a must using the following commands

(1) `sudo add-apt-repository ppa:oibaf/graphics-drivers`

(2) `sudo apt update`

(3) `sudo apt upgrade`

(4) `apt install libvulkan1 mesa-vulkan-drivers vulkan-utils`

After installing the Vulkan-Driver, the initialization process was completed. The main problem showed up while running the Python API manual script on the Pygame Window, as the Server's FPS were barely reaching 4, which is really low value to have a proper simulation, fig (64)

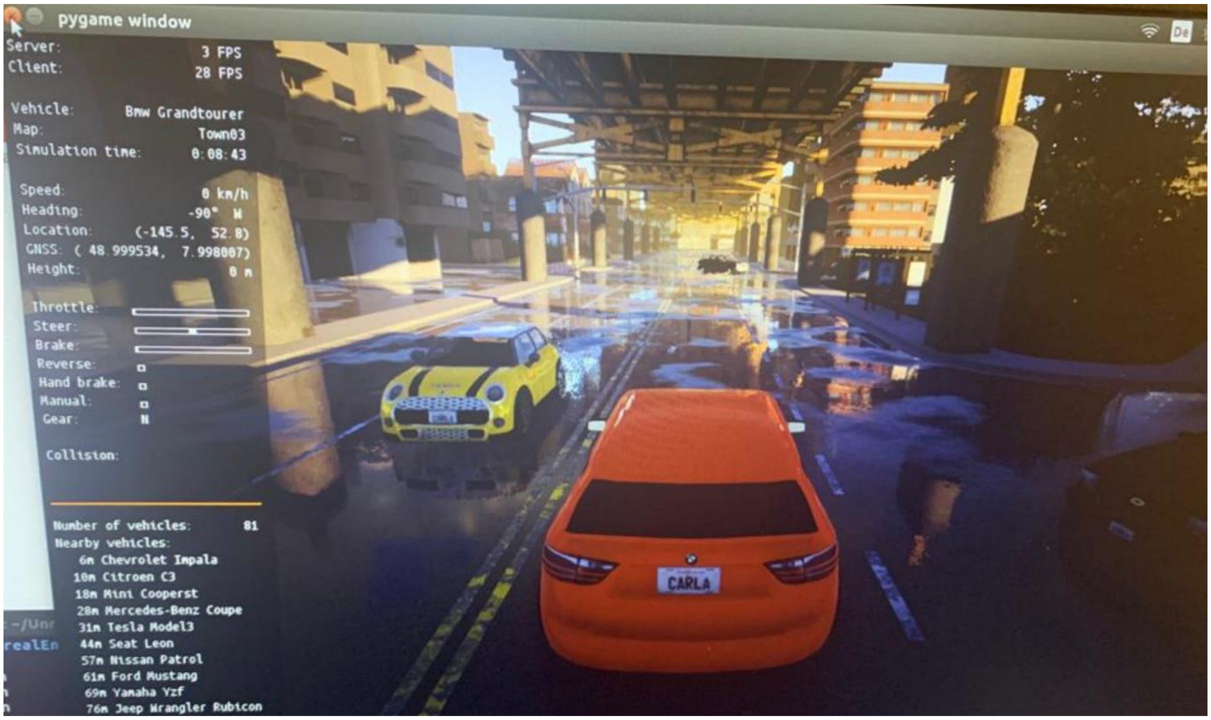


Figure 64: Pygame Window with the new Hardware

- In an attempt to solve this problem, we tried using the Nvidia Binary driver; however, the 430 driver was not compatible with Ubuntu 16.04 LTS, as after restarting the system, this message showed up, fig (65), and restoring the old graphics was really difficult, so there was no resort but deleting the operating system and starting everything from scratch

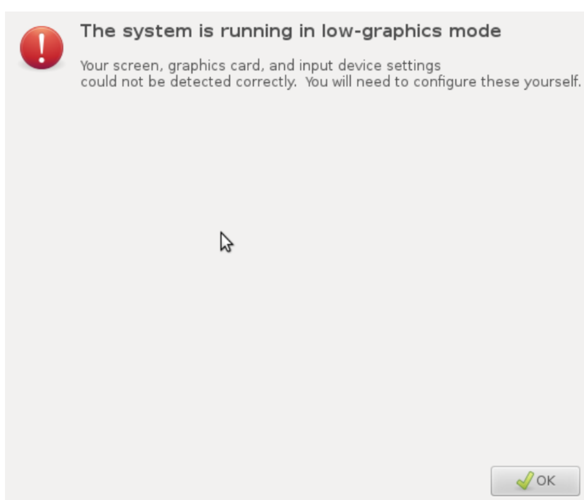


Figure 65: Low Graphics Mode Message

Testing Installation Process on a New Hardware and Operating System

- Trying a higher version of the operating system to test whether the 430 driver would be compatible with it or not was a choice, so the installation process was done again, yet on Ubuntu 18.04 LTS this time.
- This time the main problem was downloading ROS-Kinetic because it only operates on Linux 16.04, so downloading ROS-Melodic was a must, but due to lack of documentation, we could not proceed any further to check whether Carla and Simulink could be interfaced using the ROS bridge while running this time using ROS-Melodic instead of ROS-Kinetic, so our decision is to do more research later on this area, but if things worked-out in a fine way and the 430 driver was compatible, this would give the simulation a huge boost in terms of the number of FPS