Documentation of Carla-Ros Interface

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)

CARLA
ROS
MATLAB & SIMULINK

Figure 1: Connecting Carla to MATLAB & Simulink through ROS

The operating system for this process is Linux:

- Ubuntu 16.04 LTS (Xenial Xerus)

The Hardware Specifications:

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

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

- Install the build tools and dependencies

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

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

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

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

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

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

```
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
```
(7)

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

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

- Unreal Engine must be installed at first, 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

```
sudo update-alternatives --install /usr/bin/clang++ clang++ /usr/lib/llvm-7/bin/clang++ 170
```
(1)

```
sudo update-alternatives --install /usr/bin/clang clang /usr/lib/llvm-7/bin/clang 170
```
(2)

- 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

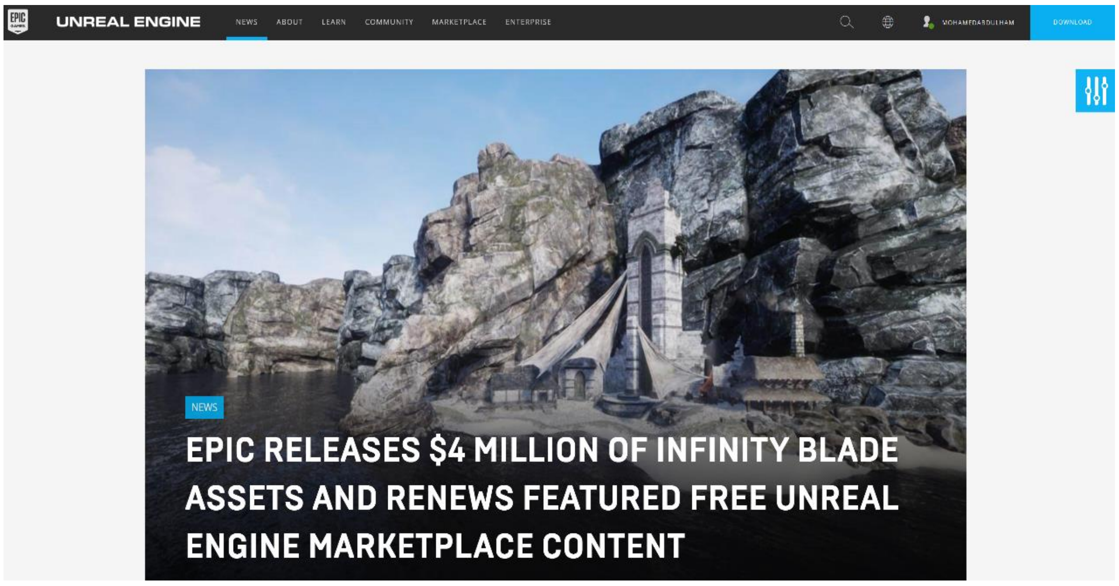

UNREAL ENGINE
NEWS
ABOUT
LEARN
COMMUNITY
MARKETPLACE
ENTERPRISE
DOWNLOAD
NEWS
EPIC RELEASES $4 MILLION OF INFINITY BLADE ASSETS AND RENEWS FEATURED FREE UNREAL ENGINE MARKETPLACE CONTENT

Figure 2: Unreal Engine Log-in Page

- Go to Personal
- Choose connected Accounts

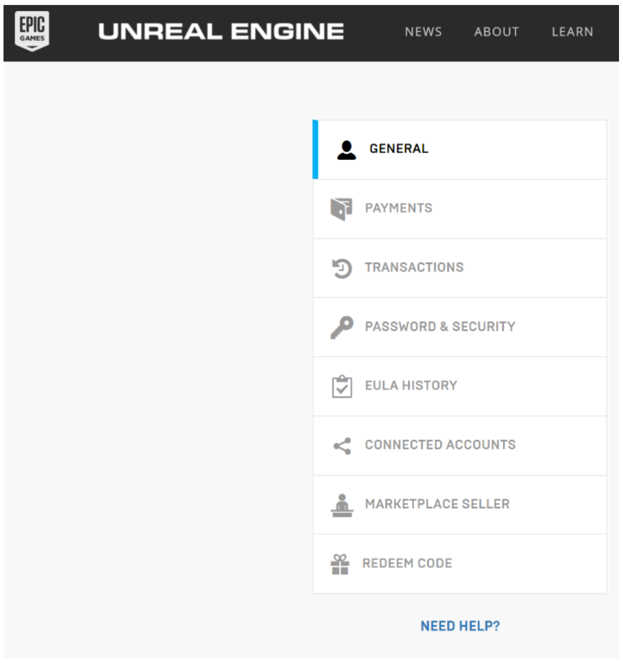
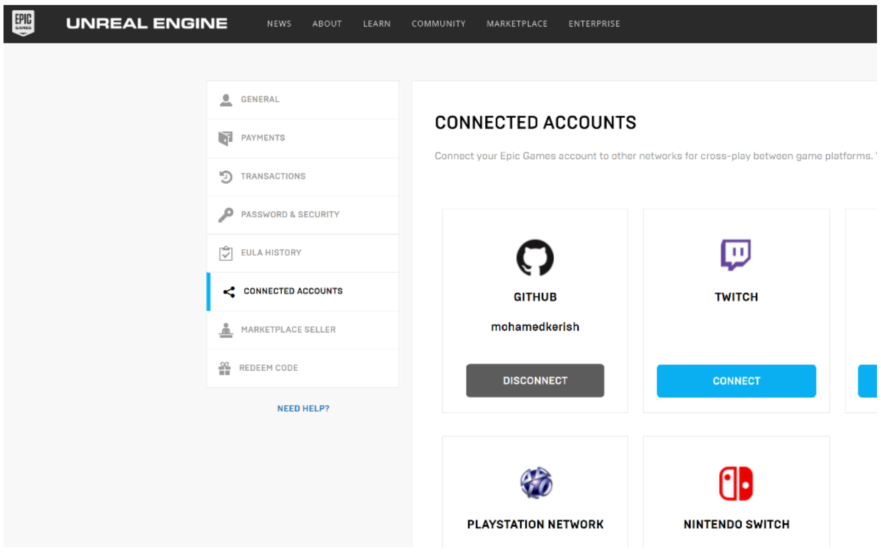


Figure 3: Connected Accounts

- Finally, connect to GitHub Account



CONNECTED ACCOUNTS

Connect your Epic Games account to other networks for cross-play between game platforms.

Figure 4: Connect to GitHub Account

- Download and Compile Unreal Engine version 4.22 (requires minimum of 8 GB of RAM)

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

```
cd ~/UnrealEngine_4.22
```
(2)

```
./Setup.sh && ./GenerateProjectFiles.sh && make
```
(3)

- The next step is to install Ros and the version that will be used is ROS-Kinetic, which only supports Ubuntu 16.04 Xenial. The installation and compilation steps are as follows:

    - Setup your computer to accept software from packages.ros.org.

```
sudo sh -c 'echo "deb http://packages.ros.org/ros/ubuntu $(lsb_release -sc) main" > /etc/apt/sources.list.d/ros-latest.list'
```
(1)

    - Set up your keys

```
sudo apt-key adv --keyserver 'hkp://keyserver.ubuntu.com:80' --recv-key C1CF6E31E6BADE8868B172B4F42ED6FBAB17C654
```
(2)

    - Installation

```
Sudo apt-get update
```
(3)

    - **Desktop-Full Install: (Recommended)** : ROS, rqt, rviz, robot-generic libraries, 2D/3D simulators, navigation and 2D/3D perception

```
sudo apt-get install ros-kinetic-desktop-full
```
(4)

    - Initialize ROSdep, which enables easy installation of system dependencies and to run some core components in ROS

```
sudo rosdep init
```
(5)

```
rosdep update
```
(6)

- Environment Setup, to automatically add the ROS environment variables to the bash session every time a new shell is launched

```
echo "source /opt/ros/kinetic/setup.bash" >> ~/.bashrc
```
(7)

```
source ~/.bashrc
```
(8)

- 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

```
sudo apt install python-rosinstall python-rosinstall-generator python-wstool build-essential
```
(9)

- 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 (do the whole compilation inside the Unreal Engine_4.22 file)

    - Setup folder structure

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

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

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

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

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

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

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

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

```
rosdep update
```
(8)

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

- The final step is to build

```
catkin_make
```
(10)

- The final procedure is to clone Carla from GitHub and install it (do the whole compilation inside the Unreal Engine_4.22 file)

  - Clone from Repository

```
git clone https://github.com/carla-simulator/carla
```
(1)

  - The following command will take so long to run; therefore, it is recommended to run this command at first

```
sudo apt-get install aria2
```
(2)

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

```
./Update.sh
```
(3)

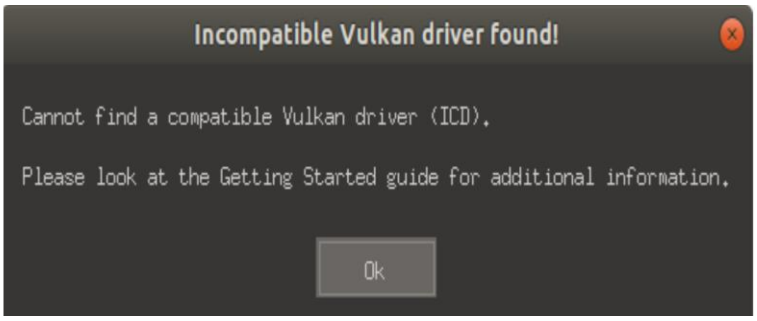
  - For Carla to find Unreal Engine's installation folder, the following environment variable must be set

```
export UE4_ROOT=~/UnrealEngine_4.22
```
(4)

  - Run the following command to compile the simulator and launch Unreal Engine's Editor

```
make launch
```
(5)

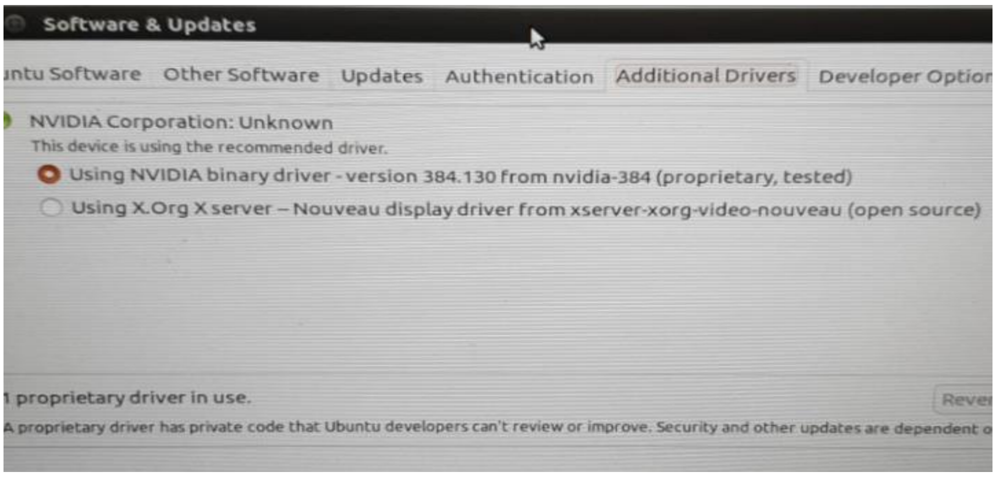
If the Carla-Unreal Editor starts to initialize and suddenly, this error shows up


Incompatible Vulkan driver found!

```
Cannot find a compatible Vulkan driver (ICD).

Please look at the Getting Started guide for additional information.
```

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)



NVIDIA Corporation: Unknown
This device is using the recommended driver.
[✓ Using NVIDIA binary driver - version 384.130 from nvidia-384 (proprietary, tested)]
Using X.Org X server – Nouveau display driver from xserver-xorg-video-nouveau (open source)

1 proprietary driver in use.
A proprietary driver has private code that Ubuntu developers can't review or improve. Security and other updates are dependent o

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)

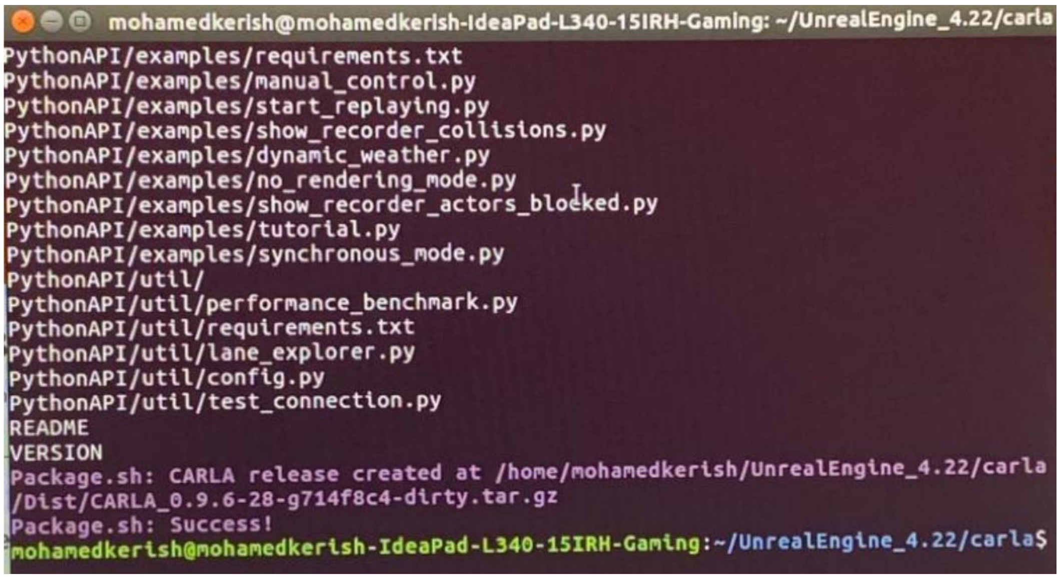
- Run the following command to compile the Python API module responsible for running the Python examples

make PythonAPI (6)

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

make package (7)

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)



```
mohamedkerish@mohamedkerish-IdeaPad-L340-15IRH-Gaming: ~/UnrealEngine_4.22/carla
PythonAPI/examples/requirements.txt
PythonAPI/examples/manual_control.py
PythonAPI/examples/start_replaying.py
PythonAPI/examples/show_recorder_collisions.py
PythonAPI/examples/dynamic_weather.py
PythonAPI/examples/no_rendering_mode.py
PythonAPI/examples/show_recorder_actors_blocked.py
PythonAPI/examples/tutorial.py
PythonAPI/examples/synchronous_mode.py
PythonAPI/util/
PythonAPI/util/performance_benchmark.py
PythonAPI/util/requirements.txt
PythonAPI/util/lane_explorer.py
PythonAPI/util/config.py
PythonAPI/util/test_connection.py
README
VERSION
Package.sh: CARLA release created at /home/mohamedkerish/UnrealEngine_4.22/carla
/Dist/CARLA_0.9.6-28-g714f8c4-dirty.tar.gz
Package.sh: Success!
mohamedkerish@mohamedkerish-IdeaPad-L340-15IRH-Gaming:~/UnrealEngine_4.22/carla$
```

Figure 7: Package. sh: Success

- After doing all the previous steps, start the ROS-bridge
  - 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

```
./CarlaUE4.sh -windowed -ResX=320 -ResY=240
```
(1)

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

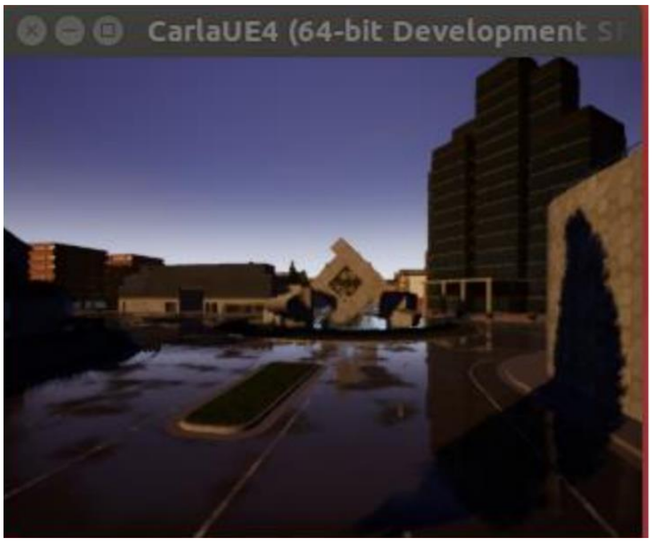


Figure 8: Carla UE4 Window

- Open the terminal and run the following command

```
export PYTHONPATH=$PYTHONPATH: <path/to/carla/>/PythonAPI/<your_egg_file>
```
(2)

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**

- Open new terminal and run the following command

```
source ~/carla-ros-bridge/catkin_ws/devel/setup.bash
```
(3)

There are three options to choose from:

- Start the ROS-bridge

```
roslaunch carla_ros_bridge carla_ros_bridge.launch
```

- Start the ROS-bridge together with RVIZ

```
roslaunch carla_ros_bridge carla_ros_bridge_with_rviz.launch
```

- Start the ROS-bridge together with an example ego vehicle

```
roslaunch carla_ros_bridge carla_ros_bridge_with_example_ego_vehicle.launch
```

****

For the purpose of verification of installation, the second option is chosen and 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.

- 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**

```
./spawn_npc.py -n (number of spawned characters)
```
(5)

n 80 was used in this example.
if the pygame dependency did not work out, run the following command

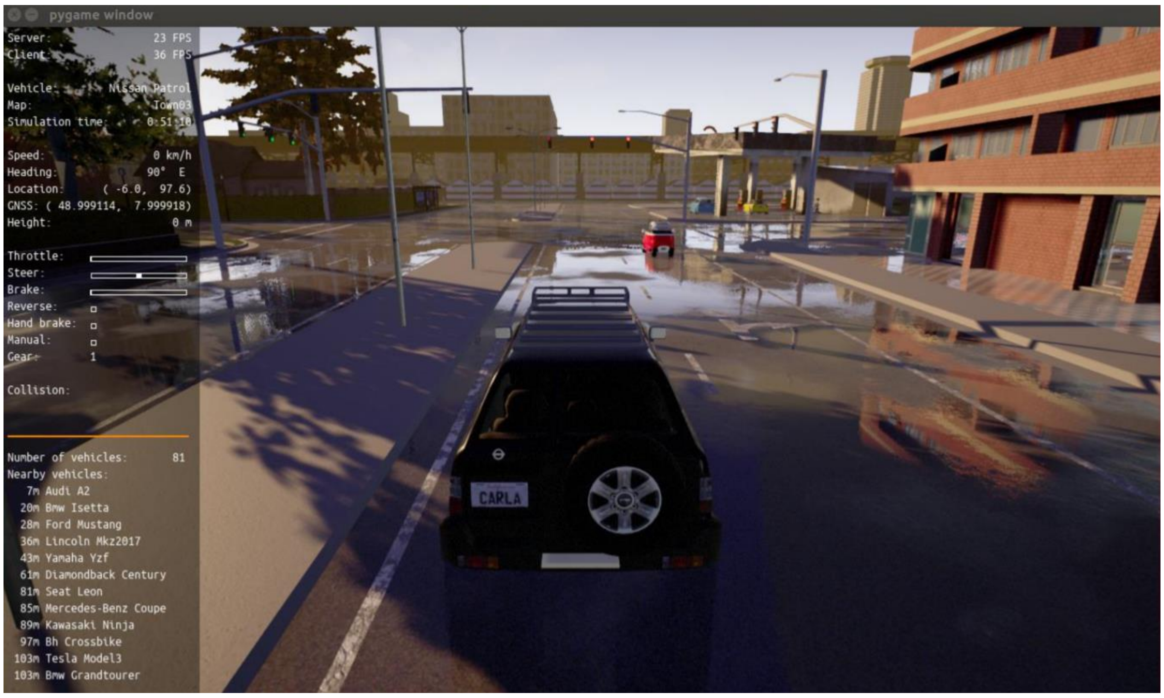
```
sudo apt-get install python-pygame
```
(6)

Run command (5) again

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

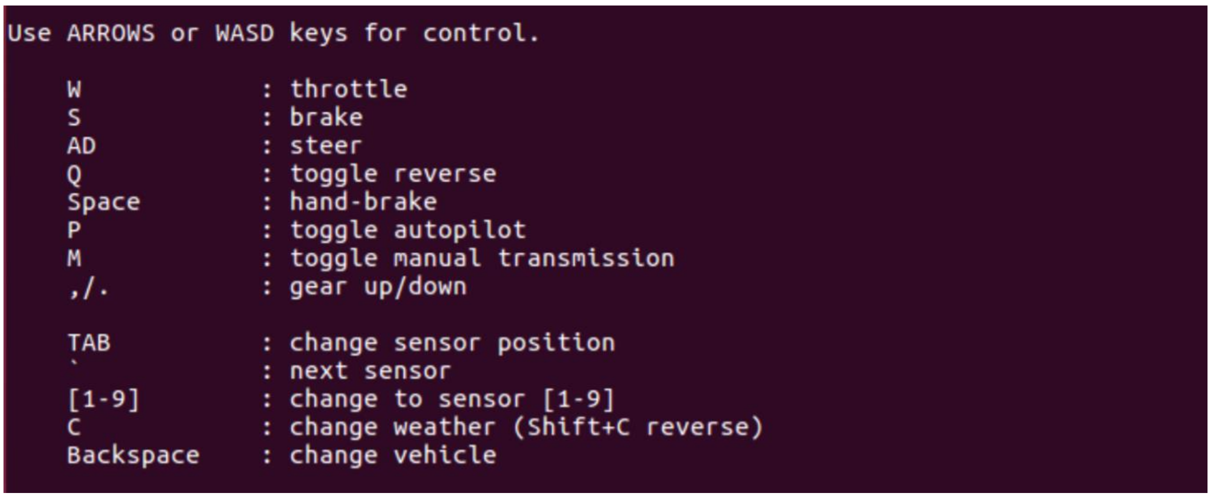
```
./manual_control.py
```

And 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)



```
Server:                23 FPS
Client:                36 FPS

Vehicle:         Nissan Patrol
Map:                    Town03
Simulation time:       0:51:10

Speed:                  0 km/h
Heading:                90°  E
Location:        ( -6.0, 97.6)
GNSS: ( 48.999114, 7.999918)
Height:                    0 m

Throttle:
Steer:
Brake:
Reverse:
Hand brake:
Manual:
Gear:          1

Collision:

Number of vehicles:        81
Nearby vehicles:
   7m Audi A2
  20m Bmw Isetta
  28m Ford Mustang
  36m Lincoln Mkz2017
  43m Yamaha Yzf
  61m Diamondback Century
  81m Seat Leon
  85m Mercedes-Benz Coupe
  89m Kawasaki Ninja
  97m Bh Crossbike
 103m Tesla Model3
 103m Bmw Grandtourer
```

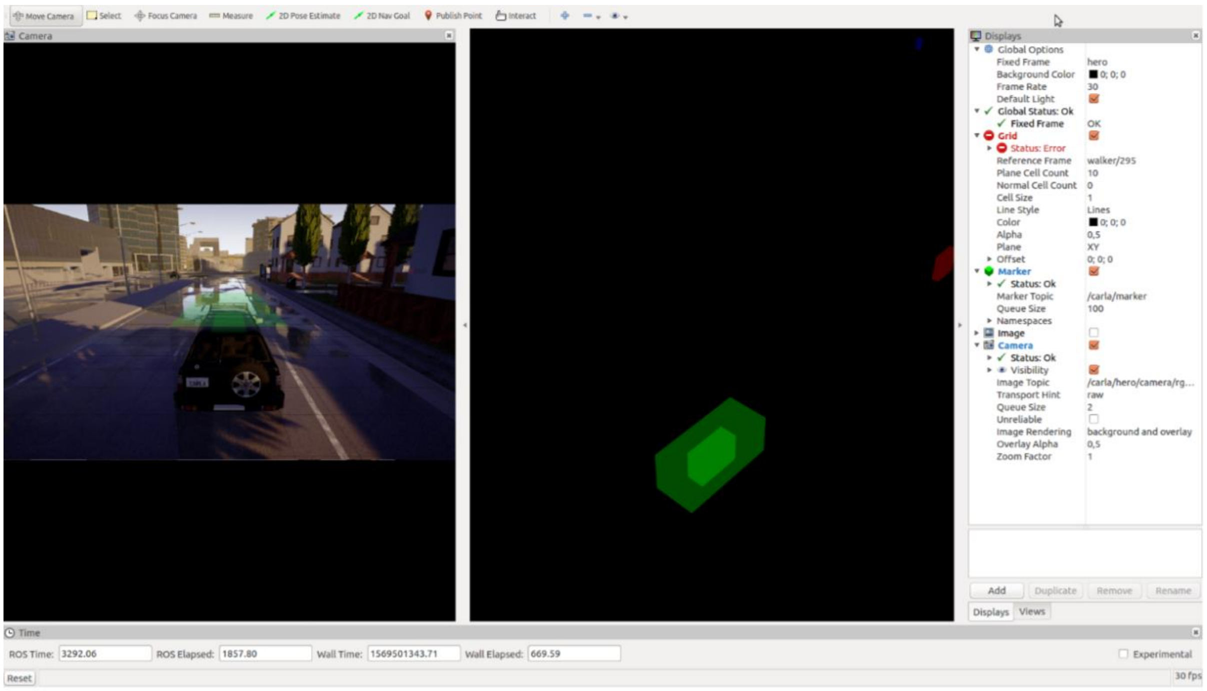
Figure 9: Pygame Window



```
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)



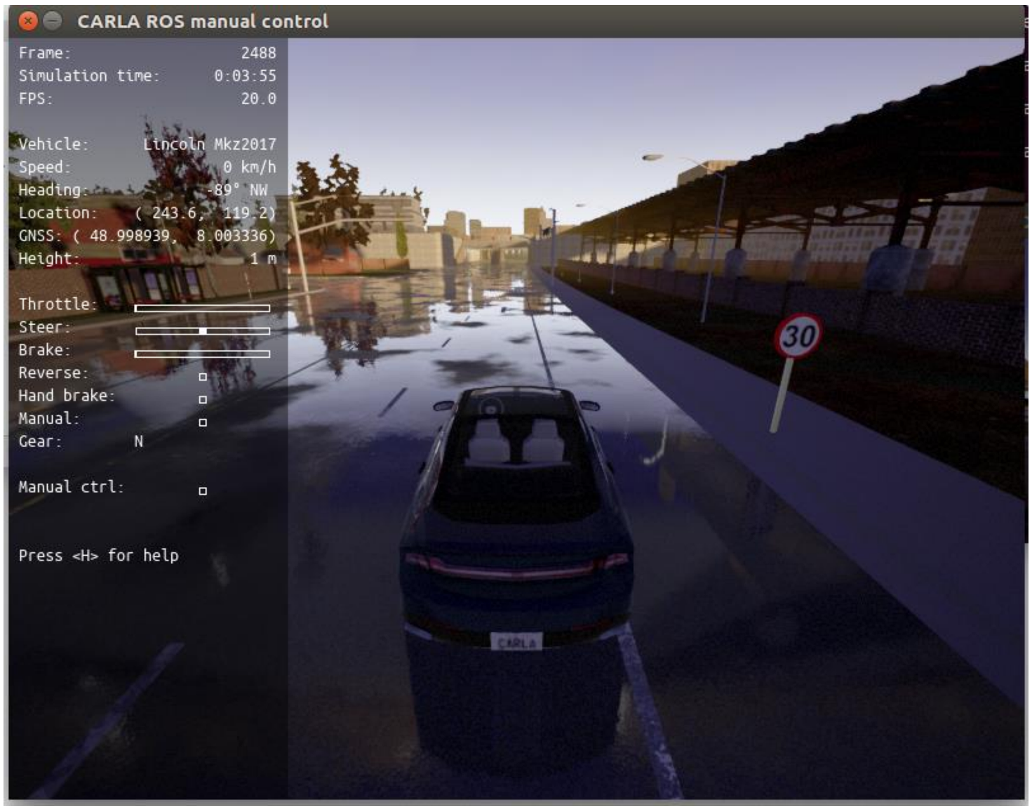
```
Global Options
  Fixed Frame            hero
  Background Color       ■ 0; 0; 0
  Frame Rate             30
  Default Light          ✓
Global Status: Ok
  ✓ Fixed Frame          OK
Grid                     ✓
  Status: Error
  Reference Frame        walker/295
  Plane Cell Count       10
  Normal Cell Count      0
  Cell Size              1
  Line Style             Lines
  Color                  ■ 0; 0; 0
  Alpha                  0,5
  Plane                  XY
  Offset                 0; 0; 0
Marker                   ✓
  ✓ Status: Ok
  Marker Topic           /carla/marker
  Queue Size             100
  Namespaces
Image                    ☐
Camera                   ✓
  ✓ Status: Ok
  Visibility             ✓
  Image Topic            /carla/hero/camera/rg...
  Transport Hint         raw
  Queue Size             2
  Unreliable             ☐
  Image Rendering        background and overlay
  Overlay Alpha          0,5
  Zoom Factor            1

ROS Time: 3292.06   ROS Elapsed: 1857.80   Wall Time: 1569501343.71   Wall Elapsed: 669.59   ☐ Experimental
30 fps
```

Figure 11: RVIZ Window with Camera Images from Carla

*********************************End of Verification**********************************

After the verification of installation, the third option will be selected. By using this option, an example vehicle is created with a command of Carla-Ros Bridge.

```
roslaunch carla_ros_bridge carla_ros_bridge_with_example_ego_vehicle.launch
```



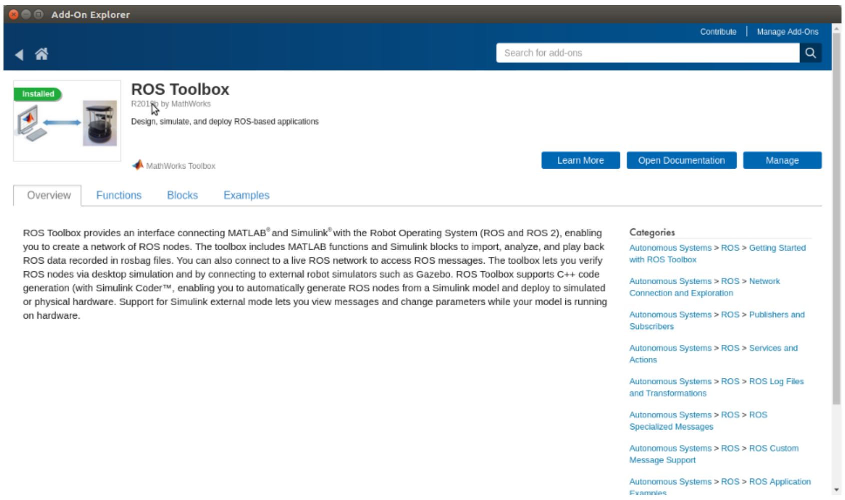
CARLA ROS manual control

```
Frame:                   2488
Simulation time:      0:03:55
FPS:                     20.0

Vehicle:       Lincoln Mkz2017
Speed:                 0 km/h
Heading:              -89° NW
Location:      ( 243.6,  119.2)
GNSS: ( 48.998939,  8.003336)
Height:                   1 m

Throttle:
Steer:
Brake:
Reverse:
Hand brake:
Manual:
Gear:          N

Manual ctrl:

Press <H> for help
```

Carla Ros manual control screen is opened after launching ROS. In this window, car can be controlled by using keyboard.

The bridge connects the Carla world and Ros. For the next step, MATLAB and Ros will be connected to send data to Simulink. There are some required add-ons will be installed for that purpose.

1)ROS Toolbox



ROS Toolbox

R2019b by MathWorks

Design, simulate, and deploy ROS-based applications

MathWorks Toolbox

ROS Toolbox provides an interface connecting MATLAB® and Simulink® with the Robot Operating System (ROS and ROS 2), enabling you to create a network of ROS nodes. The toolbox includes MATLAB functions and Simulink blocks to import, analyze, and play back ROS data recorded in rosbag files. You can also connect to a live ROS network to access ROS messages. The toolbox lets you verify ROS nodes via desktop simulation and by connecting to external robot simulators such as Gazebo. ROS Toolbox supports C++ code generation (with Simulink Coder™, enabling you to automatically generate ROS nodes from a Simulink model and deploy to simulated or physical hardware. Support for Simulink external mode lets you view messages and change parameters while your model is running on hardware.

Categories

Autonomous Systems > ROS > Getting Started with ROS Toolbox

Autonomous Systems > ROS > Network Connection and Exploration

Autonomous Systems > ROS > Publishers and Subscribers

Autonomous Systems > ROS > Services and Actions

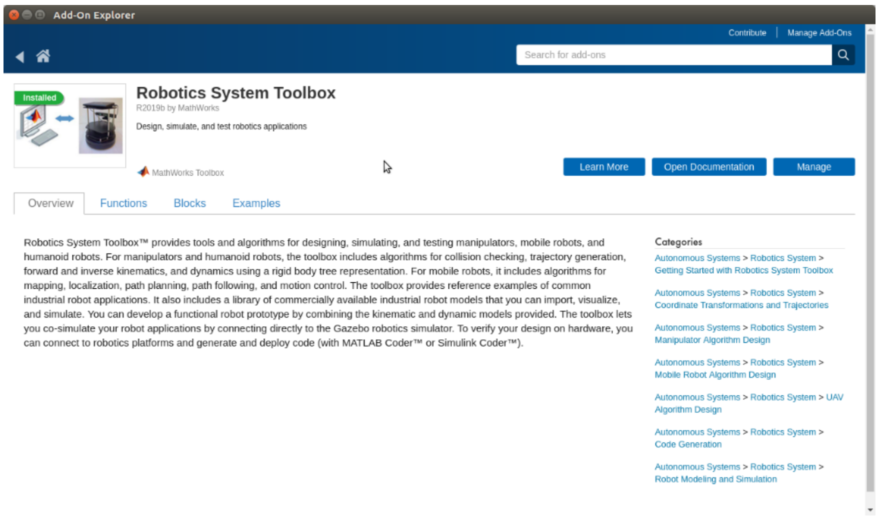
Autonomous Systems > ROS > ROS Log Files and Transformations

Autonomous Systems > ROS > ROS Specialized Messages

Autonomous Systems > ROS > ROS Custom Message Support

Autonomous Systems > ROS > ROS Application Examples

2)Robotics System Toolbox

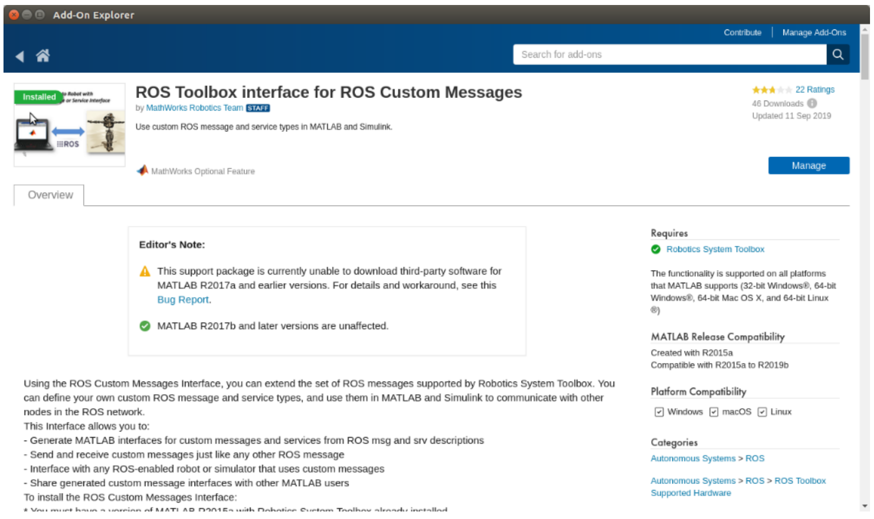


Robotics System Toolbox
R2019b by MathWorks
Design, simulate, and test robotics applications
MathWorks Toolbox

Robotics System Toolbox™ provides tools and algorithms for designing, simulating, and testing manipulators, mobile robots, and humanoid robots. For manipulators and humanoid robots, the toolbox includes algorithms for collision checking, trajectory generation, forward and inverse kinematics, and dynamics using a rigid body tree representation. For mobile robots, it includes algorithms for mapping, localization, path planning, path following, and motion control. The toolbox provides reference examples of common industrial robot applications. It also includes a library of commercially available industrial robot models that you can import, visualize, and simulate. You can develop a functional robot prototype by combining the kinematic and dynamic models provided. The toolbox lets you co-simulate your robot applications by connecting directly to the Gazebo robotics simulator. To verify your design on hardware, you can connect to robotics platforms and generate and deploy code (with MATLAB Coder™ or Simulink Coder™).

Categories
Autonomous Systems > Robotics System > Getting Started with Robotics System Toolbox
Autonomous Systems > Robotics System > Coordinate Transformations and Trajectories
Autonomous Systems > Robotics System > Manipulator Algorithm Design
Autonomous Systems > Robotics System > Mobile Robot Algorithm Design
Autonomous Systems > Robotics System > UAV Algorithm Design
Autonomous Systems > Robotics System > Code Generation
Autonomous Systems > Robotics System > Robot Modeling and Simulation

3)ROS Toolbox interface for ROS Custom Messages



ROS Toolbox interface for ROS Custom Messages
by MathWorks Robotics Team STAFF
Use custom ROS message and service types in MATLAB and Simulink.
MathWorks Optional Feature
22 Ratings
46 Downloads
Updated 11 Sep 2019

Editor's Note:

This support package is currently unable to download third-party software for MATLAB R2017a and earlier versions. For details and workaround, see this Bug Report.

MATLAB R2017b and later versions are unaffected.

Using the ROS Custom Messages Interface, you can extend the set of ROS messages supported by Robotics System Toolbox. You can define your own custom ROS message and service types, and use them in MATLAB and Simulink to communicate with other nodes in the ROS network.
This Interface allows you to:
- Generate MATLAB interfaces for custom messages and services from ROS msg and srv descriptions
- Send and receive custom messages just like any other ROS message
- Interface with any ROS-enabled robot or simulator that uses custom messages
- Share generated custom message interfaces with other MATLAB users
To install the ROS Custom Messages Interface:

Requires
Robotics System Toolbox
The functionality is supported on all platforms that MATLAB supports (32-bit Windows®, 64-bit Windows®, 64-bit Mac OS X, and 64-bit Linux ®)

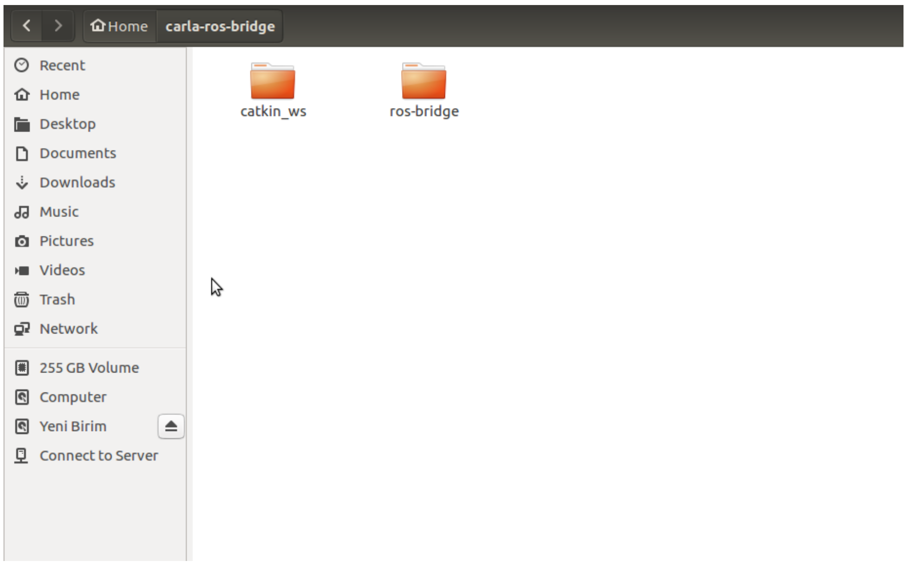
MATLAB Release Compatibility
Created with R2015a
Compatible with R2015a to R2019b

Platform Compatibility
☑ Windows ☑ macOS ☑ Linux

Categories
Autonomous Systems > ROS
Autonomous Systems > ROS > ROS Toolbox Supported Hardware

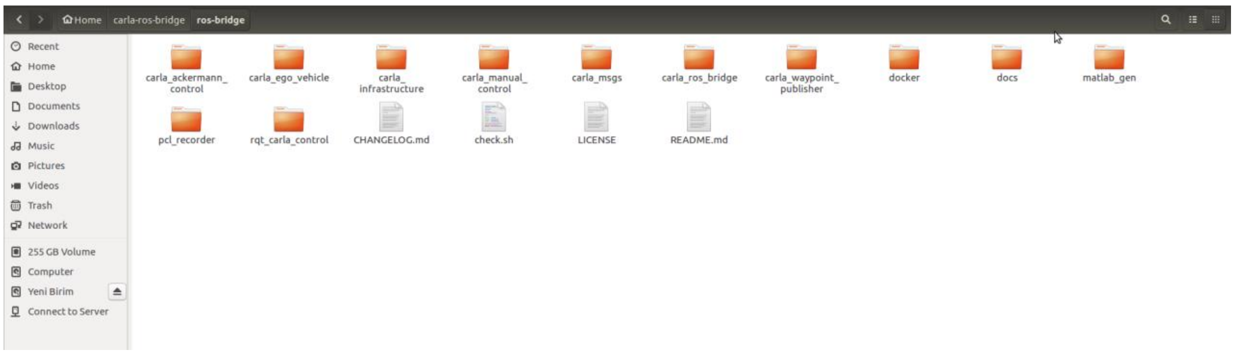
After installation is completed. Custom messages for Carla are created by using rosgenmsg command which is a function of ROS Toolbox interface for ROS Custom Messages.

Firstly, the folder is named as carla_msgs.rar inside gitlab repository should be unzip and copied into ros-bridge folder. This process is added since the MATLAB custom message create function does not accept some properties of Carla messages. After that, the address of Ros bridge directory should be used with the rosgenmsg command.



```
>> rosgenmsg('/home/aa/carla-ros-bridge/ros-bridge')
```

After this command, matlab_gen folder is created inside ROS bridge folder.



MATLAB prompts a guide with 3 steps. Firstly, javaclasspath.txt will be changed and saved. Secondly, msggen should be added into path.

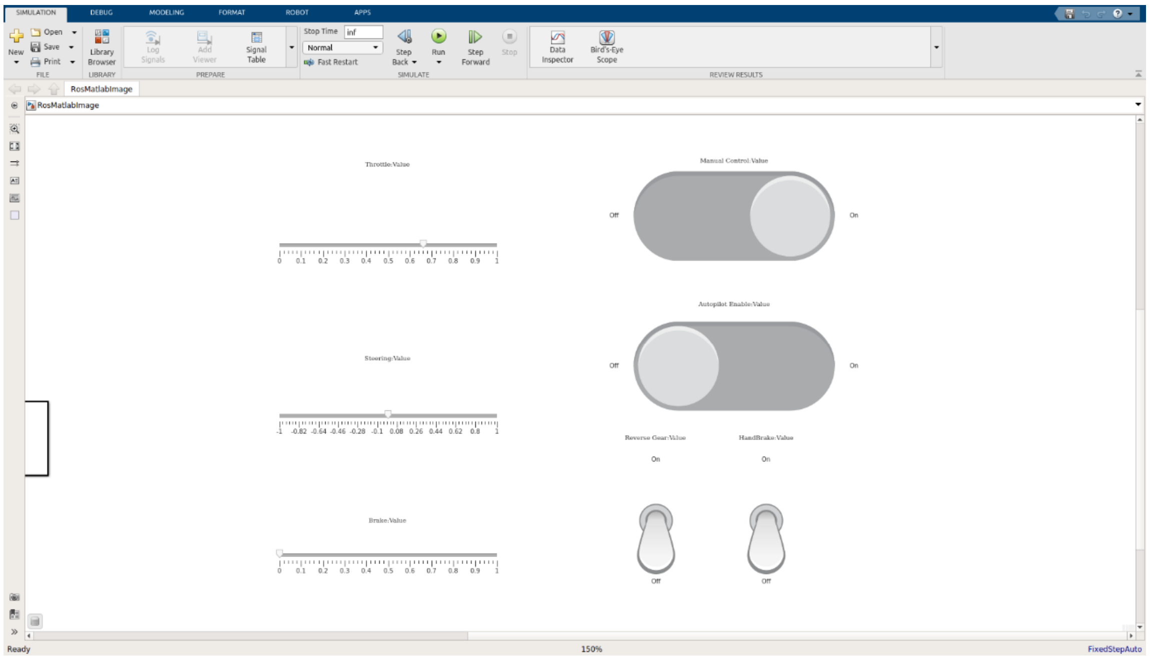
If user come across a problem with changing this step, user should change the pathdef.m file properties. For this purpose, user should open a file explorer with root access.

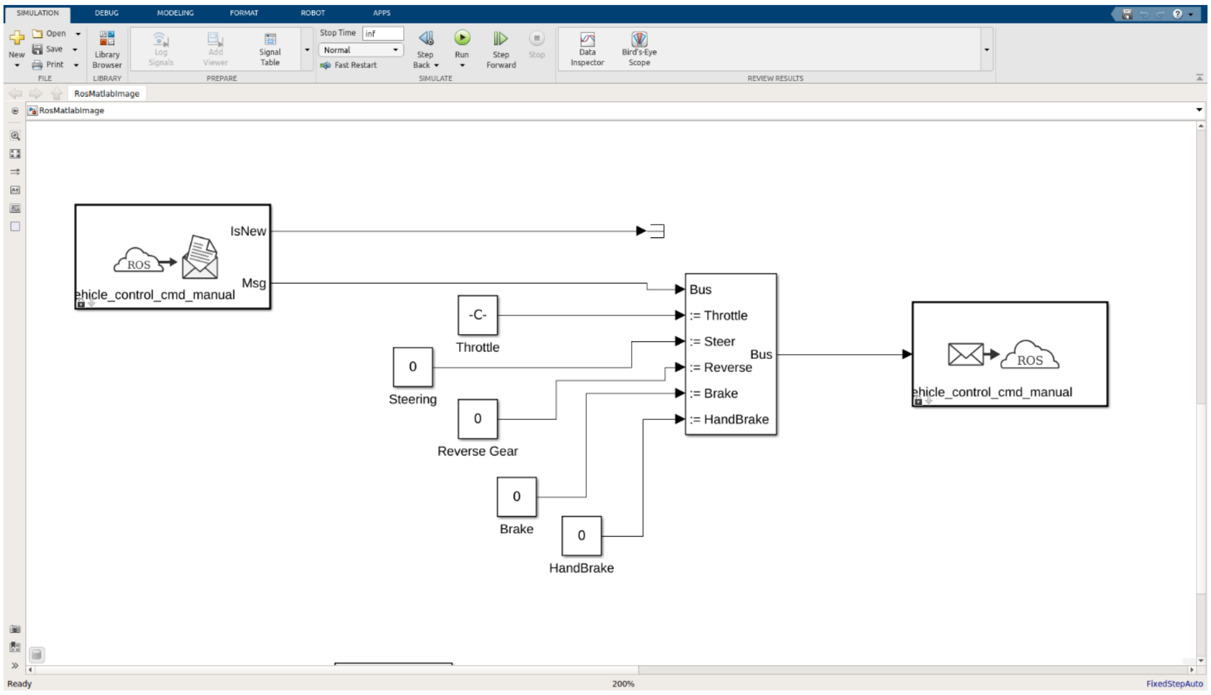
-sudo nautilus gives that permission and opens a new file explorer with root access. By this way, pathdef.m properties can be changed into Read and Write permission.

```
To use the custom messages, follow these steps:

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

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

2. Add the custom message folder to the MATLAB path by executing:

addpath('/home/aa/Desktop/carla-ros-bridge/ros-bridge/matlab_gen/msggen')
savepath

3. Restart MATLAB and verify that you can use the custom messages.
   Type "rosmsg list" and ensure that the output contains the generated
   custom message types.
```

The addition of custom messages for Carla in MATLAB can be checked by typing -rosmsg list into the command window of MATLAB. If Carla messages are seen in that list, user should see these messages in the Simulink ROS blocks.

We are going to complete the documentation with the final description of Simulink project which is located in gitlab repository under Carla-Ros Bridge directory. In this Simulink project, we use the Carla messages to change data by using ROS Subscription blocks. These data include throttle, brake, steering, handbrake and rear gear. Also, manual driving and autonomous driving selector are added.




SIMULATION
DEBUG
MODELING
FORMAT
ROBOT
APPS
New
Open
Save
Print
Library Browser
Log Signals
Add Viewer
Signal Table
Stop Time
inf
Normal
Fast Restart
Step Back
Run
Step Forward
Stop
Data Inspector
Bird's-Eye Scope
FILE
LIBRARY
PREPARE
SIMULATE
REVIEW RESULTS
RosMatlabImage
RosMatlabImage
IsNew
Msg
ROS
ehicle_control_cmd_manual
-C-
Throttle
0
Steering
0
Reverse Gear
0
Brake
0
HandBrake
Bus
:= Throttle
:= Steer
:= Reverse
:= Brake
:= HandBrake
Bus
ROS
ehicle_control_cmd_manual
Ready
200%
FixedStepAuto

Documentation of Carla-ROS using a more developed Hardware and using Ubuntu 18.04

- A new Hardware was used to test the installation process

Memory	8GB RAM
Processor	Intel Core i7 (9th Generation)
Graphics Card	Nvidia GTX 1650

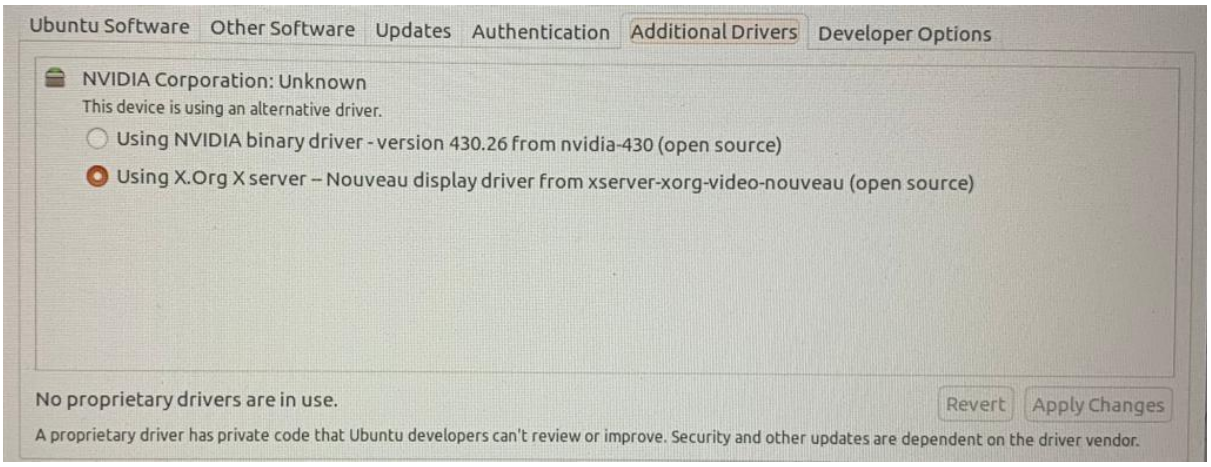
The installation process was as similar as the process of the old Hardware, but the first problem appeared while doing the initialization of Carla UE4 Editor, as the compatible Vulkan Driver was missing, so the old technique was tried in order to change to a Nvidia Proprietary, for GTX 1650, the 430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

```
sudo add-apt-repository ppa:graphics-drivers/ppa
```

```
sudo apt update
```

```
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 ()



NVIDIA Corporation: Unknown
This device is using an alternative driver.
○ Using NVIDIA binary driver - version 430.26 from nvidia-430 (open source)
● Using X.Org X server – Nouveau display driver from xserver-xorg-video-nouveau (open source)

No proprietary drivers are in use.
A proprietary driver has private code that Ubuntu developers can't review or improve. Security and other updates are dependent on the driver vendor.

Figure 12: Additional Driver in the new Hardware

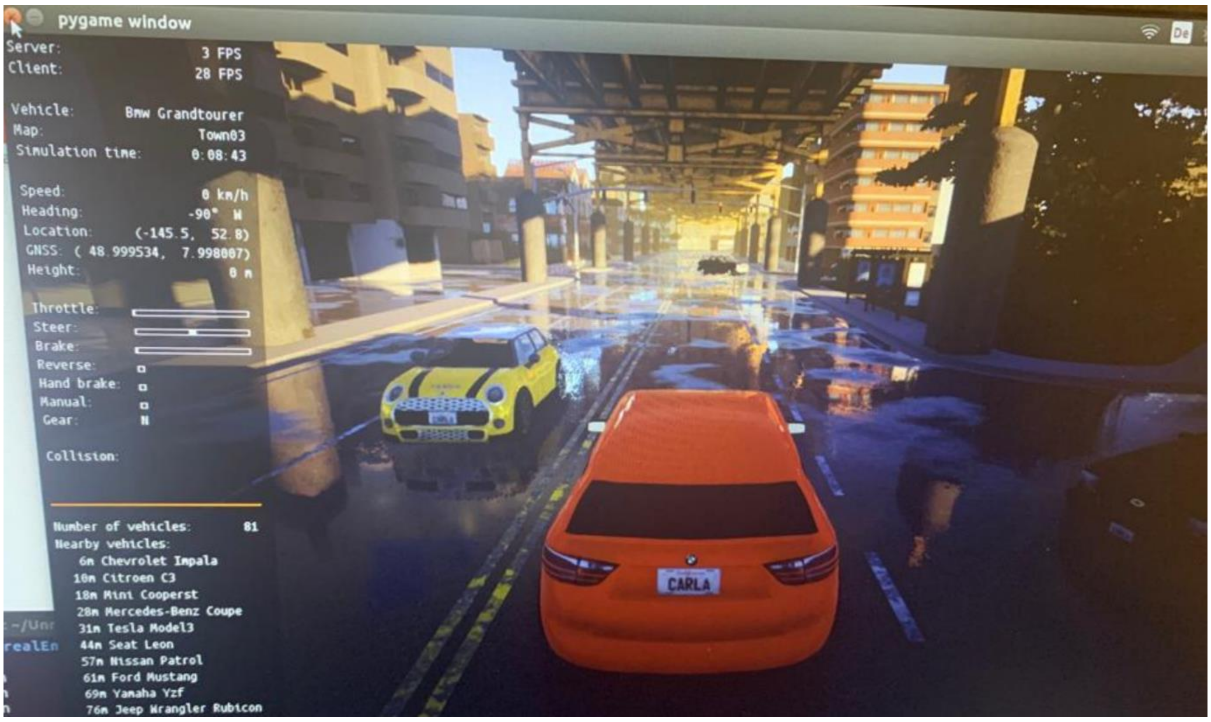
Therefore, installing a Vulkan-driver was a must using the following commands

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

```
sudo apt update
sudo apt upgrade
```

```
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 (13)



```
Server:                 3 FPS
Client:                28 FPS

Vehicle:       Bmw Grandtourer
Map:                    Town03
Simulation time:       0:08:43

Speed:                 0 km/h
Heading:               -90° W
Location:      (-145.5,  52.8)
GNSS: ( 48.999534,  7.998007)
Height:                   0 m

Throttle:
Steer:
Brake:
Reverse:
Hand brake:
Manual:
Gear:          N

Collision:

Number of vehicles:        81
Nearby vehicles:
  6m Chevrolet Impala
 10m Citroen C3
 18m Mini Cooperst
 28m Mercedes-Benz Coupe
 31m Tesla Model3
 44m Seat Leon
 57m Nissan Patrol
 61m Ford Mustang
 69m Yamaha Yzf
 76m Jeep Wrangler Rubicon
```

Figure 13: Pygame Window with the new Hardware

- For the Operating System Linux (Ubuntu 18.04), the main problem was installing ROS Kinetic, which only operates on Ubuntu 16.04, so the last resort was to install ROS Melodic, but the installation process did not work-out due to the lack of support for some packages.