Utilisation de CARLA avec ROS

Pour utiliser CARLA avec ROS, sur un système Ubuntu 20.04 vous devrez d'abord suivre ces procédures d'installations des logiciels :

→ ***Installation de ROS Noetic***
→ ***Installation de CARLA-ROS Bridge***

Ou alternativement sur Linux :

→ ***Installation de CARLA-ROS Bridge sur Docker***

Ce document examine en les caractéristiques de CARLA et de ROS, ainsi que dans les grandes lignes, le fonctionnement interne du CARLA ROS Bridge.

→ ***Documentation détaillée sur le fonctionnement théorique***

Ce document présente l'interfaçage entre CARLA et ROS, deux technologies fondamentales dans le domaine de la robotique et de la conduite autonome.

CARLA, un simulateur de conduite autonome développé par le groupe de recherche CARLA à l'Université d'Udine, fournit un environnement réaliste pour tester des algorithmes de conduite. ROS (Robot Operating System) est un framework logiciel open source couramment utilisé pour le développement de systèmes robotiques.

Le CARLA ROS Bridge permet la communication bidirectionnelle entre CARLA et ROS, permettant aux développeurs d'intégrer facilement les capacités de CARLA dans leurs projets basés sur ROS.

[Recommandé]
Pour la partie simplifier la mise en place et l'utilisation, nous allons commencer par créer des répertoires utiles pour les fichiers de configurations et les scripts.

```
mkdir -p ~/Documents/CARLA-ROS/{Configs,Scripts}
```

I) Lancement de CARLA-ROS-Bridge

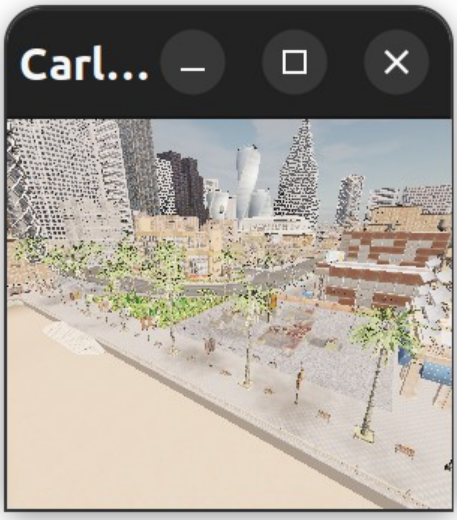
Pour faire la liaison entre CARLA et ROS, on va devoir lancer le pont entre les deux, et il existe plusieurs façon de démarrer carla-ros-bridge selon l'utilisation.

➢ Lancement du serveur CARLA

```
./CarlaUE4.sh -windowed -opengl
```

L'argument "-opengl" est nécessaire pour le bon fonctionnement avec GPU Nvidia. Ajustez en fonction de votre carte graphique.

On peut également choisir une résolution basse pour que les performances ne soient pas trop impactées ***« -ResX=320 -ResY=240 »****. Il existe également l'argument* ***« -quality-level=Low »*** *pour baisser la qualité.*



Une fenêtre comme celle-ci va alors s'ouvrir, il est nécessaire qu'elle reste ouverte pour laisser le serveur de CARLA tourner en fond.

Il s'agit de la carte par défaut, qui est modifiable, voir commande d'après.

➢ Modifier la configuration de Carla

```
python util/config.py -m Town02 --fps 10
```

On peut également modifier la configuration de la fenêtre serveur de CARLA qui s'adapte en temps réel (commande adaptable selon les besoins). Dans cet exemple, on charge la carte 2 et on modifie le taux de fps max à 10.

➢ Lancer Carla-ROS-Bridge dans le terminal

```
roslaunch carla_ros_bridge carla_ros_bridge.launch
```

C'est cette méthode que l'on va utiliser le plus souvent, car cela crée simplement un lien entre ROS et CARLA.
Cela nous permettra d'émettre des interactions entre CARLA et ROS par d'autres moyens (nous y reviendrons dans les parties suivantes).

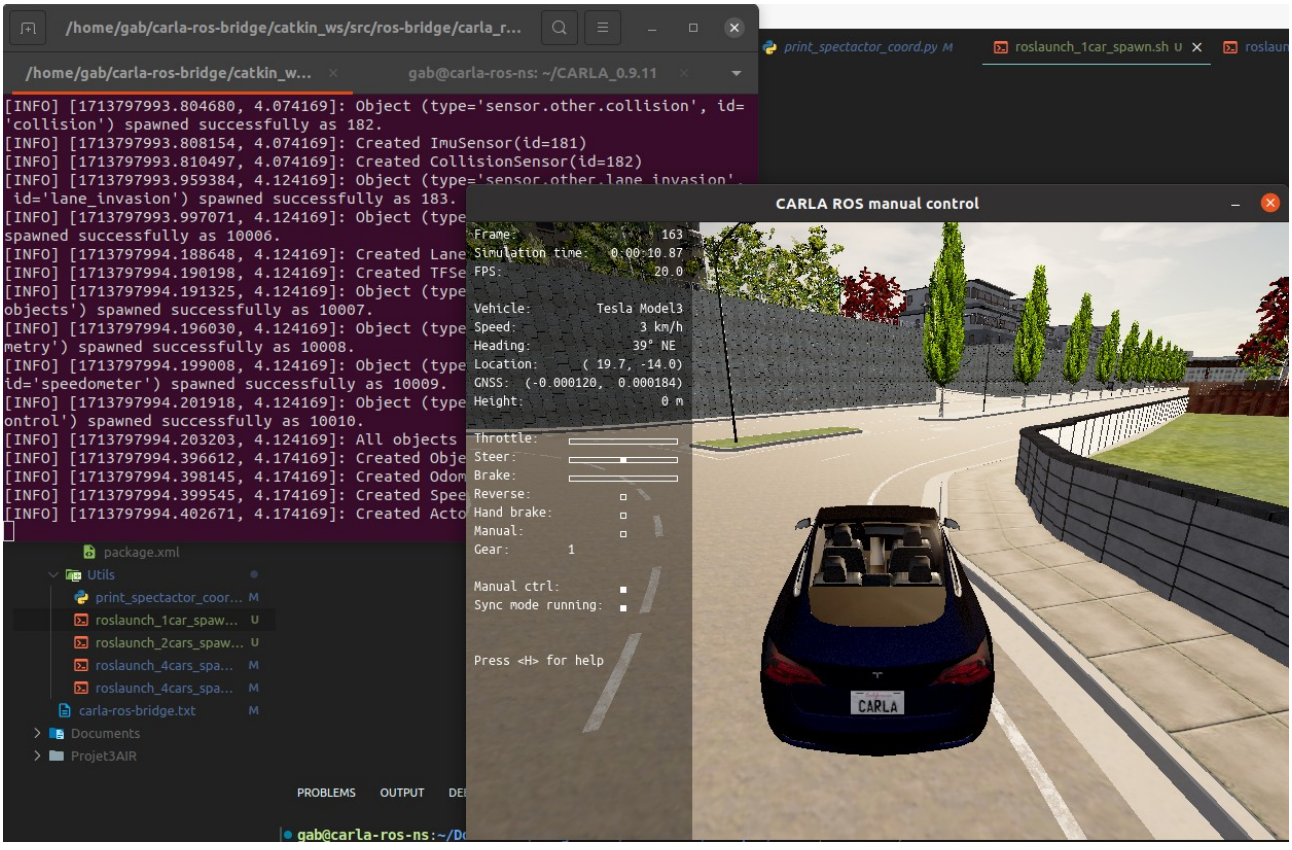
➢ Lancer Carla-ROS-Bridge avec un fichier de simulation

```
roslaunch carla_ros_bridge carla_ros_bridge_with_example_ego_vehicle.launch
```

Cette méthode permet de jouer directement une simulation CARLA en lançant le pont. Cette méthode peut être adéquate pour tester des simulations ou pour éviter d'ajouter des éléments « à la main ».

Voir les autres fichiers exemples dans le répertoire suivant (à adapter)

~/carla-ros-bridge/catkin_ws/src/ros-bridge/carla_ros_bridge/launch/



➢ Utilisation des « scenarios-runners »

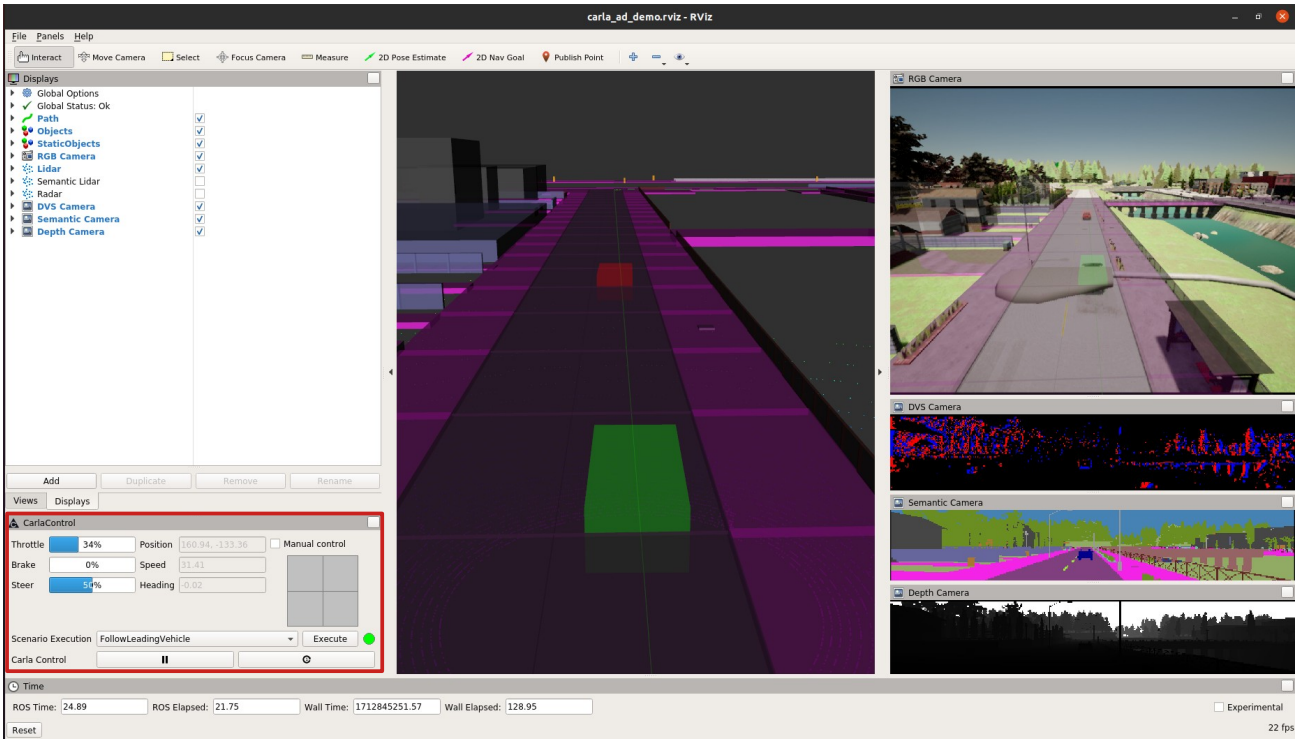
Les « scenarios-runners » sont des outils utilisés dans CARLA pour simuler et évaluer différents scénarios de conduite, tels que des situations de circulation, des manœuvres de stationnement ou des interactions avec des piétons.

1. Lancer un scénario dans Carla-ROS-Bridge

```
python scenario_runner.py --scenario FollowLeadingVehicle_1 –reloadWorld
```

2. Lancer un scénario avec l'interface graphique RVIZ

```
roslaunch carla_ad_demo carla_ad_demo_with_scenario.launch
```



À droite on a la simulation exemple comme dans CARLA

On a également les données de ROS dans l'interface graphique RVIZ
Dans la zone qu'indique l'encadré rouge on peut obtenir des informations sur le véhicule, dont la vitesse et la position.

II) Apparition d'objets dans CARLA via ROS

➢ Faire apparaître un véhicule avec les paramètres par défaut

```
roslaunch carla_spawn_objects carla_spawn_objects.launch
```

Les objets et leurs capteurs sont définis dans un fichier **.json**.
Ce fichier se trouve par défaut dans :

```
carla_spawn_objects/config/objects.json
```

On y définit différents paramètres pour les véhicules et les capteurs associés : type, id, position d'apparition, etc...

```
{
"objects":
  [
    {
      "type": "<SENSOR-TYPE>",
      "id": "<NAME>",
      "spawn_point": {"x": 0.0, "y": 0.0, "z": 0.0, "roll": 0.0, "pitch": 0.0, "yaw": 0.0},
      <ADDITIONAL-SENSOR-ATTRIBUTES>
    },
    {
      "type": "<VEHICLE-TYPE>",
      "id": "<VEHICLE-NAME>",
      "spawn_point": {"x": -11.1, "y": 138.5, "z": 0.2, "roll": 0.0, "pitch": 0.0, "yaw": -178.7},
      "sensors":
        [
        <SENSORS-TO-ATTACH-TO-VEHICLE>
        ]
    }
    ...
  ]

}
```

Exemple de la syntaxe d'un fichier de configuration

➢ Faire apparaître un objet avec sa propre configuration

1. Création du fichier configuration

Se déplacer dans le répertoire

```
cd ~/Documents/CARLA-ROS/Configs/
```

→ Créez un fichier de configuration en vous référant à la documentation

```
touch config1.json && nano config1.json
```

→ Ou récupérez des exemples de fichier de configuration sur github

Vous pouvez également cloner le répertoire git.

```
git clone https://github.com/Gabrieleirbag1/Stage-2024.git
```

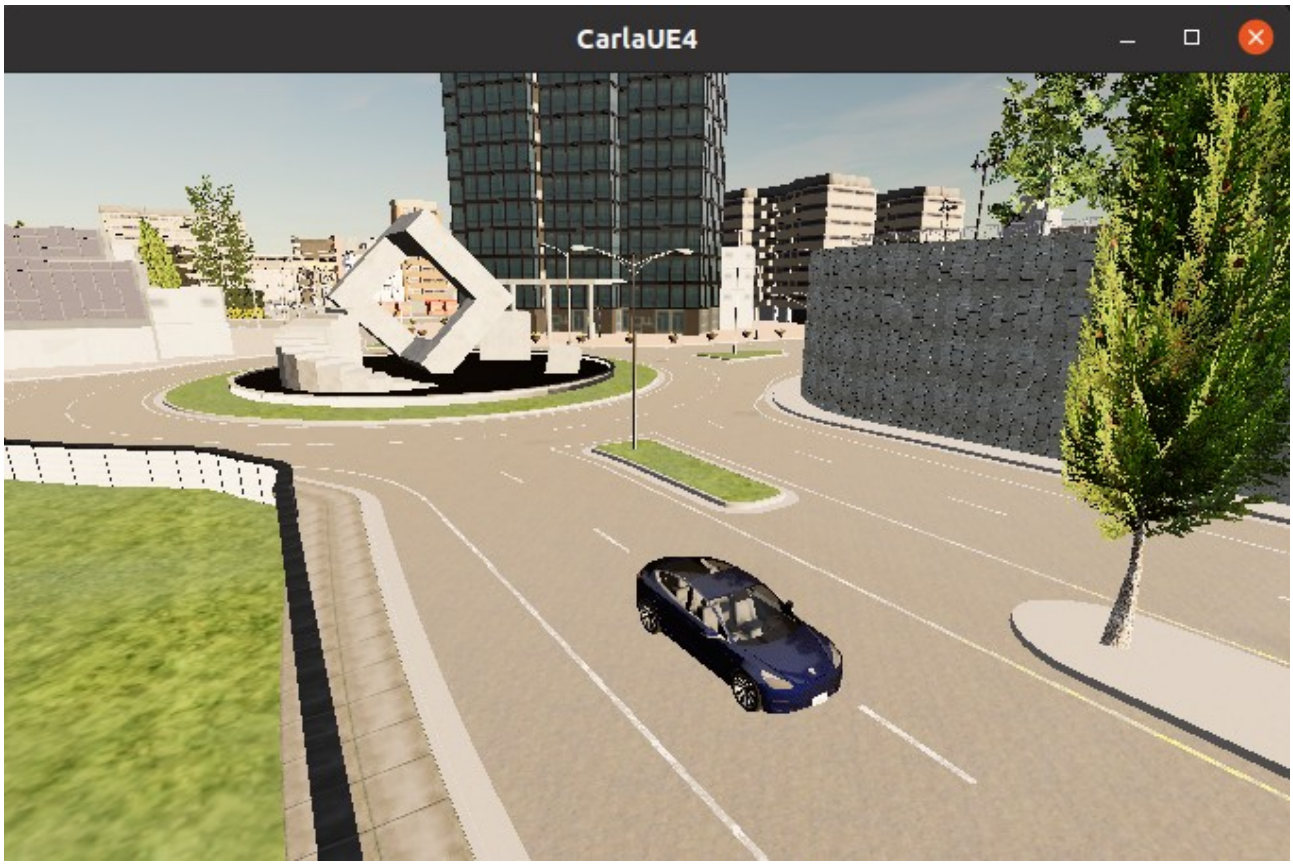
2. Utilisation des fichiers de configuration

```
roslaunch carla_spawn_objects carla_spawn_objects.launch \
objects_definition_file:=~/Documents/CARLA-ROS/Configs/config1.json
```

*Avec l'argument « **objects_definition_file** » on peu spécifier un autre fichier de configuration.*

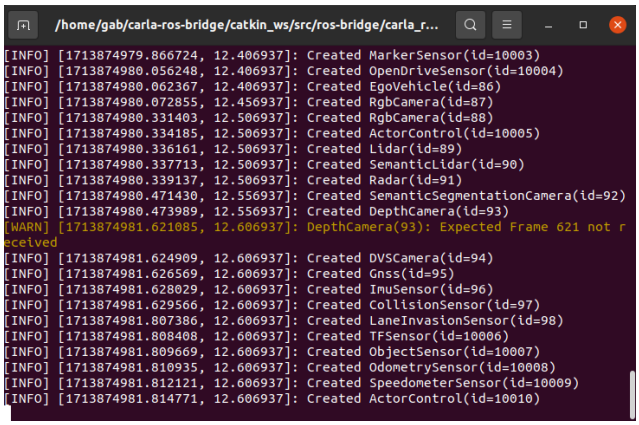
*Dans cet exemple on a ajouté une zone d'apparition spécifique dans la variable « **spawn_point** ».*
*On peut également constaté qu'on peu assignées des capteurs en les déclarant dans la variable « **sensors** ».*

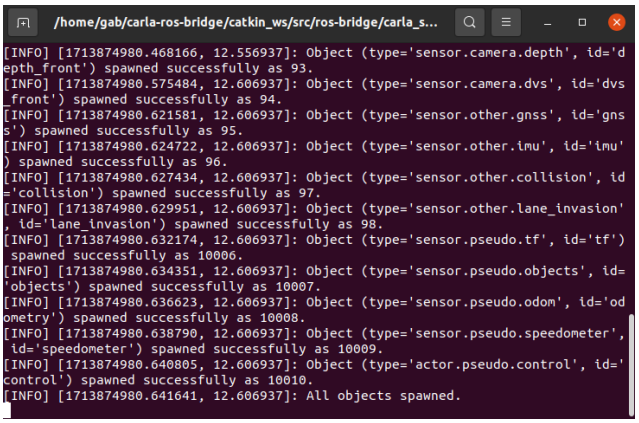
```
"type": "vehicle.tesla.model3",
"id": "ego_vehicle",
"spawn_point": {"x": 41.3 "y": -6.2, "z": 3.1, "roll": 0.0, "pitch": 0.0, "yaw": 0.0},
"sensors": {
                […]
            }
```



Capture d'écran d'un véhicule Tesla sur la carte Town03 sur la route

Dans les terminaux **« carla-ros-bridge »** et **« carla_spawn_object »** on retrouve les logs qui confirment l'apparition des différents capteurs et du véhicule.





*Fermer le terminal « **carla_spawn_object** » provoquera la suppression de l'objet et des capteurs de la voiture.*

3. Utilisation d’une commande pour le paramétrage

```
roslaunch carla_spawn_objects carla_spawn_objects.launch \
spawn_point_ego_vehicle:=2.3,0.5,63.1,0.0,0.0,0.0
```

On choisit un point d’apparition précis, ici ce sera dans le ciel au dessus de la statue sur la carte Town03.

On peut également observer la physique de collision du simulateur.

Capture d'écran d'un véhicule Tesla sur la carte Town03 dans le ciel

Pour simplifier l’usage des coordonnées avec les points d’apparition, vous pouvez utiliser ce script python qui renvoie les coordonnées du spectateur quand le serveur CARLA est ouvert (trouvable ici également).

```
gab@carla-ros-ns:~$ python3 print_spectactor_coord.py
(x,y,z) = (0.0,0.0,0.0)
(x,y,z) = (27.501983642578125,-8.681471824645996,12.999210357666016)
(x,y,z) = (22.54092788696289,-3.8097288608551025,19.053464889526367)
(x,y,z) = (18.518049240112305,-0.7710981369018555,20.82220458984375)
(x,y,z) = (17.666133880615234,-0.2543177306652069,20.73749542236328)
```

Capture d’écran réalisé après des déplacements sur CARLA

III) Communication entre CARLA et ROS

➢ Notions théoriques

1. Les capteurs

Lors de la création d'un véhicule dans CARLA, il est courant d'incorporer divers capteurs pour accroître son niveau de perception et de contrôle. Ces capteurs comprennent notamment des caméras, des LIDAR, des radars, et d'autres dispositifs sensoriels pour assurer la gestion correcte des aléas de l'environnement.

Les capteurs sont utilisés pour collecter des données sur l'environnement, y compris la détection des obstacles, la reconnaissance des panneaux de signalisation, la localisation par rapport à la carte, etc.

L'ajout de ces capteurs permet non seulement d'améliorer la perception de l'environnement par le véhicule, mais aussi de faciliter la communication avec d'autres systèmes via les topics ROS.

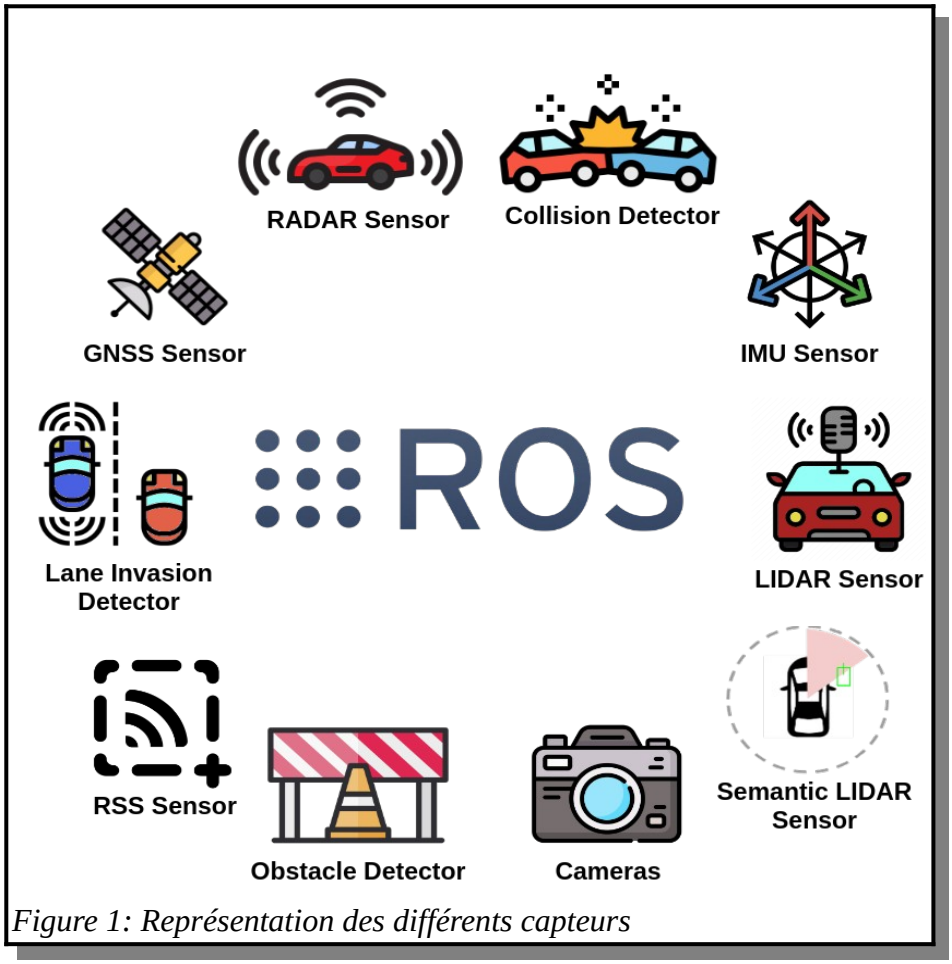


Figure 1: Représentation des différents capteurs

2. Les nœuds et les topics ROS

Dans l'écosystème ROS et CARLA, les nœuds sont comme des petits programmes qui font des tâches spécifiques, comme lire des capteurs ou contrôler des véhicules. Les topics sont comme des canaux où ces nœuds peuvent envoyer ou recevoir des informations de manière asynchrone.

Par exemple, un nœud qui lit les données d'un capteur peut les envoyer sur un topic. Ensuite, un autre nœud peut s'abonner à ce topic pour lire ces données et les utiliser pour contrôler un véhicule ou prendre des décisions. C'est comme si les nœuds se parlaient entre eux en passant des messages par les topics, ce qui rend tout le système capable de fonctionner ensemble de manière coordonnée.

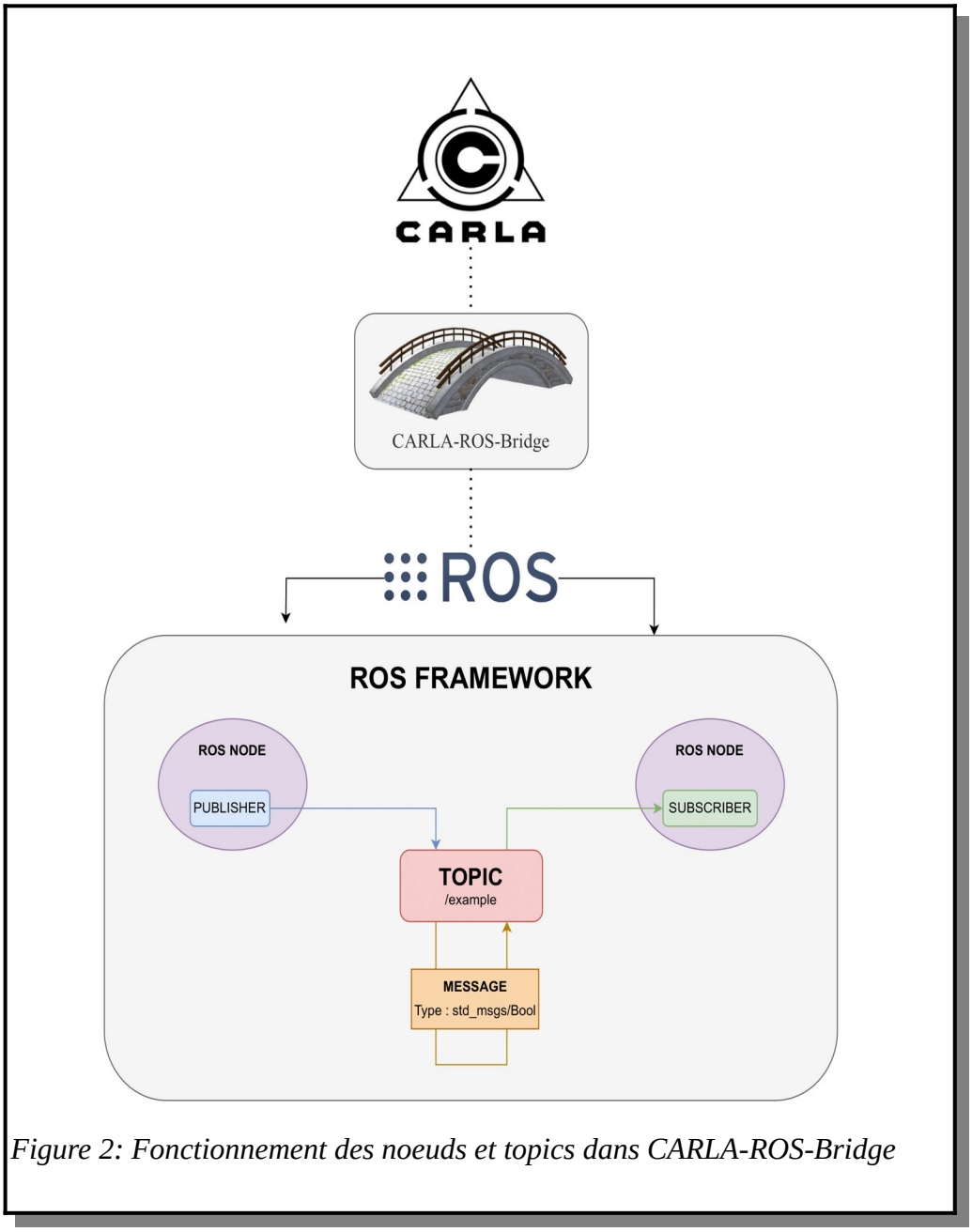


Figure 2: Fonctionnement des noeuds et topics dans CARLA-ROS-Bridge

➢ Comprendre les nœuds et les topics ROS

1. Affichez les différents nœuds (dans la continuité de notre exemple)

```
rosnode list
```

```
gab@carla-ros-ns:~/Documents/Stage-2024$ rosnode list
/carla_ros_bridge
/carla_spawn_objects_carla_ros_ns_5577_321624702039504411
/rosout
```

*Le nœud « **/carla_ros_bridge** » permet l'échange de données bidirectionnel entre CARLA et ROS.*
*Le nœud « **/carla_spawn_objects_[...]** » est spécifiquement utilisé pour générer et introduire des objets, comme dans notre cas des véhicules et des capteurs.*
*Enfin, le nœud « **/rosout** » agit comme un canal de sortie pour les messages de débogage et de journalisation provenant de ROS.*

2. Affichez les topics d'un nœud

```
rosnode info /carla_ros_bridge
```

On va alors obtenir la liste des topics répartis en 4 catégories :
- **Publications** : Diffusion active de données vers un topic spécifique.
- **Subscriptions** : Réception passive de données provenant d'un topic spécifique.
- **Services** : Permettent d'envoyer des demandes pour effectuer des actions spécifiques dans le système ROS.
- **Connected** : Indique le nombre de nœuds actuellement connectés au topic spécifié.

On peut voir par exemple que le nœud **« /carla_ros_bridge »** est connecté au nœud ***« /carla_spawn_objects_[...] »**, que nous avons ajouté précédemment (encadré).*

```
Connections:
 * topic: /rosout
    * to: /rosout
    * direction: outbound (44015 - 127.0.0.1:34354) [9]
    * transport: TCPROS
 * topic: /clock
    * to: /carla_spawn_objects_carla_ros_ns_5577_321624702039504411
    * direction: outbound (44015 - 127.0.0.1:34360) [10]
    * transport: TCPROS
```

➢ Contrôler un objet dans CARLA avec ROS

1. Contrôle manuel via les topics ROS

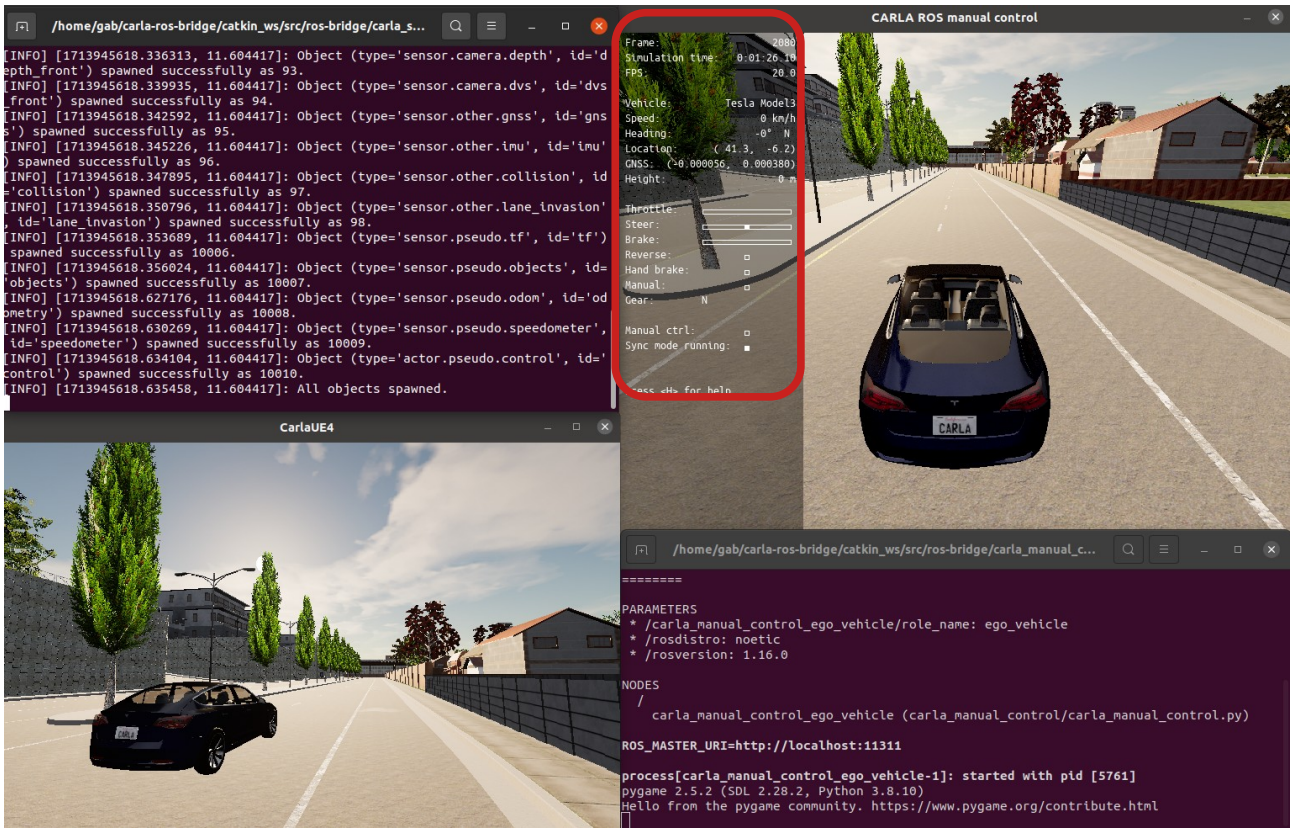
```
rostopic pub /carla/ego_vehicle/vehicle_control_cmd \
carla_msgs/CarlaEgoVehicleControl "{throttle: 1.0, steer: 1.0}" -r 10
```

Dans cet exemple, on publie sur le topic **« vehicle_control_cmd »** une commande de contrôle qui va déclencher une accélération et une rotation du véhicule.

Cette commande ROS envoie la commande de déplacement pour tous les véhicules générés dans CARLA via les topics ROS.

On peut également contrôler manuellement un véhicule, ou simplement observer son comportement via le script « **manual_control.launch »** par exemple.

```
roslaunch carla_manual_control carla_manual_control.launch timeout:=300
```



Partie gauche : carla_spawn_objects, serveur CARLA
Partie droite : interface de manual_control.py, carla_manual_control.launch

Ce script agit comme une interface utilisateur permettant à l'utilisateur de donner des commandes de contrôle au véhicule, telles que l'accélération, le freinage, le volant, etc, qui sont retranscrites visuellement (voir encadré rouge).

Ces commandes sont ensuite transmises au simulateur CARLA, qui les interprète pour modifier l'état du véhicule dans la simulation via ROS.

2. Conduite en pilote automatique via les topics ROS

```
rostopic pub /carla/ego_vehicle/enable_autopilot std_msgs/Bool "data: true"
```

*Le type de message de ce topic est booléen, alors on renseigne **« true »** pour activer le pilote automatique.*
Une fois le mode autopilote activé les capteurs sont essentiels pour permettre au véhicule de fonctionner.
Les capteurs collectent des informations sur l'environnement et permettent au véhicule de prendre des décisions de conduite autonome en fonction de ces données.

Vidéo 1: La voiture se replace sur la voie et se pilote automatiquement

➢ Lire et comprendre l'échange de données entre nœuds

1. Consulter l'état d'un topic ROS

rostopic info **<TOPIC_NAME>**

On peut vérifier l'état d'un topic à partir des informations suivantes :
→ ***Type :*** *Le type de message communiqué*
→ ***Publishers :*** *Les nœuds publicateurs*
→ ***Subscribers :*** *Les nœuds abonnés*

Reprenons notre exemple précédent :

Avant d'avoir commencé la publication :

```
gab@carla-ros-ns:~/Documents/Stage-2024$ rostopic info /carla/ego_vehicle/enable_autopilot
Type: std_msgs/Bool

Publishers: None

Subscribers:
 * /carla_ros_bridge (http://carla-ros-ns:39631/)
```

Après avoir commencé la publication :

```
gab@carla-ros-ns:~/Documents/Stage-2024$ rostopic info /carla/ego_vehicle/enable_autopilot
Type: std_msgs/Bool

Publishers:
 * /rostopic_8743_1713953884007 (http://carla-ros-ns:44095/)

Subscribers:
 * /carla_ros_bridge (http://carla-ros-ns:39631/)
```

*→ On constate qu'il est effectivement abonné au nœud « **/caral-ros-bridge** » qui est notre nœud permettant le pont entre CARLA et ROS.*

*→ On constate qu'il publie sur un nouveau nœud « **/rostopic_[...]** », et si on vérifie les nœuds, on observe que « **/rostopic_[...]** » a bien été ajouté.*

```
gab@carla-ros-ns:~/Documents/Stage-2024$ rosnode list
/carla_ros_bridge
/carla_spawn_objects_carla_ros_ns_12201_5980916997876273350
/rosout
/rostopic_12655_1713960857389
```

2. Afficher les informations d’un topic

rostopic echo **<TOPIC_NAME>**

*On va afficher le topic « **vehicle_status** » et on va pouvoir obtenir en temps réel des informations sur l’état du véhicule en continue dont :*
*- **Acceleration linear (xyz) :** Les coordonnées du véhicule*
*- **Velocity :** La vitesse du véhicule*
*- **Control :** les contrôles utilisés (frein, accélérateur, etc.)*
*- **Orientation :** L’orientation du véhicule sur la carte*
- ...

```
gab@carla-ros-ns:~/Documents/Stage-2024$ rostopic echo
/carla/ego_vehicle/vehicle_status
header:
  seq: 41
  stamp:
    secs: 2665
    nsecs: 497636247
  frame_id: "map"
velocity: 4.87266206741333
acceleration:
  linear:
    x: -0.085591971874237 06
    y: -0.00244140625
    z: 4.0241029637400 06e-05
  angular:
    x: 0.0
    y: 0.0
    z: 0.0
orientation:
  x: -0.0001486497665642078
  y: 0.00012137905495541854
  z: 0.7153452900291761
  w: 0.6987711207566077
```

```
control:
  header:
    seq: 0
    stamp:
      secs: 0
      nsecs:         0
    frame_id: ''
  throttle: 0.3793737292289734
  steer: -0.010414493270218372
  brake: 0.0
  hand_brake: False
  reverse: False
  gear: 1
  manual_gear_shift: False
```

IV) Interfaçage de CARLA et ROS en utilisant Python

➢ Préparation de l'environnement

1. Construction de l’arborescence

Pour utiliser Python avec CARLA, il est nécessaire de mettre en place une arborescence de fichiers appropriée. Cela implique la création d'un environnement de développement dans lequel les scripts Python peuvent être exécutés en interaction avec CARLA, et la construction d’un **« package catkin »**.

→ Pour la création de cette arborescence, référez vous à cette documentation.

On obtient alors une arborescence comme celle-ci.

```
.
└── Scripts/
    └── catkin_ws /
        └── src /
            └── sub_pkg /
                └── src/
                    ├── python_script1.py
                    ├── python_script2.py
                    └── python_script3.py
```

2. Objectifs

L'utilisation de fichiers Python facilite l'intégration de CARLA et ROS en simplifiant le processus de développement et de débogage. Pour communiquer entre CARLA et ROS, ce langage offre flexibilité et adaptabilité.

Dans les étapes qui vont suivre, nous allons utiliser des scripts personnalisés qui s'abonnent aux topics ROS pour recevoir et traiter en temps réel les données des capteurs du véhicule simulé, facilitant ainsi le contrôle et l'analyse des performances des véhicules autonomes.

➢ Utilisation de Python

Vous pouvez récupérez des scripts python en guise d'exemple ici.

Exécuter un script python

```
rosrun sub_pkg <PYTHON_FILE_NAME>.py
```

Voici un exemple de résultat de deux scripts python récupérant des données sur le véhicule dans CARLA (coordonnées géographiques, vitesse, etc.).

```
^Cgab@carla-ros-ns:~/Documents/Stage-2024/CARLA-ROS/Scripts/Subscribers/catkin_ws$ rosrun sub_pkg subscribegnss.py
y
[INFO] [1714039427.017438, 137.192038]: Received NavSatFix data:
[INFO] [1714039427.018568, 137.192038]: Latitude: 0.000222
[INFO] [1714039427.019488, 137.192038]: Longitude: -0.000122
[INFO] [1714039427.020308, 137.192038]: Altitude: 2.119429
gab@carla-ros-ns:~/Documents/Stage-2024/CARLA-ROS/Scripts/Subscribers/catkin_ws$ $ rosrun sub_pkg subscribeimy.py
[INFO] [1714039431.042388, 138.542038]: Received Imu data:
[INFO] [1714039431.043509, 138.542038]: Linear Acceleration: x=0.496962, y=-0.002661, z=9.797404
[INFO] [1714039431.044428, 138.542038]: Angular Velocity: x=-0.000000, y=0.000000, z=-0.000000
[INFO] [1714039431.045319, 138.542038]: Orientation: x=-0.025308, y=-0.001229, z=-0.998228, w=0.053846
```

Voici l'un d'entre eux en détail :

```python
#!/usr/bin/env python3
import rospy, time
from std_msgs.msg import Float32 # Import du type de message pour Float32

# Affiche en direct au travers de terminaux ROS les informations dont nous avons besoin
def callback(data):
    rospy.loginfo("Received data: %f", data.data)
    time.sleep(1)

def listener(): # Definition de notre subscriber
    rospy.init_node('subscriber_speedometer', anonymous=True) # Nom de notre noeud
    # Nous indiquons à quelle publishers nous nous souscrivons
    rospy.Subscriber("/carla/ego_vehicle/speedometer", Float32, callback)
    rospy.spin() # Permet au programme d'être en route continuellement

if __name__ == '__main__':
    listener()
```

Ce script Python crée un nœud ROS qui s'abonne à un topic spécifique ("/carla/ego_vehicle/speedometer") pour recevoir des données de vitesse du véhicule simulé dans CARLA. À chaque réception de données, il affiche ces informations en temps réel dans les terminaux ROS.