Utilisation de NS3 avec ROS et CARLA Simulator

Consultez la documentation d'installation de NS3 pour communiquer avec ROS et CARLA Simulator, et le site de NS3 pour des informations complémentaires.

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***

Pour plus d'informations sur l'utilisation ainsi que le fonctionnement de CARLA-ROS-Bridge, consultez la documentation suivante :

- → ***Utilisation de CARLA avec ROS***
- → ***Fonctionnement théorique***

Pour commencer, il est important de rappeler que CARLA est un simulateur de conduite automobile, mais c'est via ROS et le CARLA-ROS-Bridge que nous contrôlerons les véhicules dans CARLA. NS3, quant à lui, simulera les échanges réseau entre les véhicules présents dans CARLA.

L'objectif de ce projet est, dans un premier temps, de permettre la communication entre les véhicules dans CARLA afin qu'ils puissent échanger, par exemple, leurs données de position ou leurs vitesses. Par exemple, si deux véhicules risquent de se percuter, ils pourraient échanger leurs données de position, ce qui permettrait de prévenir les dangers en déclenchant, par exemple, un freinage d'urgence. Cependant, il n'est pas possible d'effectuer des échanges réseau directement dans CARLA.

C'est là qu'intervient NS3. Le but est de représenter les véhicules de CARLA dans NS3 en créant des **nœuds** qui agiront comme des « avatars » de ces véhicules. Ces nœuds seront connectés via un réseau Wi-Fi et pourront communiquer des messages de position ou autre directement.

Enfin, les données seront renvoyées à un script Python ROS qui pourra les analyser et déclencher des actions telles que le freinage ou un virage. Il est important de rappeler que l'on simule un environnement réel et c'est pourquoi il est nécessaire de faire tout ce cheminement, même s'il on possède déjà toutes les données avec ROS.

I) Déplacement des véhicules et positionnement des nœuds dans NS3

Il est important pour faire fonctionner les scripts python de construire une arborescence catkin afin d'avoir les dépendances nécessaires. Suivez cette documentation si l'installation n'a pas encore été effectuée.

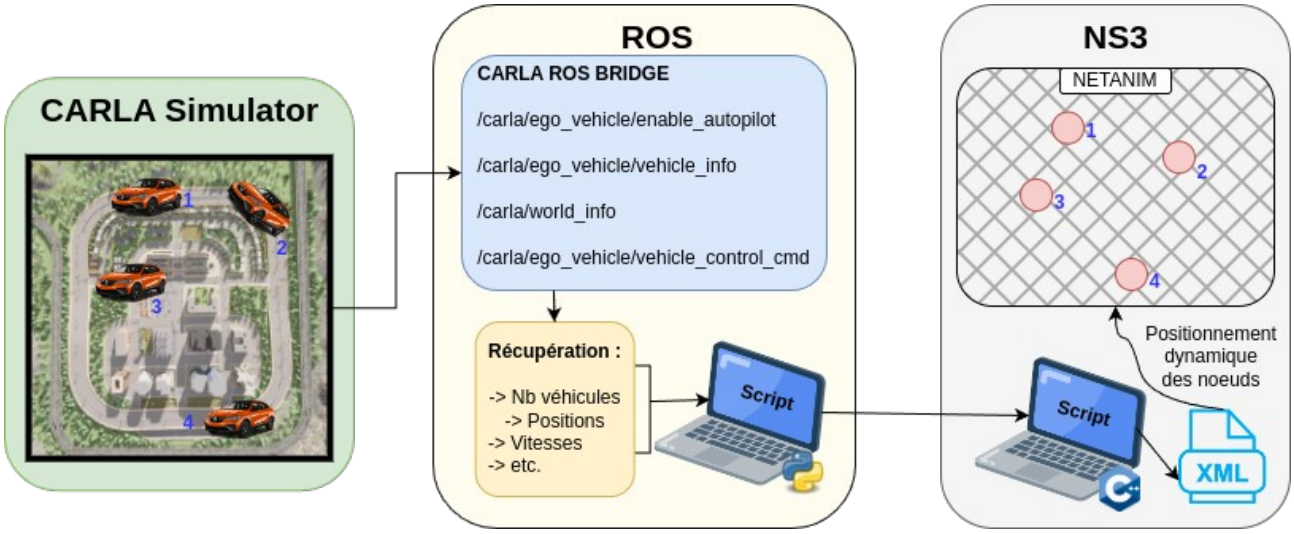

Figure 1: Schéma de la communication de CARLA et ROS vers NS3

Les deux scripts suivants vont permettre de récupérer les données positionnelles des véhicules de CARLA vers NS3, pour pouvoir les analyser par la suite dans Netanim.

- Lancer le script dans ns3

```
./waf --run rosns3_socket_xml.cc
```

- Lancer le script python publisher_waypoint_seq.py

```
rosrun sub_pkg publisher_waypoint_seq.py
```

- Lire l'animation dans Netanim

```
./Netanim
```

Vous avez à disposition une vidéo exemple de déplacement de nœuds dans Netanim.

II) Échanges Wi-fi entre les véhicules dans NS3, retour vers ROS

Il est important pour faire fonctionner les scripts python de construire une arborescence catkin afin d'avoir les dépendances nécessaires. Suivez cette documentation si l'installation n'a pas encore été effectuée.

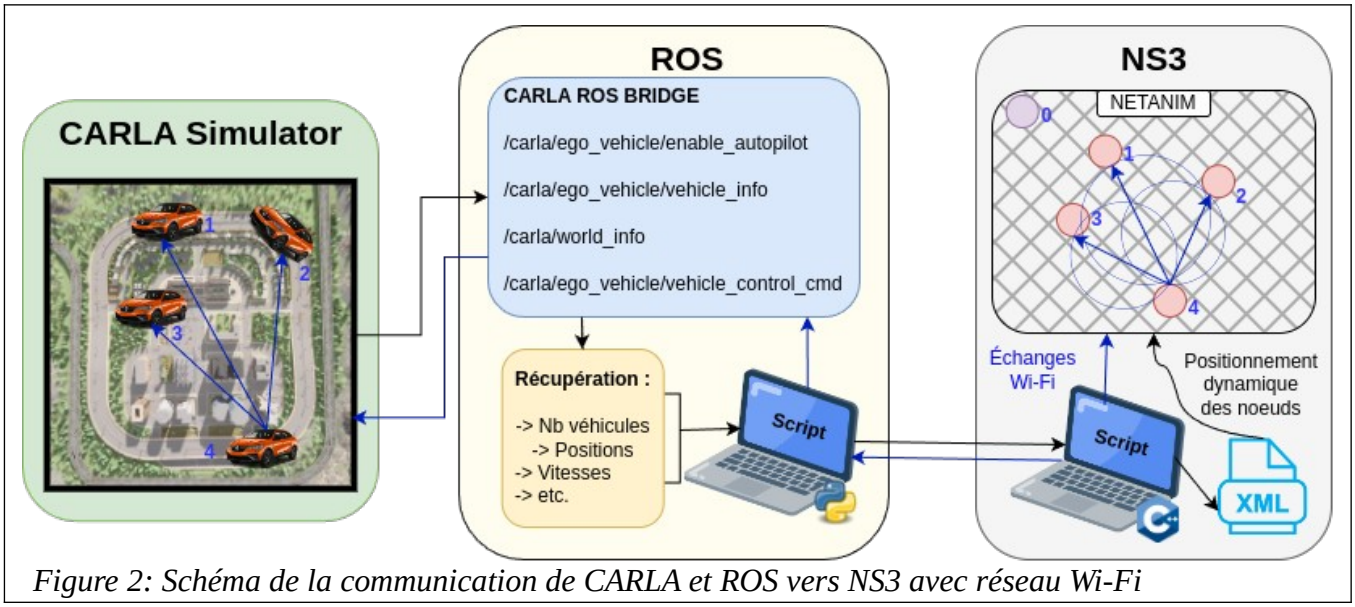

Figure 2: Schéma de la communication de CARLA et ROS vers NS3 avec réseau Wi-Fi

Les trois scripts suivants permettront de transférer les données positionnelles des véhicules de CARLA vers NS3. Ensuite, les nœuds correspondants aux véhicules seront intégrés dans un réseau Wi-Fi et échangeront des paquets contenant les données de position. Il sera possible d'observer en temps réel ces échanges de paquets dans CARLA grâce à des flèches dessinées dynamiquement via le script 1. Enfin, les données positionnelles et les échanges de données pourront être analysés dans Netanim via le script 2 (à la fin).

- Lancer les scripts dans ns3 dans deux terminaux distincts

Script 1

```
./waf --run "rosns3_adhoc_realtime_broadcast --time=120.0"
```

Script 2

```
./waf --run "rosns3_adhoc_netanim_broadcast --time=120.0"
```

Ici nous utiliserons les scripts qui permettent des échanges en **broadcast** dans le réseau Wi-Fi, mais la variante des scripts en **unicast** est également dans le github.

L’agument --time permet de définir le temps de simulation que l’on souhaite. La valeur par défaut est "30.0". Il existe d’autres arguments également.

```
CommandLine cmd;
cmd.AddValue ("phyMode", "Wifi Phy mode", phyMode);
cmd.AddValue ("rss", "received signal strength", rss);
cmd.AddValue ("packetSize", "size of application packet sent", packetSize);
cmd.AddValue ("numPackets", "number of packets generated", numPackets);
cmd.AddValue ("interval", "interval (seconds) between packets", interval);
cmd.AddValue ("verbose", "turn on all WifiNetDevice log components", verbose);
cmd.AddValue ("time", "setup the time duration of the simulation", time_duration);
cmd.Parse (argc, argv);
```

- Lancer le script python publisher_waypoint_seq_rep.py

```
rosrun sub_pkg publisher_waypoint_seq_rep.py
```

- Observez les flèches apparaître entre les voitures dans CARLA Simulator

- Lire l’animation dans Netanim

```
./Netanim
```

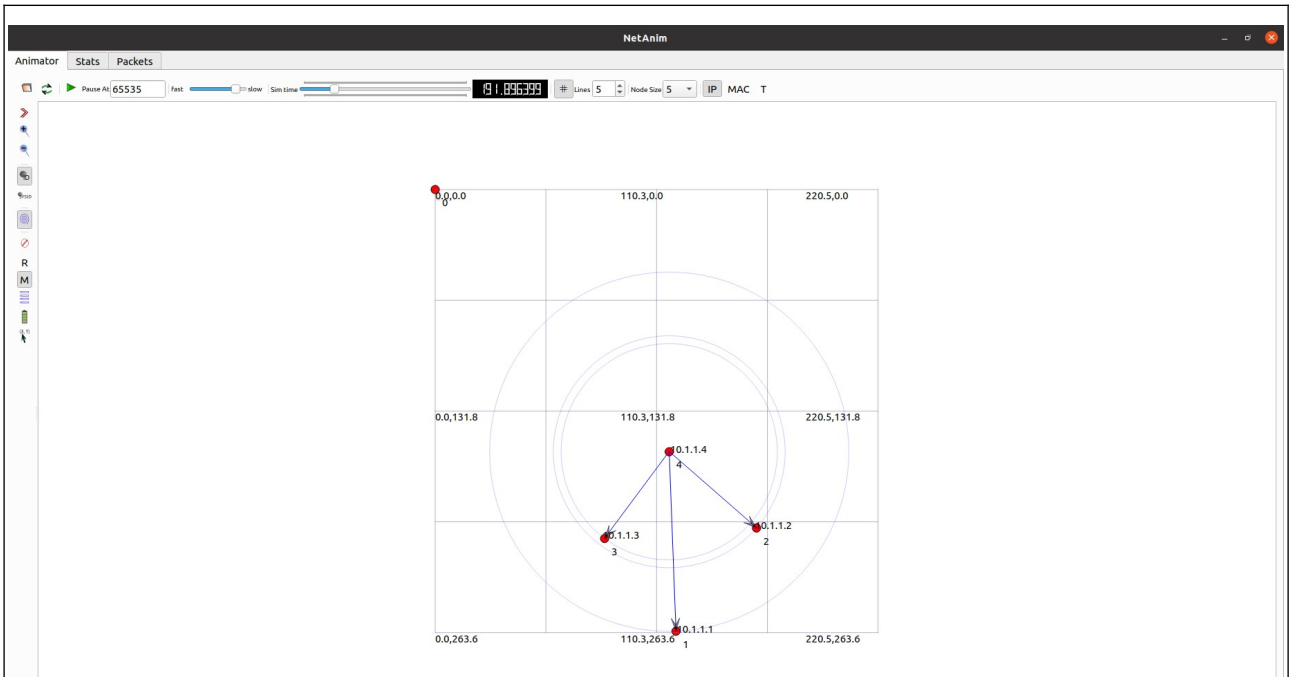

Figure 3: Simulation lue dans Netanim à titre d'exemple

Vous avez également à disposition ce tutoriel vidéo de deux minutes qui montre visuellement ces étapes avec des explications complémentaires.

Vous trouverez également, à titre d’exemple, une vidéo illustrant une simulation avec des échanges Wi-Fi en unicast, ainsi qu’une autre vidéo montrant des échanges Wi-Fi en broadcast. Il est intéressant de noter que, dans le cas du broadcast, la réduction du nombre de paquets permet une plus grande précision dans la reproduction des positions des véhicules lors du déplacement des nœuds dans NS3.

Remarque :

Le script python ROS récupère les données renvoyées par le script NS3 via la classe « ManageResponse » mais ne les gère pas. Un problème existant dans le CARLA-ROS-Bridge est l’incapacité à contrôler un véhicule indépendamment des autres, il reste donc d’autres solutions à trouver comme par exemple un contrôle directement par l’API de CARLA pour gérer les situations que l’on souhaitera implémenter.