Compte-rendu projet remplacement de RTMaps par ROS

LUCENAC Thomas

Contexte

Dans ce projet nous continuons un projet de mise en place de communication entre Carla, RTMaps et NS3, mais nous changeons de logiciel passant de RTMaps à ROS, étant un gestionnaire de nœuds permettant les communications entre ceux-ci, ce projet reprend donc la même version de Carla (0.9.11), python natif et une machine linux Ubuntu 20.04.6 LTS (focal).

Installation de ROS

Après avoir installé une machine linux ainsi que l'application Carla, nous pouvons passer à l'installation de ROS, dans notre cas nous allons installer ROS noetic :

Documentation d'installation de ROS noetic pouvant être trouvée sur le lien suivant :

http://wiki.ros.org/noetic/Installation/Ubuntu

Installation du bridge Carla ROS

Une fois ROS noetic installé nous allons pouvoir mettre en place les communications avec les différentes applications. Tout d'abord nous allons voir la communication avec Carla, pour cela il nous faut installer le bridge ROS-Carla grâce à la documentation Carla :

https://carla.readthedocs.io/projects/ros-bridge/en/latest/ros_installation_ros1/

Nous allons ensuite aller dans le fichier où se trouve notre installation Carla afin d'exécuter la commande :

./CarlaUE4.sh

Il faut que Carla soit lancé afin de pouvoir lancer le bridge Carla ROS.

Test de fonctionnement Carla ROS

Pour effectuer des tests de fonctionnement de ROS et permettre la création d'objet nous allons utiliser dans **chaque terminal** où nous allons lancer une commande ROS relier à la création d'objet Carla avec les commandes suivantes :

export CARLA_ROOT=/[Path_vers_install_Carla]/CARLA_0.9.11

export PYTHONPATH=$PYTHONPATH:$CARLA_ROOT/PythonAPI/carla/dist/carla-0.9.11-py3.7-linux-x86_64.egg:$CARLA_ROOT/PythonAPI/carla

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

Dans un autre terminal nous allons lancer la commande roscore.

Pour tester si ros est bien en fonctionnement et que le bridge et bien mis en place on peut exécuter la commande suivante dans un nouveau terminal :

roslaunch carla_ros_bridge carla_ros_bridge.launch

Celle-ci changera la carte de la town1 à la town3.

Afin de créer un véhicule nous pouvons utiliser l'exemple indiqué dans la documentation fournit par Carla :

https://github.com/carla-simulator/ros-bridge/blob/master/carla_spawn_objects/config/objects.json

Puis l'éxecuter via la commmande:

roslaunch carla_spawn_objects carla_spawn_objects.launch

objects_definition_file:=/[path_vers_objet]/objects.json

Puis nous pouvons voir la liste des noeuds actif via la commande : rosnode list

Ensuite nous pourrons récupérer la liste des publishers et subscribers de chaque nœuds avec la commande rosnode info /[NOM_DU_NODE]

Depuis cette liste nous pouvons lancer la voiture se trouvant par défaut dans la documentation en entrant la commande suivante :

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

rostopic : permet d'accéder au topics
pub : permet de publier une information
path : correspond au topic auquel nous voulons nous adresser
std_msgs/Bool : type de message que nous voulons transmettre
" data: true " : permet d'envoyé l'information true au booléen

Lors de communication il faut faire attention au type que le publisher/subscriber envoie/reçoit.

Nous pouvons le récupérer en inspectant le type via la commande :

rostopic info /[Path_de_notre_topic]

Création de l'arborescence pour les fichiers Python

Nous allons ensuite créer un fichier python permettant de communiquer avec l'un des publishers du noeud que nous souhaitons.

Tout d'abord afin de pouvoir les exécuter nous allons devoir créer un package catkin, il existe 2 façons de les créer, grâce à la commande catkin_make, l'autre grâce à la commande catkin build, dans notre cas nous allons utiliser catkin build avec les commandes suivantes :

mkdir [Path_vers_nos_subscribers]/catkin_ws/src

cd [Path_vers_nos_subscribers]/catkin_ws

catkin build

cd [Path_vers_nos_subscribers]/catkin_ws/src

Dans nos fichiers python actuel nous utilisions rospy, std_msgs ainsi que sensor_msgs pour créer le package.

catkin_create_pkg sub_pkg rospy std_msgs sensor_msgs

cd [Path_vers_nos_subscribers]/catkin_ws/src/sub_pkg

catkin build

source devel/setup.bash

Nous obtenons l'arborescence suivante :

Path_vers_nos_subscribers

└── catkin_ws

  └── src

    └── sub_pkg

      └── src

A noter : Chaque devel est unique au terminal où il est lancer, si nous lancions un autre terminal sans référencer le devel se trouvant dans notre dossier catkin_ws alors nous ne pourrons pas lancer le package et son contenu. Il faut aussi refaire le catkin_build à chaque fois que nous modifions ou créons/ajoutons un fichier python dans notre sub_pkg/src afin qu'il soit pris en compte dans notre terminal.

Une fois toute cette arborescence créée, nous devons mettre nos fichiers python dans le denier répertoire src/ .
Une fois les fichier Python mis en place nous allons faire la commande :

rosrun sub_pkg [Nom de notre fichier python].py

Ce qui permettre de lancer notre fichier Python permettant la communication.

En cas de besoin, une vidéo explicative des commandes pouvant être utile à la résolution de problème se trouve sur youtube via le lien suivant :

https://www.youtube.com/watch?v=otGWUZqB9XE

Mise en place du bridge entre ROS et NS3

Nous allons ensuite mettre en place la communication entre ROS et NS3 :

Lien vers la documentation d'installation :

https://github.com/malintha/rosns3

Lors de mon projet j'ai essayé de faire une communication avec le serveur NS3 en utilisant les publishers proposer par la documentation mais n'ai pas réussi à avoir de résultat satisfaisant. Pour récupérer les données, nous pouvons placer un fichier de la même façon que pour la communication entre Carla et les fichiers python en créant un fichier python.

Ressources

Les fichiers Pythons sont fournis joint avec cette documentation,