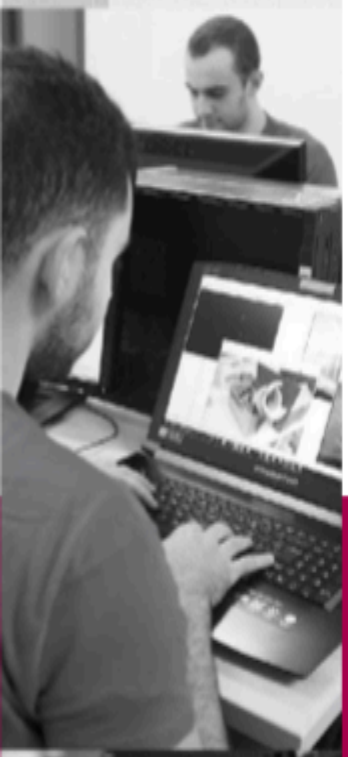

Document de travail

ENSEA

Beyond Engineering

Robotique

ROS2 - Écrire ses propres nodes

Compte rendu de travaux pratiques

3ème année MSC

FLANDIN Elio
DUGARD Kevin

COMPTE RENDU TP ROBOTIQUE

0 - Sommaire

I - Objectif

L'objectif de cette séance de travaux pratiques est de se familiariser avec la mise en œuvre de différents nœuds ROS2 pour le contrôle d'un robot simulé, à travers l'exemple de la tortue du package *turtlesim*. Dans un premier temps, nous établirons la communication entre un joystick physique et le simulateur en lançant successivement les nœuds *joy* (acquisition des commandes joystick), *teleop_twist_joy* (conversion des commandes en messages de type *Twist*) et *turtlesim* (environnement de simulation). Cette étape permettra de vérifier la bonne interconnexion des topics et de paramétrer les nœuds afin de piloter la tortue en temps réel à l'aide du joystick.

Dans un second temps, nous mettrons en œuvre l'outil *ros2 bag* pour enregistrer et rejouer les messages issus du joystick. Cette procédure vise à capturer une séquence de commandes représentative, permettant de tester les développements ultérieurs sans dépendre du périphérique d'entrée. L'analyse et la réutilisation de ces enregistrements faciliteront le débogage et la validation des comportements du robot.

Enfin, nous concevrons et implémenterons un nœud ROS personnalisé, appelé *chenille_node*, destiné à simuler une commande de type "char d'assaut". Ce nœud devra interpréter les entrées des deux sticks du joystick pour générer des messages *Twist* adaptés, offrant ainsi un contrôle différentiel de la tortue, similaire à un véhicule à chenilles. Ce développement mettra en pratique les principes de souscription, de publication et de gestion des messages dans ROS2, tout en illustrant la modularité et la flexibilité de l'architecture distribuée du système.

II - Dispositif de base

a- Explication et Configuration

Dans cette première partie, nous allons nous concentrer sur le joystick que nous allons ajouter.

```
ensea@Ros2Ensea2024:~$ lsusb
Bus 001 Device 001: ID 1d6b:0002 Linux Foundation 2.0 root hub
Bus 001 Device 002: ID 046d:c21d Logitech, Inc. F310 Gamepad [XInput Mode]
Bus 002 Device 001: ID 1d6b:0003 Linux Foundation 3.0 root hub
```

Figure 1 - Commande lsusb

Ainsi on voit bien ici que notre manette est reconnue, elle apparait en deuxième ligne. Maintenant ceci effectué, nous devons démarre tous les nodes un à un. On commence avec le node ***joy*** (celui du joystick) qui fait office de traducteur entre la manette et ROS. Puis on enchaine avec ***turtlesim*** afin de voir notre tortue que l'on va essayer de controler. Et enfin le node ***teleop_twist*** permettant de convertir les mouvements des joysticks en commandes de robot. Revenons brièvement sur ces nodes afin de mieux les comprendre.

- **joy** nous envoie des messages contenant deux groupes d'informations :
  - axes qui renvoie la position des joysticks (valeur entre -1 et 1).
  - button qui renvoie l'état des boutons (valeur égale à 0 ou 1).

- **teleop_twist** qui écoute ce que dit **joy** et publie des commandes de vitesse (aussi appelé Twist) vers **/cmd_vel** qui envoie deux informations différentes : l'axe linéaire (pour le déplacement avant ou arrière) et l'angle pour permettre au robot de tourner.

- **turtlesim** reçoit la commande envoyer par **teleop_twist** et fait tourner la tortue et nous l'affiche.

Ainsi, pour résumer voici un résumé de la chaine complète :

Joystick → /joy → /teleop_twist → /cmd_vel → /turtlesim

On peut d'ailleurs montrer cette chaine grâce à la commande `rqt`, qui nous proposera un graphe de cette dernière. Nous avons donc décidé de l'exécuter afin de vérifier que toutes liaisons de cette dite chaine était correctement faites.

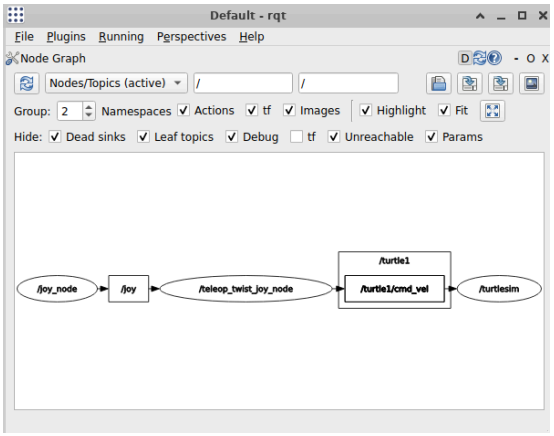

Figure 3 - Commande rqt_graph

b - Compréhension du joystick

Maintenant on connait ce qu'il faut sur les nodes que nous allons utiliser et nous savons que notre joystick est bien reconnu. Cependant, nous ne savons pas encore exactement comment il fonctionne. Pour cela, nous avons installer le paquet joystick (`sudo apt install joystick`) permettant d'utiliser la commande jstest nous en disant plus sur ce dernier. Cette commande permet d'écouter et d'afficher en direct les messages du joystick. Voici ce que nous obtenons une fois la commande exécutée.

```
ensea@Ros2Ensea2024:~$ jstest /dev/input/js0
Driver version is 2.1.0.
Joystick (Logitech Gamepad F310) has 8 axes (X, Y, Z, Rx, Ry, Rz, Hat0X, Hat0Y)
and 11 buttons (BtnA, BtnB, BtnX, BtnY, BtnTL, BtnTR, BtnSelect, BtnStart, BtnMode, BtnThumbL, BtnThumbR).
Testing ... (interrupt to exit)
Axes:  0: -2333  1:-32767  2: 23995  3:     0  4:-32767  5:-32767  6:     0  7:     0 Buttons:  0:off  1:off  2:off  3:off  4:off  5:on   6:off  7:off  8:off  9:off 10:off
```

Figure 2 - Commande jstest

Ici en regardant seulement la dernière ligne, on confirme donc bien ce que l'on a dit précédemment : le joystick envoie deux groupes de messages, un pour les joystick (*Axes*) et un pour les boutons (*Buttons*). On aperçoit ici des numéros, chacun de ces derniers est relié à une commande de la manette. En effet, il y en a un pour le mouvement vertical de chacun des joysticks, un pour celui horizontal, un pour les gâchettes arrière, et pour chaque boutons (les deux gâchettes avant, A, B, X, Y, la croix directionnelle et si l'on presse sur les deux joysticks) nous avons aussi un autre numéro.

Une autre méthode pour voir ce que le joystick nous renvoie était d'écouter ce dernier grâce à la commande `ros2 topic echo /joy`. Voici ce que l'on obtenais en direct (cette fois-ci les numéros ne sont plus indiqués).

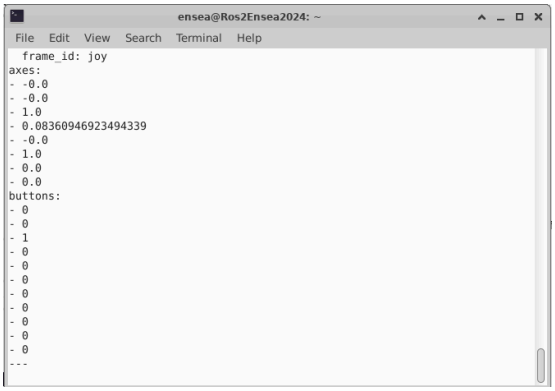

Figure 4 - Commande ros2 topic echo /joy

c - Controle de la tortue

Désormais, sans rien avoir changer, nous pouvions controler la tortue. Cependant, la configuration des touches étaient particulière : il fallait appuyer sur RB pour avancer, RT pour reculer et LB pour pivoter vers la droite (soit les numéros 5 et 2 pour le numérique et le 5 pour l'analogique).

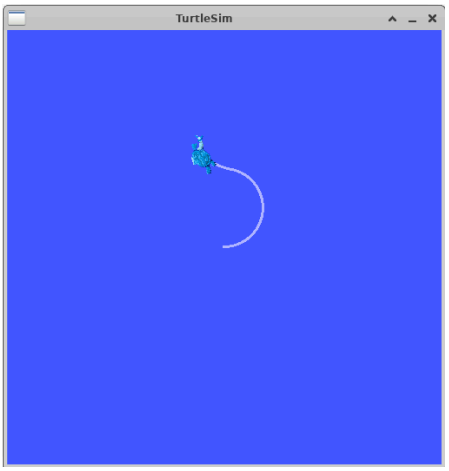

Figure 5 - Controle de la tortue

Ainsi, nous nous devions de changer ça pour obtenir quelque chose de plus naturelle : les joystick. Nous n'utiliserons que le potentiel vertical de celui de gauche pour pouvoir avancer ou reculer et le potentiel horizontal pour celui de droite afin de pivoter dans le sens horaire ou anti-horaire.

Et c'est à partir de là que nous avons décidé de partir sur un fichier launch en python afin de nous faciliter la tâche et de ne pas avoir une multitude de console de commande ouverte en train chacune d'exécuter un node. Nous précisons que pour cela, nous avons été fouiller dans les dossier de la machine virtuelle et pris un fichier que nous avons modifier (ajouter une ligne que l'on décrira ci-dessous) et l'activation du node turtle. Ici ne prenons que le morceau du code nous permettant justement de controler notre tortue avec les joysticks, sans avoir à appuyer sur RB pour avancer.

```
parameters=[{
                'enable_button': 5,
                'require_enable_button': False,    # Pas besoin d'activer
                'axis_linear.x': 1,                # stick gauche vertical
                'axis_angular.yaw': 3,             # stick droit horizontal
                'scale_linear.x': 1.0,
                'scale_angular.yaw': 1.5,
                'publish_stamped_twist': publish_stamped_twist
            }],
```

Figure 6 - Extrait de teleop_launch.py concernant le node teleop_twist

Ici la ligne `'require_enable_button': False` permet de ne pas avoir à appuyer sur un bouton pour activer le système, si l'on ne la rajoutais pas, il aurait fallu continuer d'appuyer sur RB puis d'avancer avec le joystick pour que notre tortue se déplace. Et désormais, nous pouvons commander notre turtle avec les joysticks comme on le souhaité.

Nous avions aussi la possibilité de rajouter sur un bouton un boost, mais ici nous ne l'avons pas fait car nous n'avons pas jugé cela utile.

III - Rosbag Record & Rosbag play

Passons désormais à une étape très intéressante : rosbag. Grâce à Rosbag, nous avons la possibilité d'enregistrer une suite de commande et de la rejouer autant de fois que l'on appliquera la commande nécessaire. Pour cela, nous devons dans un premier temps exécuter `ros2 bag record /joy` afin d'enregistrer nos suite de commande, effectuer cette dernière et enfin la couper avec un ctrl+C. Cette dernière est donc désormais enregistré dans le workspace défini au préalable (on doit trouver son nom en utilisant la commande ls d'ailleurs) et pouvons obtenir des informations dessus en exécutant la commande `ros2 bag info {nom du bag}`.

Une fois les informations obtenus, nous n'avons plus qu'à exécuter `ros2 bag play {nom du bag}` et le tour est joué, l'enchainement de commande enregistré est procédé par notre tortue.

```
ensea@Ros2Ensea2024:~/Documents/bag_files_turtle$ ros2 bag record /joy
[INFO] [1761315777.676839108] [rosbag2_recorder]: Press SPACE for pausing/resuming
[INFO] [1761315777.690403570] [rosbag2_recorder]: Listening for topics...
[INFO] [1761315777.690886440] [rosbag2_recorder]: Recording...
[INFO] [1761315777.690406863] [rosbag2_recorder]: Event publisher thread: Starting
[INFO] [1761315779.140897755] [rosbag2_recorder]: Subscribed to topic '/joy'
[INFO] [1761315779.141070981] [rosbag2_recorder]: All requested topics are subscribed. Stopping discovery...
[INFO] [1761315787.905492312] [rosbag2_recorder]: Pausing recording.
[INFO] [1761315789.007746533] [rosbag2_cpp]: Writing remaining messages from cache to the bag. It may take a while
[INFO] [1761315789.012578883] [rosbag2_recorder]: Event publisher thread: Exiting
[INFO] [1761315789.013249012] [rosbag2_recorder]: Recording stopped
[INFO] [1761315789.106300790] [rclcpp]: signal_handler(signum=2)
ensea@Ros2Ensea2024:~/Documents/bag_files_turtle$ ls
rosbag2_2025_10_24-16_22_57
ensea@Ros2Ensea2024:~/Documents/bag_files_turtle$ ros2 bag info rosbag2_2025_10_24-16_22_57

Files:             rosbag2_2025_10_24-16_22_57_0.mcap
Bag size:          39.8 KiB
Storage id:        mcap
ROS Distro:        jazzy
Duration:          8.690338302s
Start:             Oct 24 2025 16:22:59.188918111 (1761315779.188918111)
End:               Oct 24 2025 16:23:07.879256413 (1761315787.879256413)
Messages:          225
Topic information: Topic: /joy | Type: sensor_msgs/msg/Joy | Count: 225 | Serialization Format: cdr
Service:           0
Service information:
ensea@Ros2Ensea2024:~/Documents/bag_files_turtle$ ros2 bag play rosbag2_2025_10_24-16_22_57
[INFO] [1761315818.492919566] [rosbag2_player]: Set rate to 1
[INFO] [1761315818.564243967] [rosbag2_player]: Adding keyboard callbacks.
[INFO] [1761315818.564306979] [rosbag2_player]: Press SPACE for Pause/Resume
[INFO] [1761315818.564699512] [rosbag2_player]: Press CURSOR_RIGHT for Play Next Message
[INFO] [1761315818.564865551] [rosbag2_player]: Press CURSOR_UP for Increase Rate 10%
[INFO] [1761315818.564884927] [rosbag2_player]: Press CURSOR_DOWN for Decrease Rate 10%
[INFO] [1761315818.565706631] [rosbag2_player]: Playback until timestamp: -1
```

Figure 7 - Différentes étapes du ros bag

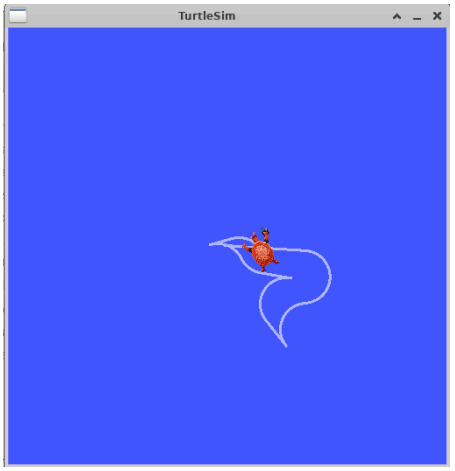

Figure 8 - Suite de commande enregistrée éxécutée par notre turtle

IV - Tortue à chenille

Nous arrivons à la dernière partie, celle où nous allons exploiter davantage les capacités de la turtle. En effet, nous avons vu qu'elle est capable d'avancer, de reculer et de tourner sur elle même dans les deux sens, comme si cette dernière était pourvue de deux chenilles (comme pourrait l'être un char d'assaut par exemple). Ainsi l'objectif de cette partie est de controler notre tortue comme si elle en était disposée, c'est à dire que le joystick gauche gère la chenille de gauche et celui de droite celle de droite (on ne s'interesse donc plus qu'à la position verticale de nos deux joysticks).

Pour cela, nous avons du créer un package chenille, le build et enfin écrire un nouveau launch en python. Pour ce qui concerne le package il s'agit juste d'un enchainement de commande à effectuer. Pour le code en Python, nous avons juste joué avec deux équations : une sur l'axe d'avancée et une sur la rotation.

```
# Avance/recul : moyenne des sticks
twist.linear.x = (left_stick + right_stick) / 2.0 * self.scale_linear
# Rotation : différence des sticks
twist.angular.z = (left_stick - right_stick) / 2.0 * self.scale_angular
```

Figure 9 - Équations pour obtenir deux chenilles

Et nous avons ajouter une feature que l'on retrouve fréquemment dans le monde de la robotique : celle de l'arrêt d'urgence. Pour cela, nous utilisons le bouton LB : dès que ce dernier est pressé, il arrête la tortue si elle était en déplacement, l'empêche de se déplacer si elle était à l'arrêt et affiche un message dans la console de commande (ça pourrait aussi faire clignoter une LED ou quelque chose semblable).

```
# Arrêt d'urgence sur bouton LB
if joy_msg.buttons[4] == 1:
    twist.linear.x = 0.0
    twist.angular.z = 0.0
    self.get_logger().info("Arrêt d'urgence !")
```

Figure 10 - Code pour l'arrêt d'urgence

Et c'est donc ici que nous mettons fin à cette séance de travaux pratiques en pouvant déplacer notre tortue comme si elle était sur deux chenilles. Cette séance nous a permis de découvrir plus en profondeur le monde vaste du ROS et donc de la commande robot.