

GE5  
2023

---

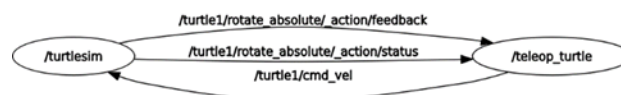
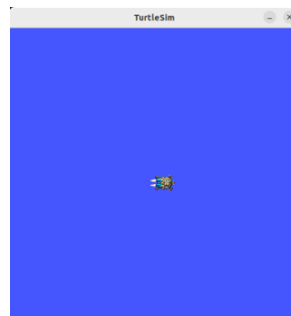
# Tutoriel ROS2

## Partie 1

---

*Auteurs :*

Arthur BOUILLÉ ([arthur.bouille@insa-strasbourg.fr](mailto:arthur.bouille@insa-strasbourg.fr))  
Alexandre THOUVENIN ([alexandre.thouvenin@insa-strasbourg.fr](mailto:alexandre.thouvenin@insa-strasbourg.fr))



# Table des matières

|          |   |           |
|----------|---|-----------|
| <b>1</b> | <b>Introduction</b>   | <b>3</b>  |
| <b>2</b> | <b>Nœuds, Topics, Services et Actions</b>   | <b>4</b>  |
| <b>3</b> | <b>Lancer Turtlesim et contrôler la tortue avec le teleopkey</b>                    | <b>6</b>  |
| 3.1      | Mise de la source dans le terminal . . . . .  | 6         |
| 3.2      | Lancer TurtleSim . . . . .  | 6         |
| 3.3      | Lancer la télécommande . . . . .  | 7         |
| 3.4      | Lancer TurtleSim et la télécommande dans le même terminal . . . . .                 | 8         |
| 3.5      | Mise de la source dans fichier <code>~/.bashrc</code> . . . . .                     | 8         |
| <b>4</b> | <b>Visualisation des nœuds et topics avec rqt graph</b>                             | <b>10</b> |
| 4.1      | Ouvrir rqt_graph . . . . .  | 10        |
| 4.2      | Lancer TurtleSim . . . . .  | 10        |
| 4.3      | Lancer la télécommande . . . . .  | 11        |
| <b>5</b> | <b>Comprendre les topics avec TurtleSim</b>   | <b>12</b> |
| 5.1      | Trouver la liste des "topics" actifs . . . . .                                      | 12        |
| 5.2      | Trouver les informations précises d'un "topic" . . . . .                            | 12        |
| 5.3      | Trouver les caractéristiques d'un "geometry_msgs/msg/Twist" . . . . .               | 12        |
| 5.4      | Envoyer un geometry_msgs/msg/Twist sur le topic /turtle1/cmd_vel . . . . .          | 13        |
| 5.5      | Envoyer un message en boucle le topic /turtle1/cmd_vel . . . . .                    | 13        |
| <b>6</b> | <b>Comprendre les topics avec TurtleSim et rqt</b>                                  | <b>15</b> |
| 6.1      | Ouvrir rqt . . . . .  | 15        |
| 6.2      | Lancer TurtleSim . . . . .  | 15        |
| 6.3      | Envoyer un geometry_msgs/msg/Twist sur le topic /turtle1/cmd_vel avec rqt . . . . . | 16        |
| 6.4      | Envoyer un geometry_msgs/msg/Twist en boucle avec rqt . . . . .                     | 17        |
| <b>7</b> | <b>Comprendre les actions avec TurtleSim</b>  | <b>18</b> |
| 7.1      | Découvrir les actions . . . . .   | 18        |
| 7.2      | Trouver la liste des "Action Servers" et "Action Clients" d'un noeud . . . . .      | 19        |
| 7.3      | Trouver la liste des actions disponibles et leurs informations . . . . .            | 20        |
| 7.4      | Visualisation de l'action /turtle1/rotate_absolute avec rqt_graph . . . . .         | 20        |
| 7.5      | Trouver la structure du type d'action "turtlesim/action/RotateAbsolute" . . . . .   | 21        |
| 7.6      | Envoyer un objectif à une action en ligne de commande . . . . .                     | 21        |
| <b>8</b> | <b>Comprendre les services avec TurtleSim</b>                                       | <b>23</b> |
| 8.1      | Découvrir les services . . . . .  | 23        |
| 8.2      | Trouver le type d'un service . . . . .  | 24        |
| 8.3      | Trouver tous les services actifs du même type . . . . .                             | 24        |
| 8.4      | Trouver la structure du type de service . . . . .                                   | 24        |
| 8.5      | Appeler un service . . . . .  | 25        |
| 8.6      | Faire spawn une nouvelle tortue dans TurtleSim . . . . .                            | 26        |
| <b>9</b> | <b>Bibliographie</b>  | <b>27</b> |

# 1 Introduction

Ce premier tutoriel a pour but d'apprendre à se servir de ROS et de ses commandes de base à l'aide de l'outil turtlesim. Dans cette optique, nous allons dans un premier temps nous intéresser à ce que sont ROS2 et turtlesim.

- **ROS2** : ROS est un middleware ( un software qui se comporte comme une passerelle entre des applications et des bases de données ou des systèmes d'exploitation, aussi appelé intergiciel en français ). ROS2 est une nouvelle version de ROS, comportant à peu près les mêmes composants, mais dont l'architecture change fondamentalement. ROS2, tout comme ROS, possède une architecture basée sur le principe de systèmes en temps réel distribués/répartis. Cela signifie que l'architecture est composée de nœuds qui se chargent d'effectuer des calculs ou des opérations simples, reliés entre eux par un réseau en temps réel.

La différence entre les deux vient du protocole réseau utilisé. ROS2 utilise un protocole réseau DDS (Data Distribution Service), qui a pour but de simplifier la programmation réseau. Chaque nœud créé avec ce protocole est à la fois un Publisher et un Subscriber ( voir la partie suivante pour plus de détails sur ce que sont les Publishers et Subscribers ), et est automatiquement créé sur une fraction du sous-réseau auquel ROS2 a accès. Cette fraction est appelée domaine, et tous les nœuds d'un domaine sont connectés entre eux lors de leurs créations. Cela rend à la fois la connexion à un réseau ROS2 plus facile, et plus sécuritaire, car la simple création d'un nœud sur le réseau permet d'avoir accès à l'ensemble du réseau, mais il faut connaître à l'avance le sous-réseau ainsi que le domaine sur lequel le réseau est situé.

- **Turtlesim** : Turtlesim est un outil d'apprentissage créé pour ROS et ROS2 afin d'appréhender plus facilement leurs différents concepts. Cet outil se compose d'un espace sur lequel plusieurs tortues peuvent évoluer, et différents topics, services et actions ( voir la partie suivante pour plus de détails sur ce que sont les topics, services et actions ) sont accessibles afin de faire bouger ces tortues et récupérer les données disponibles. Bien que pouvant paraître simple, turtlesim permet d'avoir un premier contact avec l'ensemble des concepts de ROS et ROS2, sans risque d'abimer un système réel.

## 2 Nœuds, Topics, Services et Actions

Vous trouverez quelques définitions pour mieux comprendre le fonctionnement de ROS2 (quasiment identique à ROS) :

- **Nœuds** : Un nœud est l'unité fondamentale de ROS2, et dont la tâche consiste la plupart du temps à réaliser une action spécifique. Ils peuvent être créés de diverses manières, mais leur fonctionnement reste similaire. Il en existe deux catégories principales : les nœuds « Publisher » et les nœuds « Subscriber ». De ces deux catégories est née une troisième catégorie : les nœuds « Composed », qui sont composées d'un « Publisher » et d'un « Subscriber ».

Le rôle d'un « Publisher » est de fournir des informations à au moins un « Subscriber », et celui-ci choisit quelles informations doivent lui être envoyées avant d'attendre de les recevoir.

Un « Subscriber » peut avoir souscrit à plusieurs « Publisher », et un « Publisher » peut fournir la même information à plusieurs « Subscriber ».

Les trois points suivants sont des interfaces permettant aux nœuds de communiquer entre eux. Chacun de ces types d'interface diffère des autres sur certains points. De plus, ils ont chacun leur domaine d'application.

- **Topics** : Les topics sont des canaux de communication unidirectionnels qui relient entre eux les différents nœuds. Ils agissent comme une sorte de relayeur d'informations. Ils reçoivent chacun les informations d'un seul nœud Publisher et peuvent transmettre à plusieurs Subscriber. Ils permettent ainsi à plusieurs entités d'avoir accès à un même topic, mais l'information ne se transmet que dans un seul sens.

Un topic est donc dit asynchrone. Le « Publisher » ne sait pas exactement qui sont les « Subscriber » qui vont récupérer les informations transmises à ce topic. Les topics sont utilisées pour les flots de données continus, tel que des données de capteurs (position/vitesse/distance...).

- **Services** : Les services sont des terminaux qui relient deux nœuds directement entre eux, et dont les informations peuvent être transmises dans les deux sens selon le système Request/Reply. Les services sont dit "**synchrones**".

Ainsi, le nœud ayant effectué la demande, attend la réponse de la part du second nœud. Les services sont donc utilisés pour des procédures qui se terminent rapidement, comme des calculs rapides ou la récupération de l'état d'un nœud. L'utilisation d'un service pour une procédure longue risque de bloquer les deux nœuds pour le temps de la procédure, et de générer des problèmes dans tous les autres nœuds qui pourraient être connectés à ceux-ci. Les services sont donc généralement utilisés pour récupérer rapidement des données spécifiques.

- **Actions** : Les actions ressemblent beaucoup à des services dans le sens où les actions utilisent également le système Request/Reply. La différence est qu'une action possède deux terminaux assimilés aux nœuds : un client d'action (qui formule la requête) et un serveur d'action (qui réalise la tâche). Trois services et deux topics transitent entre ces deux nœuds :

- *Les services* : **Send goal, Cancel goal, Get result**. Ils transitent du client d'action vers le serveur d'action.

- *Les topics* : **Feedback** et *Status*. Ils transitent pour leur part dans le sens inverse aux services, c'est-à-dire du serveur d'action vers le client d'action.

L'action va donc consister en demandes d'objectifs par le nœud formulant la requête. Ces demandes seront transmises au serveur d'action. Celui-ci va alors envoyer de façon récurrente la progression de l'action ainsi que son statut au client d'action. Le client d'action peut alors choisir de demander l'annulation d'une action en cours ou bien de récupérer le statut final d'une action. Plusieurs actions peuvent être demandées à un même serveur d'action car chaque action possède son propre identifiant. Les actions servent donc à pallier le problème des services : ils seront utilisés pour des processus long mais nécessitant des retours sur sa progression de façon régulière. Elles sont donc

principalement utilisées pour des actions réelles, tel que la mise en mouvement de robot.

- **Différence ROS et ROS2** : Avec ROS, les nœuds établissent des communications entre eux en passant par un serveur (un ROS Master : terminal où on écrit la commande : `$ roscore`). Dans ROS2 il n'y a plus de ROS master. La méthode de communication des nœuds a été modifiée afin que ces communications puissent se créer directement de nœud à nœud sans passer par un serveur (utilisation de la méthode *peer-to-peer* directement). ROS2 est donc un système à part de ROS, qui utilise un protocole de réseau DDS (Data Distribution Service) qui permet une sécurité et une fiabilité qui n'étaient pas le point fort des protocoles TCP/IP (Transmission Control Protocol/Internet Protocol) et UDP/IP (User Datagram Protocol) modifiés utilisés par ROS et couplés au ROS master.

En effet, le protocole DDS est bien plus sécuritaire que les protocoles UDP/IP ou TCP/IP, et l'absence de ROS master fait que l'ensemble du système ne repose plus sur le fonctionnement d'un seul élément. Il est cependant possible de lier des nœuds ROS2 à des nœuds ROS grâce à un pont qui permettent de recréer un système existant déjà sur ROS plus facilement. Ce système est pour le moment particulièrement utile car certains packages n'existent pas encore pour ROS2. Il suffit donc de créer un nœud ROS accédant au package et de le lier à un nœud ROS2 pour régler ce problème.

## 3 Lancer Turtlesim et contrôler la tortue avec le teleopkey

### 3.1 Mise de la source dans le terminal

1. Ouvrez un terminal avec "**ctrl+alt+T**"
2. Indiquez à ce terminal le chemin des fichiers de configuration de ROS2. Cela vous permettra de pouvoir utiliser des fonctions de ROS2. Voici la commande à taper dans le terminal :

```
insa@ROS2:~$ source /opt/ros/humble/setup.bash
```

Après avoir taper cette commande dans votre terminal, vous pourrez donc avoir accès à toutes les fonctions primaires de ROS2. Une autre méthode permet d'avoir automatiquement tous les terminaux sourcés. Cette seconde méthode sera vu dans la section [3.5](#)

### 3.2 Lancer TurtleSim

1. Pour lancer TurtleSim, nous devons activer le nœud de TurtleSim. Pour ce faire nous devons utiliser une structure de commande particulière. Voici la structure à utiliser :

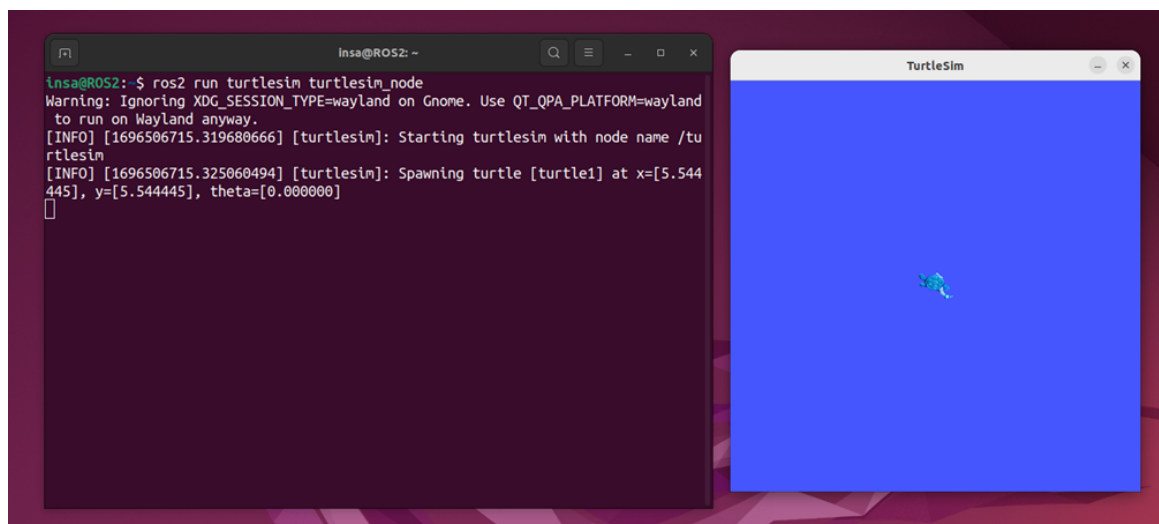
```
$ ros2 run <package_name> <executable_name>
```

Dans notre cas `<package_name>` sera l'application TurtleSim qui se nommera « turtlesim » et `<executable_name>` sera l'exécutable du nœud de TurtleSim qui sera « turtlesim\_node ».

2. Voici la commande à taper dans le terminal :

```
insa@ROS2:~$ ros2 run turtlesim turtlesim_node
```

3. Une fenêtre « TurtleSim » va s'ouvrir :



4. Plusieurs informations apparaissent dans le terminal :

- TurtleSim a été démarré avec le nœud « /turtlesim »
- Une tortue a été créée avec le nom « turtle1 »
- Et elle a été placée dans la fenêtre TurtleSim avec les coordonnées x,y et  $\theta$  suivantes :  $x = 5.54$  ;  $y = 5.54$  et une orientation  $\theta = 0.00^\circ$

### 3.3 Lancer la télécommande

Dans cette partie, nous allons voir comment lancer une télécommande qui nous permettra de faire bouger la tortue dans la fenêtre TurtleSim.

1. Ouvrez un second terminal et tapez la commande pour ajouter la source à ce terminal.
2. Pour lancer la télécommande, nous devons lancer le nœud de la télécommande. Pour ce faire nous devons utiliser la structure de commande précédente.  
Voici la structure :

```
$ ros2 run <package_name> <executable_name>
```

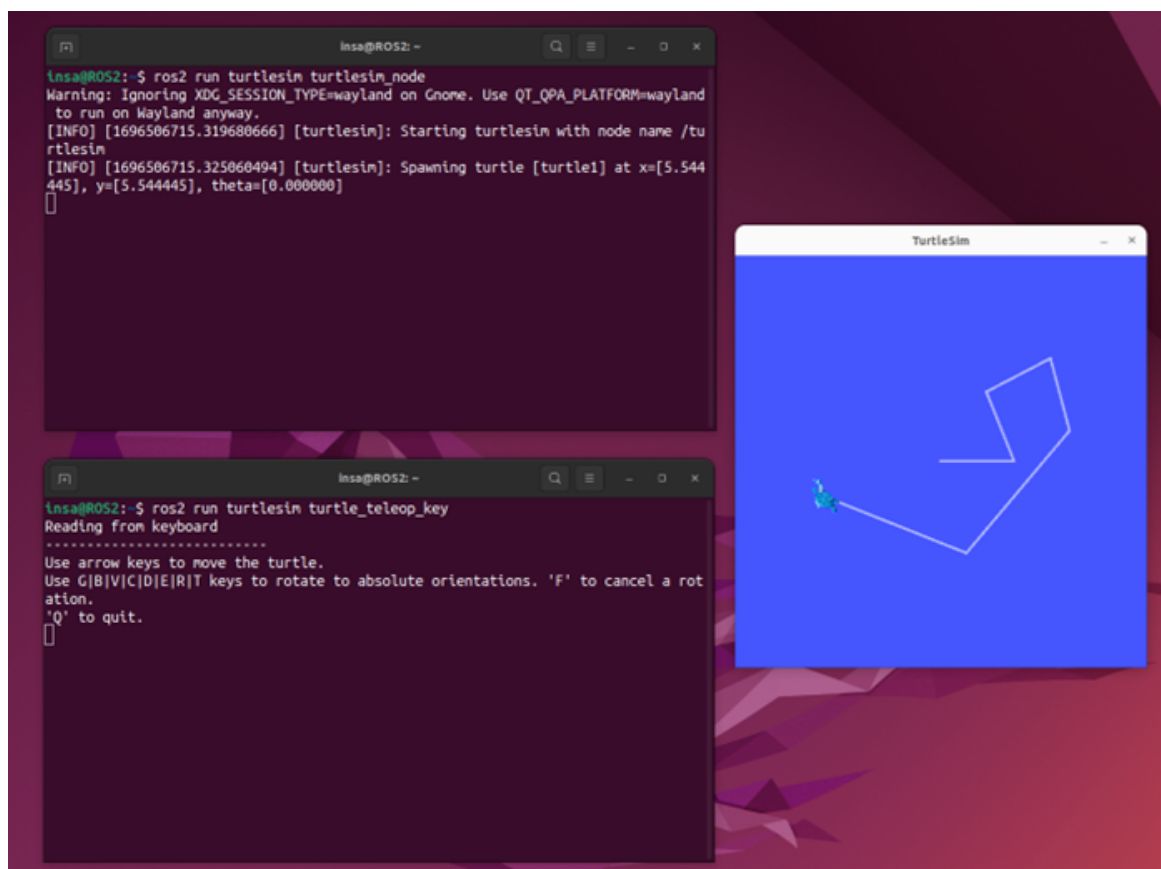
Dans notre cas <package\_name> est TurtleSim qui se nommera « turtlesim » et <executable\_name> est l'exécutible du nœud de la télécommande qui sera « turtle\_teleop\_key ».

3. Voici, finalement, la commande à taper dans le 2nd terminal :

```
insa@ROS2:~$ ros2 run turtlesim turtle_teleop_key
```

Cela va démarrer la télécommande qui vous permettra de contrôler les mouvements de la tortue dans l'environnement TurtleSim en utilisant les touches du clavier

4. Vous allez donc pouvoir contrôler la tortue avec les 4 flèches de votre clavier :



Nous verrons dans un des prochains tutoriels à quoi correspondent les rotations vers les orientations absolue avec les touches G,B,V,C,D,E,R et T.

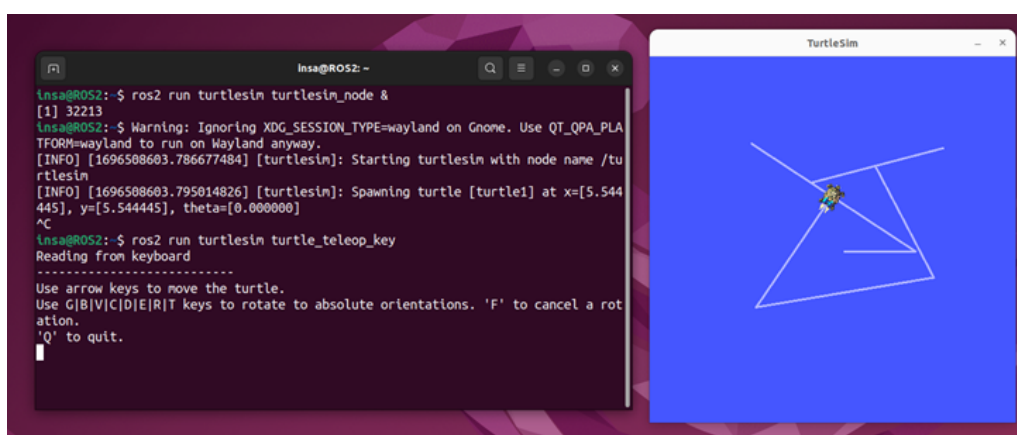
### 3.4 Lancer TurtleSim et la télécommande dans le même terminal

1. Arrêtez la commande de de la télécommande Turtlesim dans le second terminal et aussi la commande de TurtleSim dans le premier terminal, en faisant un "*ctrl+C*" dans chaque terminal.
2. Fermez un des deux terminaux.
3. Sur le terminal qui vous reste, tapez la commande pour lancer le nœud de TurtleSim avec un « & » à la fin . Le « & » a pour but de faire fonctionner TurtleSim en arrière-plan.  
Voici la commande :

```
insa@ROS2:~$ ros2 run turtlesim turtlesim_node &
```

Faites « Entrer » et vous constaterez que vous pouvez de nouveau utiliser ce Terminal.

4. Taper la commande pour avoir la télécommande teleop\_key :



Vous avez réussi à faire bouger votre tortue dans TurtleSim à l'aide de la télécommande en ayant lancé le noeuds de TurtleSim en arrière plan. La prochaine étape concerne la mise automatique de la source de RO2 pour chaque nouveau terminal ouvert.

### 3.5 Mise de la source dans fichier `~/.bashrc`

Cette partie consiste à ce que la source de ROS2 soit ajouté automatiquement pour chaque nouveau terminal que vous ouvrez. Pour ce faire suivez les instructions suivantes :

1. Ouvrez un terminal avec "*ctrl+alt+T*", puis modifier le fichier ".bashrc" avec la commande suivante puis tapez votre mot de passe (*lorsque vous rentrez votre mot de passe celui-ci ne s'affiche pas*) :

```
insa@ROS2:~$ sudo nano .bashrc
[sudo] Mot de passe de insa :
```

Explication de la commande :

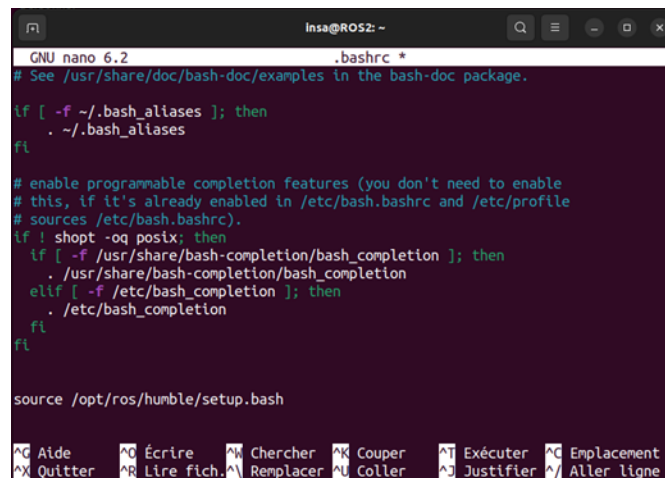
**sudo** : nous l'utilisons pour lancer une commande dans un terminal en mode administrateur.  
**nano** : c'est un éditeur de fichier et permet de les modifier.  
**.bashrc** : c'est un fichier de configuration de notre machine virtuelle.

Donc la commande **sudo nano .bashrc** permet d'ouvrir le fichier **.bashrc** dans l'éditeur de fichier **nano** le tout en **mode administrateur**.

2. Pour pouvoir utiliser TurtleSim, vous devez indiquer à le chemin du fichier setup de ROS2. Voici la ligne de code que vous avez à mettre à la fin de ce fichier : **source /opt/ros/humble/setup.bash**



Ajoutez le chemin du fichier setup de ROS2 à la fin de ce fichier et vous devriez obtenir :



```
GNU nano 6.2 .bashrc *
# See /usr/share/doc/bash-doc/examples in the bash-doc package.

if [ -f ~/.bash_aliases ]; then
  . ~/.bash_aliases
fi

# enable programmable completion features (you don't need to enable
# this, if it's already enabled in /etc/bash.bashrc and /etc/profile
# sources /etc/bash.bashrc).
if ! shopt -oq posix; then
  if [ -f /usr/share/bash-completion/bash_completion ]; then
    . /usr/share/bash-completion/bash_completion
  elif [ -f /etc/bash_completion ]; then
    . /etc/bash_completion
  fi
fi

source /opt/ros/humble/setup.bash
```

Sauvegardez et quittez le fichier avec « *Ctrl + O* », « *Entrer* » et ensuite « *Ctrl + X* ».

L'ajout de cette ligne de code permet au système de connaître automatiquement le chemin du fichier setup de ROS2. C'est-à-dire, que pour chaque nouveau Terminal que vous ouvrez, le fichier setup sera sourcé.

Le 1er tutoriel est fini. Passons au suivant pour contrôler la tortue en publiant des messages directement sur le topic « *turtle1/cmd\_vel* ».

## 4 Visualisation des nœuds et topics avec rqt\_graph

### 4.1 Ouvrir rqt\_graph

1. Vérifiez que la source des fichiers de setup de ROS2 est bien dans la fichier `.bashrc`. Cette étape a été vue à la section 3.5.
2. Ouvrez un terminal avec "`ctrl+alt+T`"
3. Lancez `rqt_graph` en tapant la commande suivante et une fenêtre `rqt_graph` va s'ouvrir :

```
insa@ROS2:~$ rqt_graph
```



- Dans la fenêtre `rqt_graph`, cliquez sur « Nodes only » (A côté des deux flèches bleues).
- Choisissez ensuite « Nodes/Topics (all)».
- Décocher ensuite « Dead sinks » et « Leaf topics » de la partie « Hide ».

`rqt_graph` permet de voir l'interaction entre les différents nœuds et topics actifs en ce moment. Notre `rqt_graph` est vide. *Est-ce normal ?*

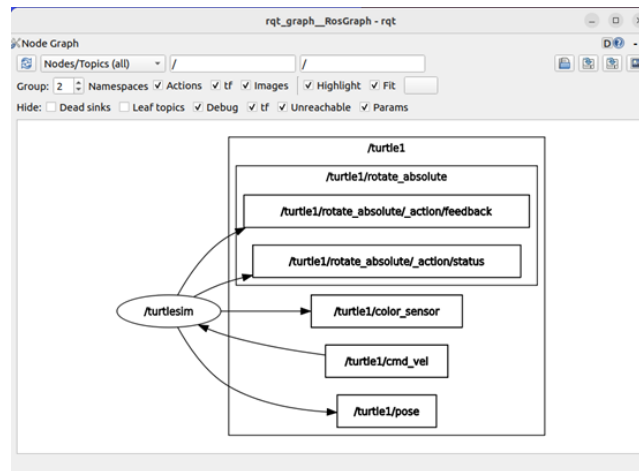
Oui, nous avons pour le moment aucun nœud actif dans les terminaux.

### 4.2 Lancer TurtleSim

1. Ouvrez un second Terminal (et vérifiez la source 3.5).
2. Tapez la commande qui permet de lancer TurtleSim en arrière plan :

```
insa@ROS2:~$ ros2 run turtlesim turtlesim_node &
```

3. Appuyez ensuite sur le bouton « Rafraîchir » de la fenêtre `rqt_graph` (bouton avec 2 flèches bleues sur la gauche).  
Vous obtiendrez ceci :



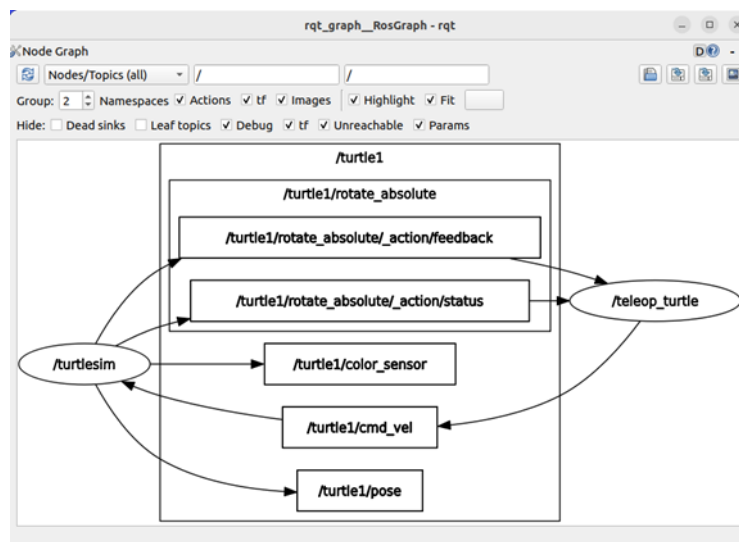
Vous pouvez donc voir le nœud turtlesim que vous venez de lancer avec la commande ci-dessus 4.2.2. Vous pouvez voir aussi tous les topics, liés à la tortue « /turtle1 », qui peuvent interagir avec le nœud. Vous retrouverez le topic « /turtle1/cmd\_vel » qui sera utilisé dans la suite de ce tutoriel

### 4.3 Lancer la télécommande

1. Lancez, toujours dans le second terminal, la télécommande pour la tortue avec la commande suivante :

```
ins@ROS2:~$ ros2 run turtlesim turtle_teleop_key
```

2. Retournez sur la fenêtre rqt\_graph et rafraîchissez le graph. Vous allez obtenir ceci :



Vous pouvez donc voir que le nœud de la télécommande est apparu lui aussi sous le nom /teleop\_turtle. On peut voir que la télécommande ne peut agir que sur le topic /turtle1/cmd\_vel.

## 5 Comprendre les topics avec TurtleSim

### 5.1 Trouver la liste des "topics" actifs

1. Lancez TurtleSim dans un nouveau terminal
2. Dans un second terminal, visualisez les topics actifs en tapant la commande :

```
lnsa@ROS2:~$ ros2 topic list
/parameter_events
/rosout
/turtle1/cmd_vel
/turtle1/color_sensor
/turtle1/pose
```

Vous constaterez que vous avez dans la liste des topics actifs, les topics sur lesquels TurtleSim publie ses données. Nous avons déjà vu ces topics dans la section précédente. Le topic qui nous intéresse est le topic `/turtle1/cmd_vel`

### 5.2 Trouver les informations précises d'un "topic"

1. Tapez la commande :

```
lnsa@ROS2:~$ ros2 topic info /turtle1/cmd_vel
Type: geometry_msgs/msg/Twist
Publisher count: 1
Subscription count: 1
```

2. Le topic `/turtle1/cmd_vel` accepte uniquement les messages de type `geometry_msgs/msg/Twist`. Nous verrons dans le prochain tutoriel à quoi correspondent les termes "Publisher" et "Subscription".

### 5.3 Trouver les caractéristiques d'un "geometry\_msgs/msg/Twist"

1. Tapez la commande suivante, pour trouver les caractéristique d'un message de type `geometry_msgs/msg/Twist` :

```
lnsa@ROS2:~$ ros2 interface show geometry_msgs/msg/Twist
# This expresses velocity in free space broken into its linear and angular parts.

Vector3  linear
  float64 x
  float64 y
  float64 z
Vector3  angular
  float64 x
  float64 y
  float64 z
```

2. Vous pouvez voir qu'un message de type `geometry_msgs/msg/Twist` contient deux vecteurs. Un pour indiquer les vitesses linéaires que l'on veut donner à la tortue et le second pour les vitesses angulaires.

La tortue évoluant dans un système 2D, il sera uniquement possible de lui envoyer des vitesses linéaires selon x et y et des vitesses angulaires selon z. L'axe x est correspond à la direction de la tortue, l'axe y à la vitesse latérale de la tortue et l'axe z correspond la vitesse angulaire de la tortue en sachant que z pointe vers vous.

## 5.4 Envoyer un geometry\_msgs/msg/Twist sur le topic /turtle1/cmd\_vel

1. Observez la structure de la commande permettant d'envoyer un message d'un certain type sûr un topic :

```
$ ros2 topic pub <pub_frequency> <topic_name> <message_type> <message>
```

2. Tapez la commande suivante, pour envoyer un geometry\_msgs/msg/Twist sur le topic /turtle1/cmd\_vel (en spécifiant en paramètre "- -once" pour n'envoyer qu'un seul message sur ce topic) :

```
lnsa@ROS2:~$ ros2 topic pub --once /turtle1/cmd_vel geometry_msgs/msg/Twist "{lin
ear: {x: 2.0, y: 0.0, z: 0.0}, angular: {x: 0.0, y: 0.0, z: 1.8}}"
```

Vous pourrez voir dans le terminal que le geometry\_msgs/msg/Twist a été publié /turtle1/cmd\_vel :

```
lnsa@ROS2:~$ ros2 topic pub --once /turtle1/cmd_vel geometry_msgs/msg/Twist "{lin
ear: {x: 2.0, y: 0.0, z: 0.0}, angular: {x: 0.0, y: 0.0, z: 1.8}}"
```

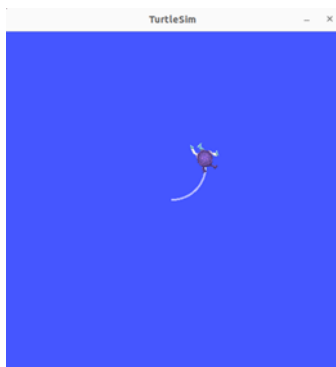
Waiting for at least 1 matching subscription(s)...

Waiting for at least 1 matching subscription(s)...

publisher: beginning loop

publishing #1: geometry\_msgs.msg.Twist(linear=geometry\_msgs.msg.Vector3(x=2.0, y=0.0, z=0.0), angular=geometry\_msgs.msg.Vector3(x=0.0, y=0.0, z=1.8))

3. On remarque que notre tortue « turtle1 » a avancée en tournant sur la gauche.



Ce qui correspond bien à une vitesse selon x et une vitesse angulaire selon z

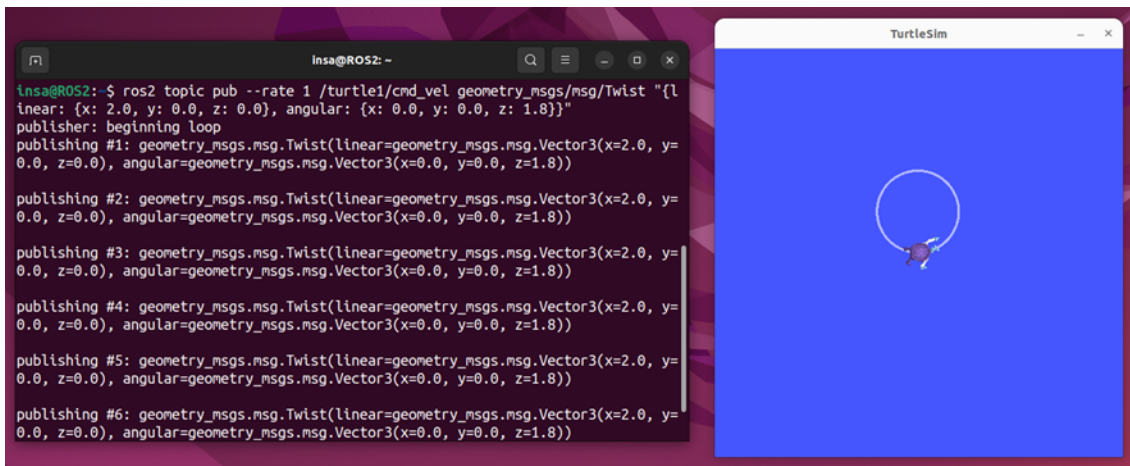
## 5.5 Envoyer un message en boucle le topic /turtle1/cmd\_vel

1. Pour envoyer en boucle le message précédent, il vous suffit de modifier le paramètre "- -once" par "- -rate <freq\_Hz>" ( <freq\_Hz> correspond à la fréquence d'envoi du message). Dans notre cas, nous allons écrire le paramètre « - -rate 1 » pour envoyer ce message successivement à une fréquence de 1Hz

Voici la commande :

```
lnsa@ROS2:~$ ros2 topic pub --rate 1 /turtle1/cmd_vel geometry_msgs/msg/Twist "{l
inear: {x: 2.0, y: 0.0, z: 0.0}, angular: {x: 0.0, y: 0.0, z: 1.8}}"
```

2. On remarque donc que le message est bien envoyé en boucle avec une fréquence de 1Hz et que la tortue tourne en boucle :



Vous pourrez voir les messages que vous avez publiés sur le topic `/turtle1/cmd_vel`, s'afficher toutes les secondes (*car on a mis un rate de 1Hz*)

Le 2ème tutoriel est fini. Passons au suivant pour visualiser les nœuds de TurtleSim en publiant des messages directement sur le topic « `turtle1/cmd_vel` » à l'aide de `rqt`.

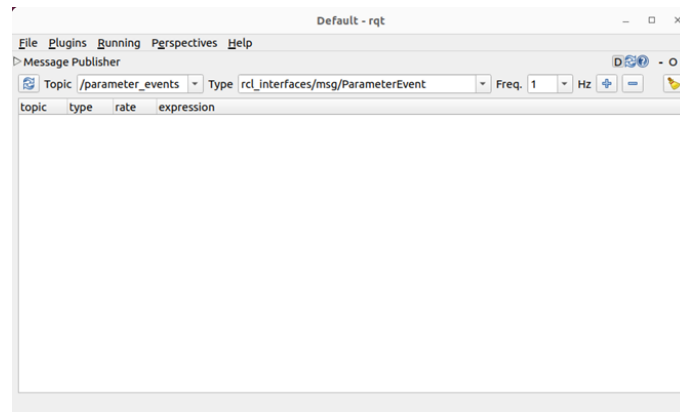
## 6 Comprendre les topics avec TurtleSim et rqt

### 6.1 Ouvrir rqt

1. Ouvrez un nouveau terminal et vérifiez la source de ROS2
2. Tapez la commande suivante pour ouvrir rqt :

```
insa@ROS2:~$ rqt
```

Une fenêtre "rqt" va s'ouvrir :



Si vous arrivez sur rqt mais pour autre chose que ">Message Publisher". Allez dans Plugins, puis Topics et cliquez sur Message Publisher.

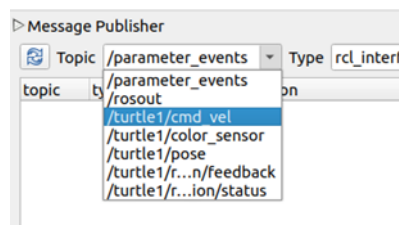
*Définition* : rqt est un outil de ROS permettant de visualiser, interagir et modifier la structure des nœuds, des topics et des services du système, en offrant une interface graphique pour le développement et le débogage de robots.

Dans notre cas, nous allons utiliser rqt pour envoyer des messages sur le topic `cmd_vel` d'une tortue dans TurtleSim.

3. Cliquez sur le menus déroulant des différents topics actifs. Vous verrez que les topics liés à la tortue de Turtlesim ne sont pas présents. C'est normal, vous n'avez pas lancé TurtleSim

### 6.2 Lancer TurtleSim

1. Ouvrez un nouveau terminal et lancez TurtleSim.
2. Cliquez sur les flèches bleues pour rafraîchir rqt et ensuite cliquez sur le menus déroulant des différents topics actifs. Les topics liés à TurtleSim sont apparus et sélectionnez le topic `/turtle1/cmd_vel` :

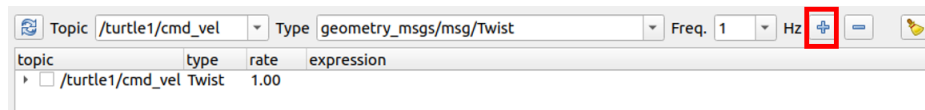


3. On remarque que rqt trouve directement le type de message à utiliser sur ce topic :



### 6.3 Envoyer un geometry\_msgs/msg/Twist sur le topic /turtle1/cmd\_vel avec rqt

1. Cliquez sur le "+" bleu pour créer un geometry\_msgs/msg/Twist :



2. Cliquez sur la petite flèche sur la gauche du message pour développer le message :

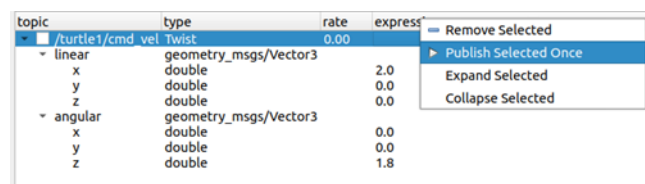
| topic              | type                  | rate | expression |
|--------------------|-----------------------|------|------------|
| ▾ /turtle1/cmd_vel | Twist                 | 1.00 |            |
| ▾ linear           | geometry_msgs/Vector3 |      |            |
| x                  | double                | 0.0  |            |
| y                  | double                | 0.0  |            |
| z                  | double                | 0.0  |            |
| ▾ angular          | geometry_msgs/Vector3 |      |            |
| x                  | double                | 0.0  |            |
| y                  | double                | 0.0  |            |
| z                  | double                | 0.0  |            |

3. Paramétrez le message que vous allez envoyer. Choisissez les mêmes valeurs que pour le [tutoriel 2](#) soit une vitesse linéaire x : 2.0, y : 0.0 , z : 0.0 et une vitesse angulaire x : 0.0, y : 0.0 , z : 1.8 :

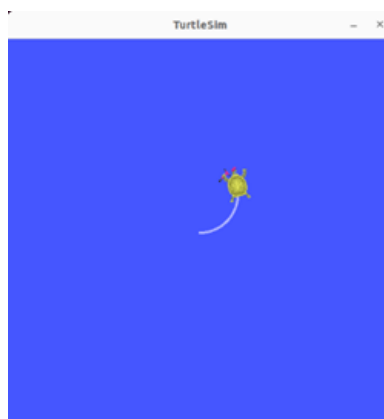
| topic              | type                  | rate | expression |
|--------------------|-----------------------|------|------------|
| ▾ /turtle1/cmd_vel | Twist                 | 0.00 |            |
| ▾ linear           | geometry_msgs/Vector3 |      |            |
| x                  | double                | 2.0  |            |
| y                  | double                | 0.0  |            |
| z                  | double                | 0.0  |            |
| ▾ angular          | geometry_msgs/Vector3 |      |            |
| x                  | double                | 0.0  |            |
| y                  | double                | 0.0  |            |
| z                  | double                | 1.8  |            |

La valeur du paramètre "rate" est effective uniquement lorsque la case en haut à gauche du message est cochée. Dans notre cas, nous voulons l'envoyer qu'une seule fois, donc laissez cette case décochée.

4. Faites un clic droit sur le message à envoyer, puis cliquez sur "Publish Selected Once" (permet d'envoyer qu'un seul message)

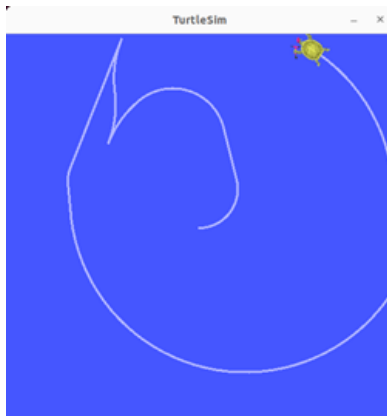


Vous pouvez alors voir votre tortue bouger dans la fenêtre TurtleSim :





5. Amusez-vous en changeant les paramètres du message et puis en l'envoyant. ***Cependant certains messages ne sont pas valides. Vous ne pouvez utiliser que les vitesses linéaires  $x$  et  $y$  (resp. déplacement avant/arrière et déplacement latéral droite/gauche) et une vitesse angulaire selon  $z$  (l'axe  $z$  pointant vers vous) car ces 3 paramètres déterminent la position de la tortue dans ce système 2D :***



#### 6.4 Envoyer un geometry\_msgs/msg/Twist en boucle avec rqt

1. Utilisez la fonctionnalité "rate" qui permet d'envoyer en boucle le même message. Modifiez les paramètres de votre message et aussi la valeur de "rate". La valeur de rate correspond à la fréquence d'envoi du message.
2. Cochez la case à gauche du message pour que le paramètre « rate » soit pris en compte :

| topic  | type                  | rate | expression |
|--|-----------------------|------|------------|
| <input checked="" type="checkbox"/> /turtle1/cmd_vel | Twist                 | 1.00 |            |
| <input type="checkbox"/> linear                      | geometry_msgs/Vector3 |      |            |
| x  | double                |      | 0.0        |
| y  | double                |      | 0.0        |
| z  | double                |      | 0.0        |
| <input type="checkbox"/> angular                     | geometry_msgs/Vector3 |      |            |
| x  | double                |      | 0.0        |
| y  | double                |      | 0.0        |
| z  | double                |      | 0.0        |

Vous verrez votre tortue bouger en boucle en fonction de votre message dès que vous aurez coché cette case. Décochez la case pour arrêter l'envoi cyclique de message.

Le 3ème tutoriel est fini. Vous comprenez donc maintenant le fonctionnement et les relations entre les « nœuds » et « topics ». Dans le prochain tutoriel vous allez utiliser et comprendre le fonctionnement des « actions » avec TurtleSim.

## 7 Comprendre les actions avec TurtleSim

### 7.1 Découvrir les actions

1. Faites "ctrl+C" dans tous les terminaux pour arrêter tous les processus en cours et fermez tous les terminaux ouverts.
2. Ouvrez un terminal et lancez TurtleSim en arrière plan (Voir la section 3.4.3).
3. Lancez, ensuite, la télécommande dans le même terminal. Comme déjà vu en 3.4.4 :

```
Use arrow keys to move the turtle.
Use G|B|V|C|D|E|R|T keys to rotate to absolute orientations. 'F' to cancel a rotation.
'Q' to quit.
```

La deuxième ligne correspond aux actions que l'on peut demander à la tortue de TurtleSim. (La première ligne correspond à l'utilisation des flèches pour piloter la tortue avec le topic "cmd\_vel", abordé précédemment dans les tutoriels).

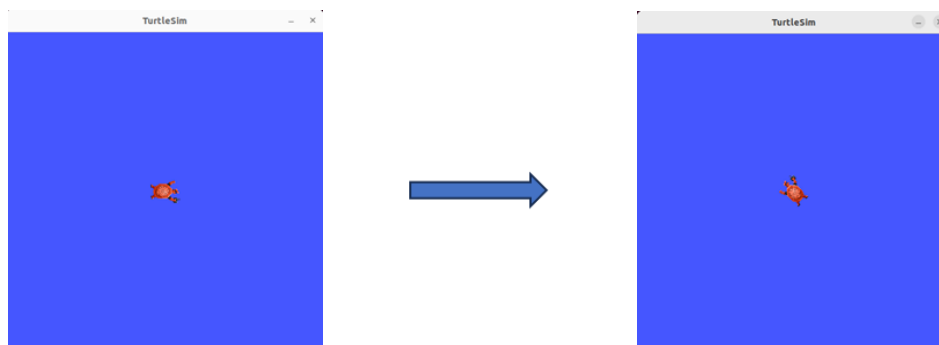
On remarque que les touches G|B|V|C|D|E|R|T forment un "carré" autour de la touche F sur votre clavier. La position de chaque touche autour de F correspond à cette orientation dans la fenêtre TurtleSim. Par exemple, le E fera pivoter l'orientation de la tortue vers le coin supérieur gauche.

Prêtez attention au terminal où le nœud /turtlesim est exécuté. Chaque fois que vous appuyerez sur l'une de ces touches, vous enverrez un objectif à un serveur d'action qui fait partie du nœud /turtlesim. L'objectif est de faire tourner la tortue dans une direction précise.

4. Appuyez sur la touche 'E'. Un message relayant le résultat de l'objectif devrait s'afficher une fois que la tortue a terminée sa rotation :

```
[INFO] [1696614408.831599474] [turtlesim]: Rotation goal completed successfully
```

Votre tortue à tournée vers le coin supérieur gauche :



La touche 'F', au centre du carré de touche, permet d'annuler l'objectif en cours.

5. Appuyer, par exemple sur la touche 'C', pour faire tourner la tortue vers une coin inférieur gauche, puis sur la touche 'F' avant que la tortue ne termine sa rotation. Dans le terminal où le nœud /turtlesim est exécuté, vous verrez le message suivant :

```
[INFO] [1696615742.129682601] [turtlesim]: Rotation goal canceled
```

Non seulement le côté client (votre entrée dans le teleop\_key ) peut arrêter un objectif, mais le côté serveur (le nœud /turtlesim) le peut également. Lorsque le serveur décide d'arrêter le traitement d'un objectif, on dit qu'il "abandonne" l'objectif.

6. Essayez d'appuyer sur la touche 'T', puis sur la touche 'G' avant que la première rotation ne soit terminée. Dans le terminal où le nœud /turtlesim est exécuté, vous verrez le message suivant :

```
[WARN] [1696616283.341494676] [turtlesim]: Rotation goal received before a previous goal finished. Aborting previous goal
[INFO] [1696616285.078650246] [turtlesim]: Rotation goal completed successfully
```

On remarque, sur la fenêtre TurtleSim, que la rotation vers 'T' (coin supérieur droit) a été annulée et a été remplacée par une rotation vers l'orientation de 'G' (droite).

## 7.2 Trouver la liste des "Action Servers" et "Action Clients" d'un nœud

1. Ouvrez un second terminal
2. Tapez la commande suivante :

```
insa@ROS2:~$ ros2 node info /turtlesim
```

Cette commande renvoie une liste des « Subscribers », des « Publishers », des « Services », des « serveurs d'actions » et des « clients d'actions » de /turtlesim :

```
insa@ROS2:~$ ros2 node info /turtlesim
/turtlesim
Subscribers:
  /parameter_events: rcl_interfaces/msg/ParameterEvent
  /turtle1/cmd_vel: geometry_msgs/msg/Twist
Publishers:
  /parameter_events: rcl_interfaces/msg/ParameterEvent
  /rosout: rcl_interfaces/msg/Log
  /turtle1/color_sensor: turtlesim/msg/Color
  /turtle1/pose: turtlesim/msg/Pose
Service Servers:
  /clear: std_srvs/srv/Empty
  /kill: turtlesim/srv/Kill
  /reset: std_srvs/srv/Empty
  /spawn: turtlesim/srv/Spawn
  /turtle1/set_pen: turtlesim/srv/SetPen
  /turtle1/teleport_absolute: turtlesim/srv/TeleportAbsolute
  /turtle1/teleport_relative: turtlesim/srv/TeleportRelative
  /turtlesim/describe_parameters: rcl_interfaces/srv/DescribeParameters
  /turtlesim/get_parameter_types: rcl_interfaces/srv/GetParameterTypes
  /turtlesim/get_parameters: rcl_interfaces/srv/GetParameters
  /turtlesim/list_parameters: rcl_interfaces/srv/ListParameters
  /turtlesim/set_parameters: rcl_interfaces/srv/SetParameters
  /turtlesim/set_parameters_atomically: rcl_interfaces/srv/SetParametersAtomically
Service Clients:
Action Servers:
  /turtle1/rotate_absolute: turtlesim/action/RotateAbsolute
Action Clients:
```

On remarque que le nœud /turtlesim est un serveur de l'action /turtle1/rotate\_absolute. C'est-à-dire qu'il peut recevoir des actions à effectuer et qu'il renvoie aussi des données au client.

3. Utilisez et modifiez la commande précédente pour trouver les informations du nœud /teleop\_turtle :

```
insa@ROS2:~$ ros2 node info /teleop_turtle
/teleop_turtle
Subscribers:
  /parameter_events: rcl_interfaces/msg/ParameterEvent
Publishers:
  /parameter_events: rcl_interfaces/msg/ParameterEvent
  /rosout: rcl_interfaces/msg/Log
  /turtle1/cmd_vel: geometry_msgs/msg/Twist
Service Servers:
  /teleop_turtle/describe_parameters: rcl_interfaces/srv/DescribeParameters
  /teleop_turtle/get_parameter_types: rcl_interfaces/srv/GetParameterTypes
  /teleop_turtle/get_parameters: rcl_interfaces/srv/GetParameters
  /teleop_turtle/list_parameters: rcl_interfaces/srv/ListParameters
  /teleop_turtle/set_parameters: rcl_interfaces/srv/SetParameters
  /teleop_turtle/set_parameters_atomically: rcl_interfaces/srv/SetParametersAtomically
Service Clients:
Action Servers:
Action Clients:
  /turtle1/rotate_absolute: turtlesim/action/RotateAbsolute
```

On remarque que le nœud /teleop\_turtle est un client de l'action /turtle1/rotate\_absolute. C'est-à-dire qu'il peut envoyer actions que le serveur doit effectuer et il peut recevoir des données de la part du serveur.

### 7.3 Trouver la liste des actions disponibles et leurs informations

1. Tapez la commande suivante dans le second terminal pour avoir la liste des actions disponibles :

```
insa@ROS2:~$ ros2 action list
```

Vous obtiendrez dans le terminal la liste des actions disponible :

```
insa@ROS2:~$ ros2 action list
/turtle1/rotate_absolute
```

Cependant, chaque action possède un type précis. Il est possible de l'afficher en ajoutant le paramètre « -t » dans la commande précédente.

2. Tapez la commande suivante pour obtenir la liste des actions ainsi que leur type directement :

```
insa@ROS2:~$ ros2 action list -t
/turtle1/rotate_absolute [turtlesim/action/RotateAbsolute]
```

Entre crochets à droite de chaque nom d'action (dans notre cas, uniquement /turtle1/rotate\_absolute) se trouve le type d'action, turtlesim/action/RotateAbsolute. Vous en aurez besoin lorsque vous voudrez exécuter une action à partir de la ligne de commande ou du code.

3. Tapez la commande suivante dans le second terminal pour Vous pouvez analyser l'action /turtle1/rotate\_absolute :

```
insa@ROS2:~$ ros2 action info /turtle1/rotate_absolute
```

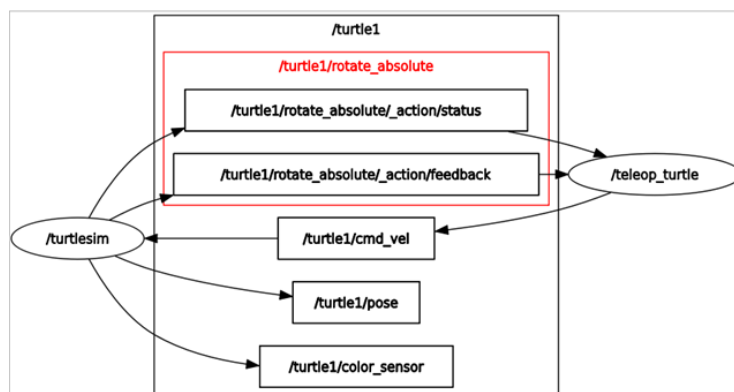
Vous obtiendrez dans le terminal les informations de cette action :

```
insa@ROS2:~$ ros2 action info /turtle1/rotate_absolute
Action: /turtle1/rotate_absolute
Action clients: 1
  /teleop_turtle
Action servers: 1
  /turtlesim
```

Cela nous indique ce que nous avons appris plus tôt en exécutant `ros2 node info` sur chaque nœud : Le nœud /teleop\_turtle est un client d'action et le nœud /turtlesim est un serveur d'action pour l'action /turtle1/rotate\_absolute.

### 7.4 Visualisation de l'action /turtle1/rotate\_absolute avec rqt\_graph

1. Tapez la commande pour ouvrir `rqt_graph`, toujours dans le second terminal (Voir : 4.1).
2. Faites les mêmes réglages que dans le 4.1. (Concernant les cases à cocher ou à décocher dans `rqt_graph`)
3. Comme TurtleSim et la télécommande `teleop_key` sont lancés, les nœuds /turtlesim et /teleop\_turtle le sont aussi. Vous obtiendrez :



Vous pouvez constater que l'action `/turtle1/rotate_absolute` peut recevoir des données du nœud `/turtlesim` (serveur de l'action) et que le nœud `/teleop_turtle` peut recevoir des données de cette action (client de l'action).

## 7.5 Trouver la structure du type d'action "turtlesim/action/RotateAbsolute"

Une autre information dont vous aurez besoin avant d'envoyer ou d'exécuter vous-même un objectif d'action est la structure du type de cette action. Rappelez-vous que vous avez identifié précédemment le type de `/turtle1/rotate_absolute` en exécutant la commande `$ ros2 action list -t` (type : `turtlesim/action/RotateAbsolute`).

1. Entrez la commande suivante pour déterminer la structure de ce type d'action :

```
insa@ROS2:~$ ros2 interface show turtlesim/action/RotateAbsolute
```

Vous obtiendrez dans le terminal :

```
insa@ROS2:~$ ros2 interface show turtlesim/action/RotateAbsolute
# The desired heading in radians
float32 theta
---
# The angular displacement in radians to the starting position
float32 delta
---
# The remaining rotation in radians
float32 remaining
```

2. Identifiez les différents paramètres :
  - La section du message au-dessus du premier `- - -` est la structure (type de données et nom) de la demande d'objectif.
  - La section suivante est la structure du résultat.
  - La dernière section est la structure du feedback.

On remarque que ces trois paramètres sont du type `float32` (soit des chiffres à virgule codés sur 32 bits).

## 7.6 Envoyer un objectif à une action en ligne de commande

1. Observez la structure d'une commande pour envoyer un objectif à une action :

```
$ ros2 action send_goal <action_name> <action_type> <values>
```

2. Gardez un oeil sur la fenêtre TurtleSim, et entrez la commande suivante dans votre terminal :

```
insa@ROS2:~$ ros2 action send_goal /turtle1/rotate_absolute turtlesim/action/RotateAbsolute "{theta: 1.57}"
```

Votre tortue, dans la fenêtre TurtleSim, va tourner. Et vous obtiendrez cette réponse dans le terminal :

```
insa@ROS2:~$ ros2 action send_goal /turtle1/rotate_absolute turtlesim/action/RotateAbsolute "{theta: 1.57}"
Waiting for an action server to become available...
Sending goal:
  theta: 1.57

Goal accepted with ID: d6257a29f46a4e728c974dc724e84cf8

Result:
  delta: -1.5520000457763672

Goal finished with status: SUCCEEDED
```

Tous les objectifs ont un identifiant unique, indiqué dans le message de retour. Vous pouvez également voir le résultat, un champ avec le nom `delta`, qui est le déplacement par rapport à la position

de départ.

3. Ajoutez le paramètre « -feedback » à la commande `ros2 action send_goal`, pour voir le feedback d'un objectif. Changez la valeur de `theta`, par exemple `theta = -1.57` et n'oubliez pas d'ajouter le paramètre « -feedback » :

```
lnsa@ROS2:~$ ros2 action send_goal /turtle1/rotate_absolute turtlesim/action/RotateAbsolute "{theta: -1.57}" --feedback
```

Votre terminal vous renvoie le message :

```
lnsa@ROS2:~$ ros2 action send_goal /turtle1/rotate_absolute turtlesim/action/RotateAbsolute "{theta: -1.57}" --feedback
Waiting for an action server to become available...
Sending goal:
  theta: -1.57

Feedback:
  remaining: -3.126814842224121

Goal accepted with ID: cc9e16751e9d440aabb5cb6f4407baf9

Feedback:
  remaining: -3.1108148097991943

Feedback:
  remaining: -3.0948147773742676

Feedback:
  remaining: -3.078814744949341
```

↓ Vous continuerez à recevoir les feedback (les degrés restants), jusqu'à ce que l'objectif soit atteint.

```
Feedback:
  remaining: -0.03881478309631348

Feedback:
  remaining: -0.02281475067138672

Feedback:
  remaining: -0.006814718246459961

Result:
  delta: 3.119999885559082

Goal finished with status: SUCCEEDED
```

Les actions sont comme des services qui vous permettent d'exécuter des tâches de longue durée. Ces actions fournissent un retour d'information régulier et peuvent être annulées.

Un système robotique utilisera probablement des actions pour la navigation. Un objectif d'action pourrait indiquer à un robot de se rendre à une position. Pendant que le robot navigue vers la position, il peut envoyer des mises à jour en cours de route (c'est-à-dire un retour d'information), puis un message de résultat final une fois qu'il a atteint sa destination.

Turtlesim dispose d'un serveur d'action auquel les clients d'action peuvent envoyer des objectifs pour faire bouger les tortues. Dans ce tutoriel, vous avez étudié cette action, `/turtle1/rotate_absolute`, pour avoir une meilleure idée de ce que sont les actions et comment elles fonctionnent.

## 8 Comprendre les services avec TurtleSim

### 8.1 Découvrir les services

1. Ouvrez un terminal et tapez la commande suivante pour avoir la liste des services actifs en ce moment :

```
insa@ROS2:~$ ros2 service list
insa@ROS2:~$
```

Le terminal ne vous renvoie rien. Est-ce normal ? Oui, car aucun noeud n'a encore été lancé !

2. Lancez TurtleSim en arrière plan dans un second terminal.
3. Tapez à nouveau la commande pour avoir la liste des services actifs :

```
insa@ROS2:~$ ros2 service list
/clear
/kill
/reset
/spawn
/turtle1/set_pen
/turtle1/teleport_absolute
/turtle1/teleport_relative
/turtlesim/describe_parameters
/turtlesim/get_parameter_types
/turtlesim/get_parameters
/turtlesim/list_parameters
/turtlesim/set_parameters
/turtlesim/set_parameters_atomically
```

Comme vous pouvez le voir, tous les services liés au noeud turtlesim et à la /turtle1 sont visibles, donc actifs.

4. Tapez la commande pour lancer la télécommande dans le second terminal et tapez à nouveau la commande pour avoir la liste des services actifs. Vous obtiendrez la liste de services actifs suivants :

```
insa@ROS2:~$ ros2 service list
/clear
/kill
/reset
/spawn
/teleop_turtle/describe_parameters
/teleop_turtle/get_parameter_types
/teleop_turtle/get_parameters
/teleop_turtle/list_parameters
/teleop_turtle/set_parameters
/teleop_turtle/set_parameters_atomically
/turtle1/set_pen
/turtle1/teleport_absolute
/turtle1/teleport_relative
/turtlesim/describe_parameters
/turtlesim/get_parameter_types
/turtlesim/get_parameters
/turtlesim/list_parameters
/turtlesim/set_parameters
/turtlesim/set_parameters_atomically
insa@ROS2:~$
```

Vous verrez que les deux nœuds ont six services identiques avec des paramètres dans leurs noms. Presque tous les nœuds de ROS 2 ont ces services d'infrastructure à partir desquels les paramètres sont construits.

Dans ce tutoriel, les services de paramètres seront omis de la discussion.

Pour l'instant, concentrons-nous sur les services spécifiques à turtlesim, /clear, /kill, /reset, /spawn, /turtle1/set\_pen, /turtle1/teleport\_absolute, et /turtle1/teleport\_relative.

## 8.2 Trouver le type d'un service

Les services ont des types qui décrivent comment les données de demande et de réponse d'un service sont structurées. Les types de service sont définis de la même manière que les types de topics, sauf que les types de service ont deux parties : un message pour la demande et un autre pour la réponse.

1. Observez la structure d'une commande pour trouver le type d'un service :

```
$ ros2 service type <service_name>
```

2. Utilisez cette commande pour le service /clear de turtlesim (Le service /clear permet d'effacer la trace que la tortue laisse sur la fenêtre de TurtleSim après avoir bougée) :

```
insa@ROS2:~$ ros2 service type /clear
```

Le terminal vous renvoie :

```
insa@ROS2:~$ ros2 service type /clear
std_srvs/srv/Empty
```

Le type *Empty* signifie que l'appel de service n'envoie aucune donnée lorsqu'il émet une requête et ne reçoit aucune donnée lorsqu'il reçoit une réponse.

3. Tapez la commande suivante, pour directement avoir la liste des services actifs et leur type respectif :

```
insa@ROS2:~$ ros2 service list -t
/clear [std_srvs/srv/Empty]
/kill [turtlesim/srv/Kill]
/reset [std_srvs/srv/Empty]
/spawn [turtlesim/srv/Spawn]
/teleop_turtle/describe_parameters [rcl_interfaces/srv/DescribeParameters]
/teleop_turtle/get_parameter_types [rcl_interfaces/srv/GetParameterTypes]
/teleop_turtle/get_parameters [rcl_interfaces/srv/GetParameters]
/teleop_turtle/list_parameters [rcl_interfaces/srv/ListParameters]
/teleop_turtle/set_parameters [rcl_interfaces/srv/SetParameters]
/teleop_turtle/set_parameters_atomically [rcl_interfaces/srv/SetParametersAtomically]
/turtle1/set_pen [turtlesim/srv/SetPen]
/turtle1/teleport_absolute [turtlesim/srv/TeleportAbsolute]
/turtle1/teleport_relative [turtlesim/srv/TeleportRelative]
/turtlesim/describe_parameters [rcl_interfaces/srv/DescribeParameters]
/turtlesim/get_parameter_types [rcl_interfaces/srv/GetParameterTypes]
/turtlesim/get_parameters [rcl_interfaces/srv/GetParameters]
/turtlesim/list_parameters [rcl_interfaces/srv/ListParameters]
/turtlesim/set_parameters [rcl_interfaces/srv/SetParameters]
/turtlesim/set_parameters_atomically [rcl_interfaces/srv/SetParametersAtomically]
```

## 8.3 Trouver tous les services actifs du même type

1. Observez la structure de la commande pour trouver les services d'un certain type :

```
$ ros2 service find <type_name>
```

2. Utilisez cette commande pour trouver tous les services actifs du type Empty :

```
insa@ROS2:~$ ros2 service find std_srvs/srv/Empty
```

Le terminal vous renvoie :

```
insa@ROS2:~$ ros2 service find std_srvs/srv/Empty
/clear
/reset
```

## 8.4 Trouver la structure du type de service

1. Observez la structure de la commande pour trouver la structure d'un type de service :

```
$ ros2 interface show <type_name>
```



- Utilisez cette commande pour trouver la structure du type de service Empty :

```
insa@ROS2: $ ros2 interface show std_msgs/msg/Empty
```

Le terminal ne vous renvoie rien et c'est normal car ce type de service est Empty. Le type *Empty* signifie que l'appel de service n'envoie aucune donnée lorsqu'il émet une requête et ne reçoit aucune donnée lorsqu'il reçoit une réponse.

- Trouvez le type du service /spawn. Vous trouverez :

```
turtlesim/srv/Spawn
```

- Trouvez ensuite la structure du type du service /spawn avec la commande :

```
insa@ROS2:~$ ros2 interface show turtlesim/srv/Spawn
float32 x
float32 y
float32 theta
string name # Optional. A unique name will be created and returned if this is empty
---
string name
```

Les informations au-dessus de la ligne - - - nous indiquent les arguments nécessaires pour appeler /spawn. x, y et theta déterminent la pose 2D de la tortue engendrée, et name est clairement optionnel.

Les informations sous la ligne ne sont pas nécessaires dans ce cas, mais elles peuvent vous aider à comprendre le type de données de la réponse que vous obtenez lors de l'appel du service /spawn.

## 8.5 Appeler un service

- Observez la structure de la commande pour appeler un service :

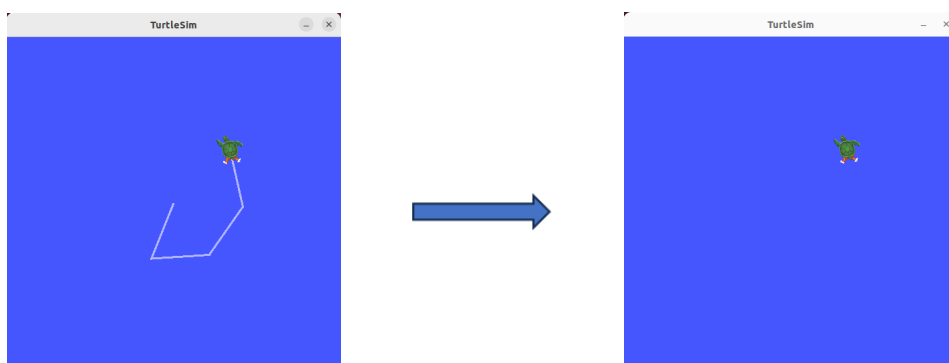
```
$ ros2 service call <service_name> <service_type> <arguments>
```

La partie <arguments> est facultative. Par exemple, vous savez que les services du type /Empty n'ont pas d'arguments.

- Utilisez cette structure de commande pour appeler le service /clear. Le service clear a pour but d'effacer le chemin parcouru par la tortue dans la fenêtre TurtleSim. Avant d'utiliser cette commande pour effacer le chemin parcouru par la tortue faites bouger la tortue grâce à la télécommande.
- Tapez la commande suivante pour supprimer le tracé de la tortue :

```
insa@ROS2:~$ ros2 service call /clear std_srvs/srv/Empty
requester: making request: std_srvs.srv.Empty_Request()
response:
std_srvs.srv.Empty_Response()
```

Vous pourrez voir le changement suivant dans la fenêtre TurtleSim :



## 8.6 Faire spawn une nouvelle tortue dans TurtleSim

1. Faites spawn une nouvelle tortue en appelant le service `/spawn` et en définissant des arguments. Les `<arguments>` d'entrée dans un appel de service à partir de la ligne de commande doivent être en syntaxe YAML.  
Voici la syntaxe :

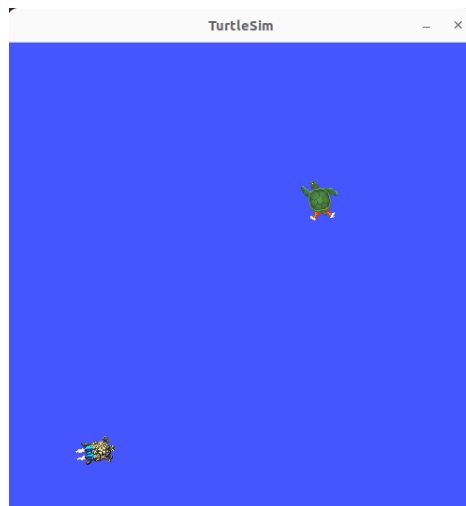
```
lnsa@R052:~$ ros2 service call /spawn turtlesim/srv/Spawn "{x: 2, y: 2, theta: 0.2, name: 'turtle_tutoriel'}"
```

Le terminal va vous renvoyer les informations suivantes :

```
lnsa@R052:~$ ros2 service call /spawn turtlesim/srv/Spawn "{x: 2, y: 2, theta: 0.2, name: 'turtle_tutoriel'}"
requester: making request: turtlesim.srv.Spawn_Request(x=2.0, y=2.0, theta=0.2, name='turtle_tutoriel')
response:
turtlesim.srv.Spawn_Response(name='turtle_tutoriel')
```

Ces informations montrent que nous avons appelé le service `spawn` avec les données `x,y` et `theta` et qu'en sortie nous avons bien fait spawn une nouvelle tortue nommée comme on le souhaitait.

2. Observez TurtleSim et vous verrez qu'une seconde tortue à apparu au coordonnées que vous avez écrit dans la commande :



Les nœuds peuvent communiquer en utilisant des services dans ROS 2. Contrairement à un topic - un modèle de communication à sens unique où un nœud publie des informations qui peuvent être consommées par un ou plusieurs subscribers - un service est un modèle de demande/réponse où un client fait une demande à un nœud fournissant le service et le service traite la demande et génère une réponse.

En général, vous ne souhaitez pas utiliser un service pour des appels continus ; des sujets ou même des actions conviendraient mieux.

Dans ce tutoriel, vous avez utilisé des outils de ligne de commande pour identifier, introspecter et appeler des services.

## 9 Bibliographie

- [ROS2 Documentation : TOPIC/SERVICES/ACTIONS](#),
- [ROS2 Design : ACTIONS](#)
- [Différences ROS1/ROS2](#),
- [ROS2 Documentation : Humble](#),
- Tutoriel ROS Noetic, Guillaume Hansen et Alexandre Thouvenin, 2023
- How to Control a Real TurtleBot with ROS through a Remote Raspberry Pi as ROS Master and with an OptiTrack Motion Capture System, Sylvain Durand, 2023
- [Robot Operating System](#), Olivier Stasse