

GE5
2023

Tutoriel ROS2

Partie 4

Auteurs :

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



Table des matières

1	Introduction	3
2	Connecter Matlab à un réseau ROS2	4
2.1	Vérifier le domaine ROS2 dans le fichier .bashrc	4
2.2	Ouvrir Matlab et créer un premier nœud sur le bon domaine	5
2.3	Lancer turtlesim et vérifier que Matlab se connecte bien au réseau ROS2	5
3	Utiliser le service /kill avec Matlab	7
3.1	Trouver le type de ce service sur Matlab	7
3.2	Lancer le client de ce service	8
3.3	Créer le message à envoyer au service	8
3.4	Envoyer le message au service	9
4	Utiliser le service /spawn avec Matlab	11
4.1	Trouver le type de ce service sur Matlab	11
4.2	Lancer le client de ce service	11
4.3	Créer le message à envoyer au service	12
4.4	Envoyer le message au service	13
5	Bibliographie	14

1 Introduction

Ce tutoriel vise à mettre en pratique les connaissances acquises lors du tutoriel précédent en intégrant ROS2 et turtlesim de manière cohérente avec Matlab. Pour ce faire, nous allons revoir les concepts de ROS2 et de turtlesim, ainsi que présenter Matlab et Simulink.

ROS2 : ROS (Robot Operating System) est un middleware qui agit comme une interface entre les applications et les bases de données ou les systèmes d'exploitation. ROS2, la nouvelle version de ROS, partage des composants similaires à sa version précédente, mais avec une architecture fondamentalement différente. Basé sur des systèmes distribués en temps réel, ROS2 consiste en des nœuds effectuant divers calculs ou opérations simples, connectés en réseau temps réel. Contrairement à ROS, ROS2 utilise le protocole réseau DDS (Data Distribution Service) pour simplifier la programmation réseau. Chaque nœud créé sous ce protocole, agit en tant que Publisher et Subscriber et est automatiquement créé sur un sous-réseau spécifique appelé domaine, permettant une connexion réseau plus simple et sécurisée.

Turtlesim : Turtlesim sert d'outil d'apprentissage pour ROS et ROS2, offrant une approche pratique des concepts associés. Cet outil simule un espace où plusieurs tortues peuvent se déplacer, permettant l'accès à différents topics, services et actions pour contrôler ces tortues et collecter les données disponibles. Bien que simple en apparence, turtlesim constitue une première introduction aux concepts de ROS et ROS2, sans risquer de perturber un système réel.

Matlab/Simulink : Matlab est une plateforme de calcul numérique et de programmation, offrant diverses bibliothèques appelées toolboxes pour une utilisation dans des domaines tels que l'ingénierie, les sciences et l'économie. Elle propose notamment une toolbox ROS qui sera utilisée dans ce tutoriel. D'autre part, Simulink est une plateforme de simulation et de modélisation de systèmes dynamiques, présentée comme un environnement de schémas en blocs, étroitement associée à Matlab pour simplifier la programmation des systèmes dynamiques à modéliser. Bien que non utilisé dans ce tutoriel (Simulink pas encore compatible à 100% avec ROS2), il sera abordé à la fin.

2 Connecter Matlab à un réseau ROS2

Avant de débiter ce guide, si vous rencontrez un problème avec l'une des commandes utilisées ici, il est probable que l'installation des prérequis pour l'utilisation de Matlab avec ROS2 n'ait pas été correctement effectuée. Dans ce cas, il vous suffit de consulter le tutoriel intitulé « Tuto installation des prérequis à l'utilisation de Matlab avec ROS2 ».

ROS2 opère via un protocole réseau appelé DDS (Data Distribution Service). Ce protocole crée des domaines distincts au sein d'un même sous-réseau, chacun étant identifié par un ID différent. Ces domaines ne sont pas interconnectés et ne peuvent pas échanger de données entre eux.

Contrairement à ROS, ROS2 ne possède pas de ROS Master ; les nœuds établissent automatiquement des connexions entre eux s'ils se trouvent dans le même sous-réseau et le même domaine. Afin de connecter Matlab à un réseau ROS2, il suffit de créer un nouveau nœud depuis Matlab dans le même sous-réseau et le même domaine, et la connexion s'établira automatiquement entre les deux.

2.1 Vérifier le domaine ROS2 dans le fichier .bashrc

1. Lancez un premier terminal dans la machine virtuelle ROS2 et modifiez le fichier .bashrc avec la commande suivante :

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

2. Vérifiez qu'il existe un domaine ROS2 de défini à la fin du fichier avec la ligne suivante :

```

GNU nano 6.2                                .bashrc
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
export ROS_DOMAIN_ID=0
  
```

3.
 - a. Si ce n'est pas le cas, ajoutez cette ligne en choisissant un ID qui n'est pas encore utilisé par une autre personne sur le même sous-réseau.
 - b. Si cette ligne existe déjà, retenez l'ID qui y est écrit.
4. Vous pouvez alors quitter le fichier .bashrc avec la commande ctrl+X et enregistrer les changements si vous avez du ajouter la définition du domaine.

2.2 Ouvrir Matlab et créer un premier nœud sur le bon domaine

1. Ouvrez alors Matlab.
2. Observez la structure de la commande « `setenv` ». cette commande permet de définir le domaine sur lequel sera connecté Matlab. Voici la structure de la commande :

`setenv(varname,varvalue)`

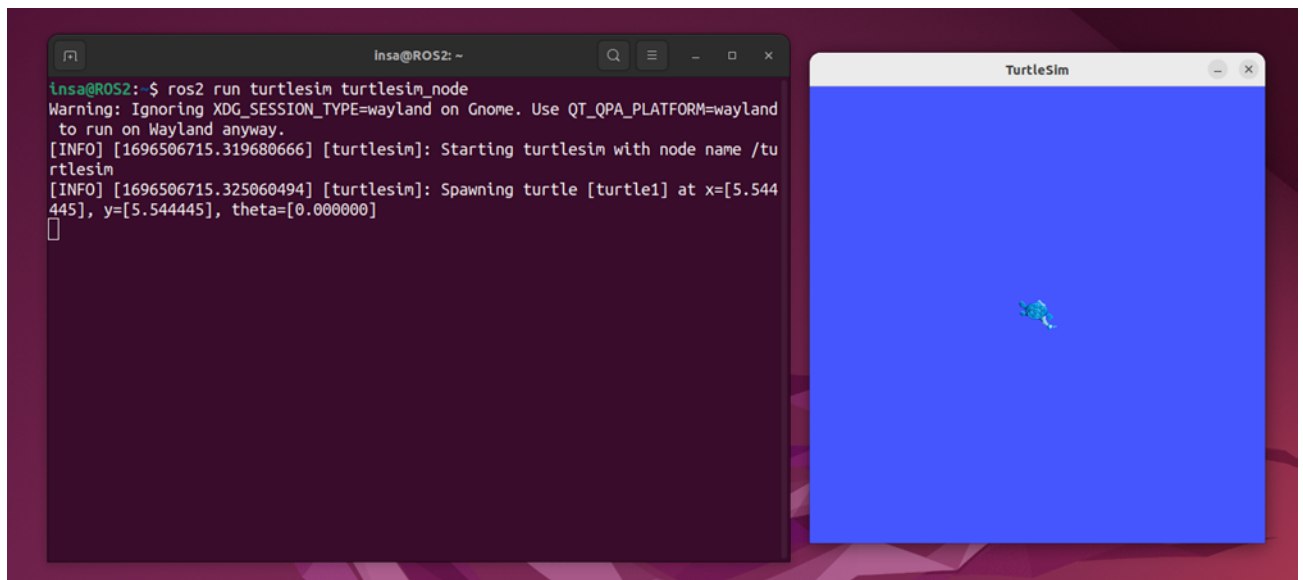
3. Utilisez cette commande en choisissant le même domaine que la VM ROS2 :

```
>> setenv("ROS_DOMAIN_ID","0")
fx >> |
```

S'il n'y a pas d'erreur, Matlab doit maintenant avoir accès au bon domaine ROS2 et il nous reste plus qu'à tester tout ceci.

2.3 Lancer turtlesim et vérifier que Matlab se connecte bien au réseau ROS2

1. Commencez par lancer turtlesim dans un terminal avec la commande suivante :



2. Observez la structure de la commande « `ros2node` ». cette commande permet de créer un nœud grâce à Matlab, ce qui nous permettra de nous connecter au réseau ROS2 et de le découvrir. Voici la structure de la commande :

`node = ros2node(Name)`

3. Créez alors un nœud `"/default_Node"` comme suit :

```
>> defaultNode = ros2node("/default_Node")

defaultNode =

ros2node with properties:

    Name: '/default_Node'
    ID: 0

fx >> |
```

4. Vérifiez alors que l'ID correspond bien à l'ID du domaine qui a été défini plus haut.
5. Visualisez les noeuds ROS2 depuis Matlab pour vérifier que Matlab soit bien connecté au réseau ROS2. On peut alors dresser une liste des nœuds ROS2 depuis Matlab, et vérifier que turtlesim s'y trouve bien. On utilise alors la commande suivante :

```
>> defaultNode = ros2node("/default_Node")

defaultNode =

    ros2node with properties:

        Name: '/default_Node'
        ID: 0

fx >> |
```

On observe bien que turtlesim apparait dans la liste des nœuds. **Matlab est donc bien connecté au réseau ROS2.**

3 Utiliser le service /kill avec Matlab

Dans la présente section, nous allons examiner les services et étudier leur utilisation avec Matlab. Nous débuterons par répertorier les services accessibles sur turtlesim à partir de Matlab.

3.1 Trouver le type de ce service sur Matlab

1. Utilisez la commande suivante, pour voir la liste des services actifs :

```
>> ros2 service list
/_matlab_introspec__0_rmw_fastrtps_cpp/describe_parameters
/_matlab_introspec__0_rmw_fastrtps_cpp/get_parameter_types
/_matlab_introspec__0_rmw_fastrtps_cpp/get_parameters
/_matlab_introspec__0_rmw_fastrtps_cpp/list_parameters
/_matlab_introspec__0_rmw_fastrtps_cpp/set_parameters
/_matlab_introspec__0_rmw_fastrtps_cpp/set_parameters_atomically
/clear
/kill
/reset
/spawn
/turtle1/rotate_absolute/_action/cancel_goal
/turtle1/rotate_absolute/_action/get_result
/turtle1/rotate_absolute/_action/send_goal
/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
```

On observe alors que les 6 premiers services sont pour Matlab, et que les autres sont ceux que nous avons déjà vu dans les précédents tutoriels. Ce sont donc ces derniers qui vont nous intéresser.

2. Pour faire disparaître la tortue, il faut utiliser le service /kill. Pour se connecter à un service ROS2, il faut créer un client de service. Or, pour créer un client de service, on doit d'abord connaître le type du service que nous voulons utiliser.
3. Tapez la commande suivante dans Matlab :

```
>> ros2 msg list
```

4. Vérifiez que les messages suivants sont disponibles :

```
turtlesim/Color
turtlesim/KillRequest
turtlesim/KillResponse
turtlesim/Pose
turtlesim/RotateAbsoluteFeedback
turtlesim/RotateAbsoluteGoal
turtlesim/RotateAbsoluteResult
turtlesim/SetPenRequest
turtlesim/SetPenResponse
turtlesim/SpawnRequest
turtlesim/SpawnResponse
turtlesim/TeleportAbsoluteRequest
turtlesim/TeleportAbsoluteResponse
turtlesim/TeleportRelativeRequest
turtlesim/TeleportRelativeResponse
```

- a. Si ce n'est pas le cas, reprenez le tutoriel « Tuto installation des prérequis à l'utilisation de Matlab avec ROS2 ».
 - b. Si ces commandes sont présentes, vous pouvez passer à la suite.
5. Observez la liste précédente, et vous pourrez voir :

```
turtlesim/KillRequest
```

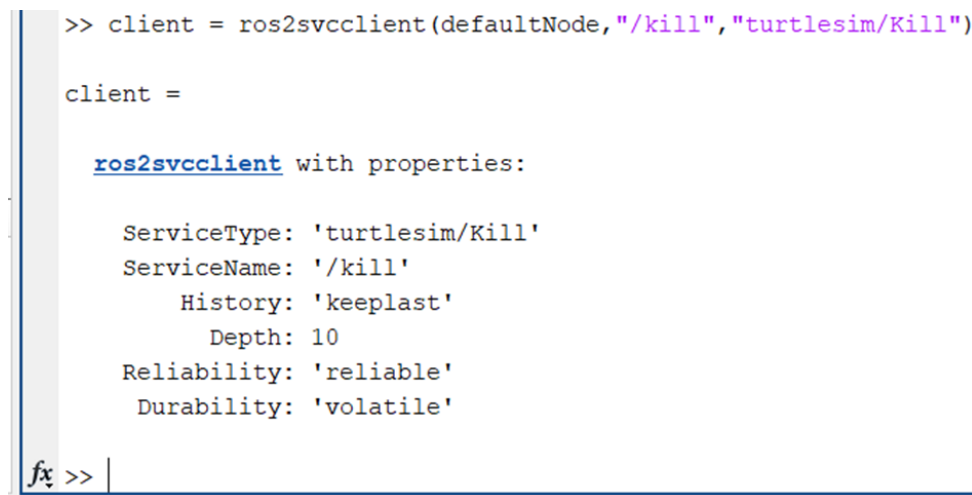
Malheureusement, Matlab a des règles particulières en termes de notation, et lors de l'appel à un service, il va rajouter lui-même le terme "Request" à la fin du type de service. Le type de service, correspondant au service `/kill`, à envoyer sera donc « `turtlesim/Kill` ».

3.2 Lancer le client de ce service

1. Retournez dans Matlab et observez la commande « `ros2svcclient` » pour créer un client de service :

```
client = ros2svcclient(node, servicename, servicetype)
```

2. utilisez la commande « `ros2svcclient` » pour créer notre client de service :



```
>> client = ros2svcclient(defaultNode, "/kill", "turtlesim/Kill")

client =

    ros2svcclient with properties:

        ServiceType: 'turtlesim/Kill'
        ServiceName: '/kill'
           History: 'keeplast'
            Depth: 10
    Reliability: 'reliable'
    Durability: 'volatile'

fx >> |
```

Notre client de service est maintenant créé. . Passons donc à la création du message à envoyer au service.

3.3 Créer le message à envoyer au service

Pour cela nous allons utiliser une des capacités de Matlab qui est de créer directement un message de type adapté à un client de service.

1. Observez la structure de la commande suivante :

```
msg = ros2message(client)
```

Elle permet de renvoyer différentes informations sur le type de message de client.

2. Tapez la commande suivante :

```
>> message = ros2message(client)

message =

  struct with fields:

    MessageType: 'turtlesim/KillRequest'
    name: ''

fx >> |
```

On voit bien que le message est bien du type voulu , mais que le name est vide. Pour cela nous allons donner le nom de la tortue que nous voulons faire disparaître qui est "turtle1".

3. Nous pouvons donc utiliser la commande suivante, pour lui attribuer le nom "turtle1" :

```
>> message.name='turtle1'

message =

  struct with fields:

    MessageType: 'turtlesim/KillRequest'
    name: 'turtle1'

fx >>
```

Le message possède maintenant bien le nom de notre tortue.

3.4 Envoyer le message au service

1. Envoyez ce message via le client, créé plus tôt. Pour cela nous allons utiliser la commande "call". Observez la structure de la commande suite :

```
response = call(serviceclient,requestmsg)
```

2. Utilisez cette fonction "call" :

```
>> call(client,message)

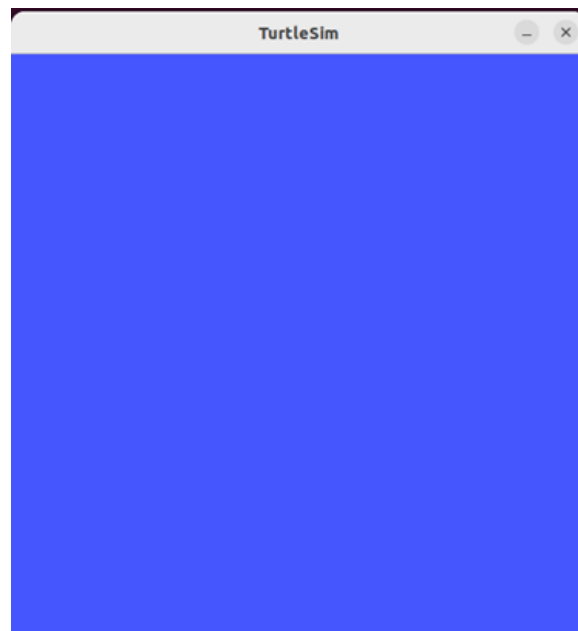
ans =

  struct with fields:

    MessageType: 'turtlesim/KillResponse'

fx >> |
```

3. Observez la fenêtre TurtleSim. La tortue a bien disparu :



4 Utiliser le service /spawn avec Matlab

Nous allons maintenant reprendre les mêmes étapes mais cette fois-ci pour recréer notre tortue avec le service /spawn.

4.1 Trouver le type de ce service sur Matlab

1. Vous pouvez voir ce service dans la liste des services :

```
>> ros2 service list
/_matlab_introspec__0_rmw_fastrtps_cpp/describe_parameters
/_matlab_introspec__0_rmw_fastrtps_cpp/get_parameter_types
/_matlab_introspec__0_rmw_fastrtps_cpp/get_parameters
/_matlab_introspec__0_rmw_fastrtps_cpp/list_parameters
/_matlab_introspec__0_rmw_fastrtps_cpp/set_parameters
/_matlab_introspec__0_rmw_fastrtps_cpp/set_parameters_atomically
/clear
/kill
/reset
/spawn
/turtle1/rotate_absolute/_action/cancel_goal
/turtle1/rotate_absolute/_action/get_result
/turtle1/rotate_absolute/_action/send_goal
/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
fx >> |
```

2. Cherchez le type de ce service dans la liste services et type. Vous devriez trouver ce type :

```
turtlesim/SpawnRequest
```

4.2 Lancer le client de ce service

1. Créez le client de service avec la commande suivante :

```
>> client2 = ros2svcclient(defaultNode, "/spawn", "turtlesim/Spawn")

client2 =

    ros2svcclient with properties:

    ServiceType: 'turtlesim/Spawn'
    ServiceName: '/spawn'
        History: 'keeplast'
        Depth: 10
    Reliability: 'reliable'
    Durability: 'volatile'
fx >> |
```

4.3 Créer le message à envoyer au service

Pour cela nous allons utiliser une des capacités de Matlab qui est de créer directement un message de type adapté à un client de service.

1. Créez le message correspondant à ce service avec la commande suivante :

```
>> message2 = ros2message(client2)

message2 =

  struct with fields:

    MessageType: 'turtlesim/SpawnRequest'
             x: 0
             y: 0
            theta: 0
             name: ''
```

En plus du nom, nous allons donner une position de base à notre tortue. Le centre de l'image se trouve à la position (5.54,5.54), nous allons donc faire apparaître notre tortue ici. Comme le type demandé est un single et que Matlab crée automatiquement des double, il faudra préciser single lors du paramétrage de la position.

2. Fournissez les trois commandes suivantes pour paramétrer le message :

```
>> message2.x=single(5.54)
>> message2.y=single(5.54)
>> message2.name='turtle1'
```

3. Vous obtiendrez donc dans la console le récapitulatif du message construit :

```
message2 =

  struct with fields:

    MessageType: 'turtlesim/SpawnRequest'
             x: 5.5400
             y: 5.5400
            theta: 0
             name: 'turtle1'
```

4.4 Envoyer le message au service

1. Envoyez ce message via le client, créé plus tôt. Utilisez la fonction "call" :

```
>> call(client2,message2)

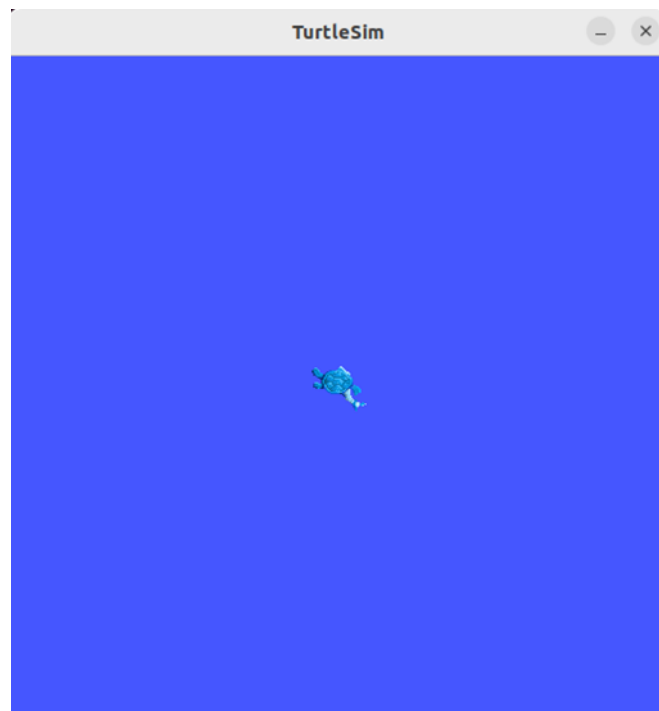
ans =

    struct with fields:

        MessageType: 'turtlesim/SpawnResponse'
        name: 'turtle1'

fx >> |
```

2. Observez la fenêtre TurtleSim. La tortue a bien disparu :



5 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