Rapport du TP1

SLAM

5 Septembre 2024

Elèves :
Guillaume Charvolin
-
Promotion EPITA 2025

Table des matières

1 Partie II **1**
1.1 Question1 1
1.2 Question2 2
1.3 Question 3 6
1.4 Question4 7

2 Exercise III **8**
2.1 Question1 8
2.2 Question2 8
2.3 Question3 11

1 Partie II

1.1 Question1

On obtient la position du package avec la commande :

```
:# ros2 pkg prefix turtlesim

/opt/ros/humble
```

Le package turtlesim est donc situé dans /opt/ros/humble.

- Dans /opt/ros/humble/share/turtlesim il y à des fichiers de configurations.
- Dans /opt/ros/humble/lib/turtlesim il y à les binaires du package, notamment les binaire des noeuds.
- Et enfin dans /opt/ros/humbles/include/turtlesim/turtlesim/ on trouve des headers et du code complémentaires.

Pour avoir la liste des noeuds de turtlesim :

```
:# ros2 pkg executables turtlesim

draw_square
mimic
turtle_teleop_key
turtlesim_node
```

Dans le dossier share dans le fichier package.xml on obtient la liste des ses dépendances :

```
<depend>ament_index_cpp</depend>
<depend>geometry_msgs</depend>
<depend>rclcpp</depend>
<depend>rclcpp_action</depend>
<depend>std_msgs</depend>
<depend>std_srvs</depend>
```

1.2 Question2

1.
Pour lancer le node :

```
:# ros2 run turtlesim turtlesim\_node
```

a.
On obtient la liste des topics avec :

```
:# ros2 topic list

/parameter_events
/rosout
/turtle1/cmd_vel
/turtle1/color_sensor
/turtle1/pose
```

On obtient les informations qui nous intéresse sur les topics écouté et sur ceux publiés en faisant :

```
#: ros2 node info /turtlesim
```

Dans la section subscribers, il s'agit des topic écouter par le noeud :

```
/parameter_events
/turtle1/cmd_vel
```

On peut obtenir des informations sur ces topic en utilisant :

```
#: ros2 topic info /<nom_du_topic>
```

Et leurs descriptions respective est donc :

```
/parameter_events
    Type: rcl_interfaces/msg/ParameterEvent
    Publisher count: 2
    Subscription count: 2
/turtle1/cmd_vel
    Type: geometry_msgs/msg/Twist
    Publisher count: 0
    Subscription count: 1
```

Subscribers :
- /parameter_events : Permet au nœud de réagir aux changements de ses paramètres.
- /turtle1/cmd_vel : Reçoit les commandes de mouvement pour contrôler la tortue.

c.
Dans la section publishers, il s'agit des topic sur lesquels le noeud publie des messages :

```
/parameter_events
/rosout
/turtle1/color_sensor
/turtle1/pose
```

et leur descriptions respectives :

```
/parameter_events
    Type: rcl_interfaces/msg/ParameterEvent
    Publisher count: 2
    Subscription count: 2
/rosout
    Type: rcl_interfaces/msg/Log
    Publisher count: 2
    Subscription count: 0
/turtle1/color_sensor
    Type: turtlesim/msg/Color
    Publisher count: 1
    Subscription count: 0
/turtle1/pose
    Type: turtlesim/msg/Pose
    Publisher count: 1
    Subscription count: 0
```

En se basant sur leur type on peut déduire :
Publishers :

- /parameter_events : Publie les événements relatifs aux changements de paramètres dynamiques du nœud.
- /rosout : Publie les logs
- /turtle1/color_sensor : Publis la couleur détecté par la tortue
- /turtle1/pose : Publie la position et l'orientation de la tortue

2.
Je construit un bag pour enregistrer les différents messages sur les topics décrit précédemment.

```
:# ros2 bag record -a -o /tmp/turtle_record.bag
```

Ensuite je publie le message sur un autre terminal avec la commande :

```
:# ros2 topic pub /turtle1/cmd_vel geometry_msgs/msg/Twist "{linear: {x: 1.0, y:
0.0, z: 0.0}, angular: {x: 0.0, y: 0.0, z: 1.0}}" --once
```

Je ferme le bag et je récupère ces infos avec la commande :

```
:# ros2 bag info /tmp/turtle_record.bag
```

```
Topic information: Topic: /turtle1/cmd_vel | Type: geometry/msg/Twist | Count: 1 |
Topic: /parameter_events | Type: rcl_interfaces/msg/ParameterEvent | Count: 1
Topic: /rosout | Type: rcl_interfaces/msg/Log | Count: 0 |
Topic: /turtle1/color_sensor | Type: turtlesim/msg/Color | Count: 824 |
Topic: /turtle1/pose | Type: turtlesim/msg/Pose | Count: 824 |
```

En écoutant sur les topic avec :

```
:# ros2 topic echo <topic> <message\_type>
```

Et en lançant le bag avec :

```
:# ros2 bag play /tmp/turtle\_record --clock
```

On obtient les différents messages :

- Sur le topic /turtle1/cmd_vel, il s'agit de notre requête de deplacement.
- Sur le topic /parameter_events est publier un compte-rendu du paramètre qui à été mis à jour.
- Sur le topic turtle1/color_sensor est publier en continue la couleur sur laquel est la tortue.
- Sur le topic turtle1/pose est publier en continue la pose de la tortue.

Le message dans parameter_events (tronqué) :

```
stamp:
  sec: 1725546349
  nanosec: 519845268
node: /_ros2cli_1353
new_parameters:
- name: use_sim_time
  value:
    type: 1
    bool_value: false
    integer_value: 0
    double_value: 0.0
    ...
    ...
changed_parameters: []
deleted_parameters: []
```

Ce message montre qu'un noeud à modifier l'un de ces paramètres.

3. b.

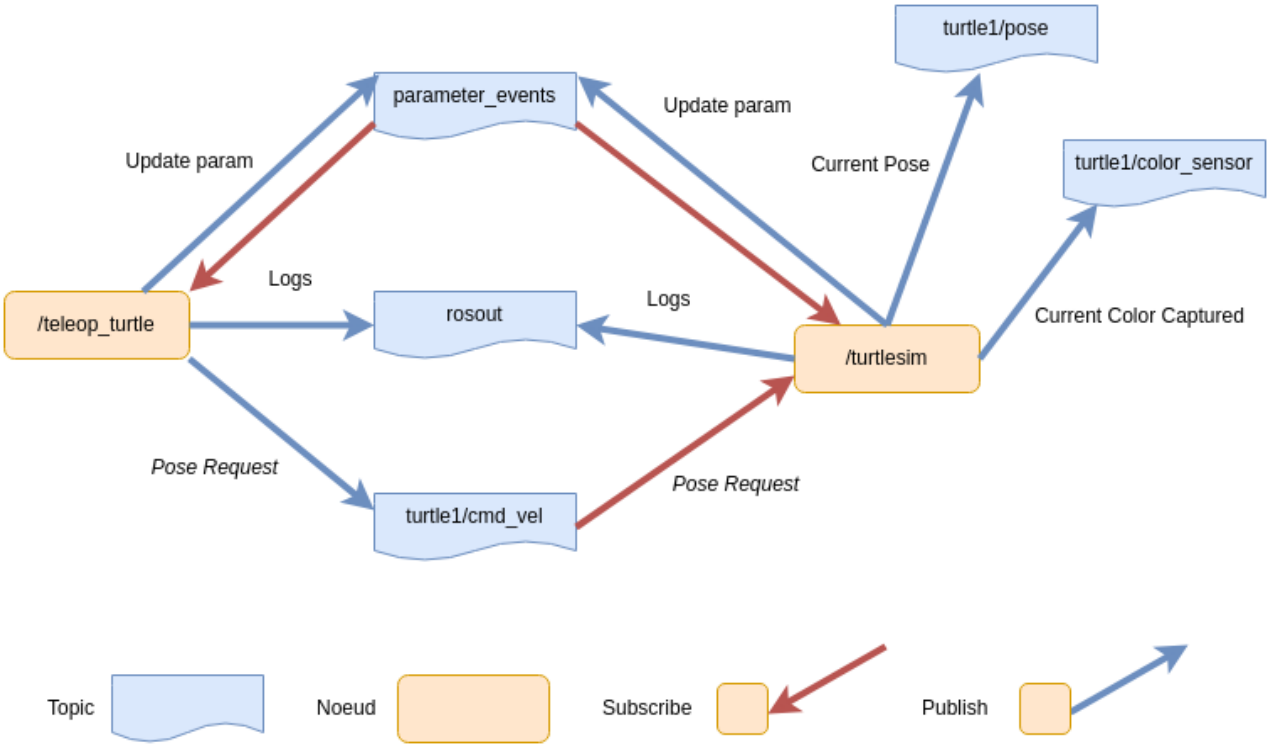
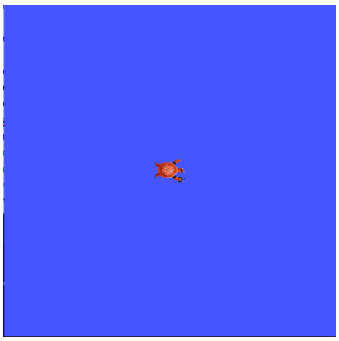


FIGURE 1 – Simulation Tortue - node, topic, message relation

Lorsqu'une flèche est appuyés le noeud /teleop_turtle reçois l'information et publie sur turtle/cmd_vel une requête de déplacement qui est traiter par /turtlesim qui écoute turtle1/cmd_vel, la position de la tortue est publié en temps réel par /turtlesim au topic turtle1/pose (pas seulement lorsqu'elle se déplace).

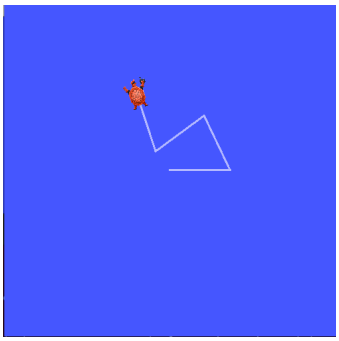
1.3 Question 3



1. Pour enregistrer un bag avec quelques déplacements, il suffit de créer un bag et de lui faire enregistrer 'cmd_vel'.

```
:# ros2 bag record -o /tmp/fast_turtle.bag /turtle1/cmd_vel
```

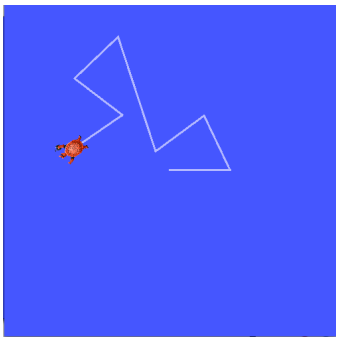
Ensuite, j'entre les différents inputs que je veux faire répéter à la tortue. (Le bag comprend une notion temporel donc les inputs peuvent ne pas être espacé.)



2. Je ferme l'enregistrement du bag. Le bag contient les différentes pose requests (les inputs traduits en requêtes pour déplacer la tortue).

```
:# ros2 bag info /tmp/fast_turtles.bag

Topic information: Topic: /turtle1/cmd_vel | Type:
geometry_msgs/msg/Twist | Count: 7
```



3. Maintenant je le lance :

```
:# ros2 bag play /tmp/fast_turtles.bag
```

1.4 Question4

1. Pour lister les services :

```
:# 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
```

2. J'utilise la commande :

```
:# ros2 service -h
```

Je prend connaissance de la commande ros2 service type <type_name> ce qui va me servir pour faire un appel au service /spawn
Le type du service /spawn est donc turtlesim/srv/Spawn Avec ce type je peux connaître la forme de l'argument à passer dans le call avec :

```
:# 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
```

J'ajoute James, ma tortue grecque :

```
 :# ros2 service call /spawn turtlesim/srv/Spawn '{x: 5.0, y: 5.0, theta: 0.0, name:
"James"}'
```

3. Avec ros2 service list de nouvelle option sont apparus pour déplacer James, à l'aide de type et interface je construit la requête :

```
:# ros2 service call /James/teleport_absolute turtlesim/srv/TeleportAbsolute '{x:
7.0, y: 7.0, theta: 0.0}'
```

2 Exercise III

2.1 Question1

Après compilation ./Printer donne :

```
    printing stuffs
```

Après création du package :

```
    # ros2 run my_package printer_node

    [INFO] [1725627502.299036167] [simple_node]: Hello, ROS 2!
```

2.2 Question2

1

test.launch.py

```
from launch import LaunchDescription
from launch_ros.actions import Node

def generate_launch_description():
    return LaunchDescription([
        Node(
            package='turtlesim',
            executable='turtlesim_node',
            name='turtlesim'
        ),
        Node(
            package='my_package',
            executable='printer_node',
            name='printer_node',
            parameters=[{
                'message': 'Hello? fixed'
            }]
        )
    ])
```

2.
printer_node.cpp

```
#include "rclcpp/rclcpp.hpp"
#include <string>

class SimpleNode : public rclcpp::Node
{
public:
  SimpleNode() : Node("simple_node")
  {
    this->declare_parameter<std::string>("message", "Hello, ROS 2!");

    std::string message;
    this->get_parameter("message", message);

    RCLCPP_INFO(this->get_logger(), message.c_str());
  }
};

int main(int argc, char * argv[])
{
  rclcpp::init(argc, argv);
  auto node = std::make_shared<SimpleNode>();
  rclcpp::spin(node);
  rclcpp::shutdown();
  return 0;
}
```

On peut tester si le paramètre existe avec set parameter :

```
# ros2 param set /printer_node message "new_message"
Set parameter successful
```

3.

test.launch.py

```
from launch import LaunchDescription
from launch_ros.actions import Node
from launch.actions import DeclareLaunchArgument
from launch.substitutions import LaunchConfiguration

def generate_launch_description():
    message_param = DeclareLaunchArgument(
        'message',
        default_value='Hello? interface',
        description='Message to print'
    )

    return LaunchDescription([
        Node(
            package='turtlesim',
            executable='turtlesim_node',
            name='turtlesim'
        ),
        Node(
            package='my_package',
            executable='printer_node',
            name='printer_node',
            parameters=[{
                'message': LaunchConfiguration('message')
            }]
        ),
        message_param
    ])
```

Je peux maintenant utilise ros2 launch avec des paramètres :

```
:# ros2 launch my_package test.launch.py message:="The message"

[INFO] [launch]: All log files can be found below /root/.ros/log
/2024-09-06-13-52-56-412723-1174262e542e-33987
[INFO] [launch]: Default logging verbosity is set to INFO
[INFO] [turtlesim_node-1]: process started with pid [33988]
[INFO] [printer_node-2]: process started with pid [33990]
[turtlesim_node-1] QStandardPaths: XDG_RUNTIME_DIR not set, defaulting to '/tmp/
runtime-root'
[printer_node-2] [INFO] [1725630776.494781345] [printer_node]: The message
[turtlesim_node-1] [INFO] [1725630776.527995910] [turtlesim]: Starting turtlesim
with node name /turtlesim
[turtlesim_node-1] [INFO] [1725630776.533486147] [turtlesim]: Spawning turtle [
turtle1] at x=[5.544445], y=[5.544445], theta=[0.000000]
```

2.3 Question3

1.

```
ros2 bag info data/car_vlp16

Files:             car_vlp16.db3
Bag size:          265.2 MiB
Storage id:        sqlite3
Duration:          76.462s
Start:             Oct 21 2020 07:49:58.201 (1603266598.201)
End:               Oct 21 2020 07:51:14.664 (1603266674.664)
Messages:          759
Topic information: Topic: velodyne_points | Type: sensor_msgs/msg/PointCloud2 |
Count: 759 | Serialization Format: cdr
```

Il y a 759 points 2d dans le bag car_vlp16
Il faut ajouter l'aquisition des points avec "add" puis nommé fixed_frame "velodyne" qui semble être le nom du LiDAR

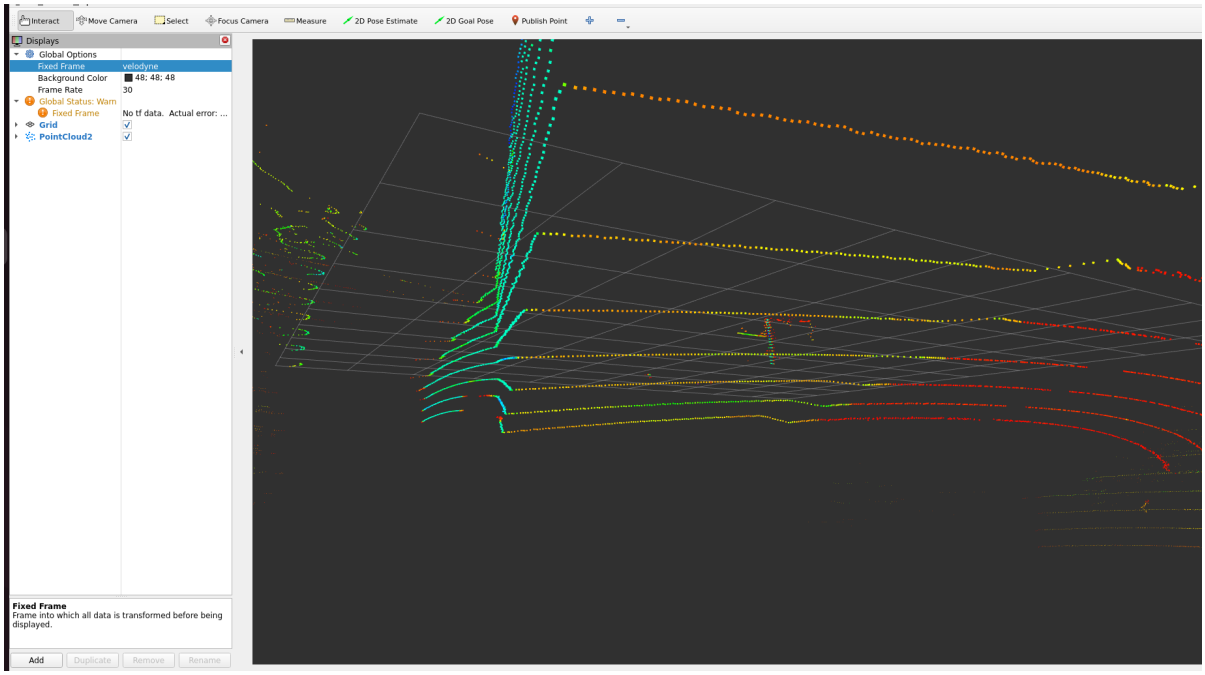


FIGURE 2 – Image de la capture des points de velodyne avec Rviz2