Understanding ROS 2 Topics

Estimated time to completion: **8 minutes**

3.2 Basic Topic commands

Before diving into a full understanding of **Topics**, let's start by interacting with them a bit. To help with this, you'll be introduced to some basic command-line tools that **ROS 2** provides for working with **topics**.

Open Terminal #1 and run the commands to source it for working with ROS2.

- TIP -

- No matter where you are, in order to be able to execute **ROS2** commands, you need to source ROS2.
- We can directly source the basic ROS2 installation:

Execute in Terminal #1

In [ ]:

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

- Or source your ROS2 workspace.
- Both things will give you access to the ROS2 commands.
- The main difference is that, in order to be able to use your ROS2 packages that are compiled inside the `ros2_ws`, you need to source this workspace.

In [ ]:

```
cd ~/ros2_ws
source install/setup.bash
```

- END -

Now you're ready to get to work. Start by trying the following command:

Execute in Terminal #1

In [ ]:

```
ros2 topic -h
```

You should get a result equal to the following:

Terminal #1 Output

```
usage: ros2 topic [-h] [--include-hidden-topics]
                  Call `ros2 topic  -h` for more det
ailed usage. ...

Various topic related sub-commands

options:
  -h, --help            show this help message and e
xit
  --include-hidden-topics
                        Consider hidden topics as we
ll

Commands:
  bw     Display bandwidth used by topic
  delay  Display delay of topic from timestamp in he
ader
  echo   Output messages from a topic
  find   Output a list of available topics of a give
n type
  hz     Print the average publishing rate to screen
  info   Print information about a topic
  list   Output a list of available topics
  pub    Publish a message to a topic
  type   Print a topic's type

  Call `ros2 topic command -h` for more detailed usa
ge.
```

The most important results to review in this section of the course are the following: `echo`, `info`, `list`, `pub`.

- Example 3.1 -

Type the following command in the Terminal:

Execute in Terminal #1

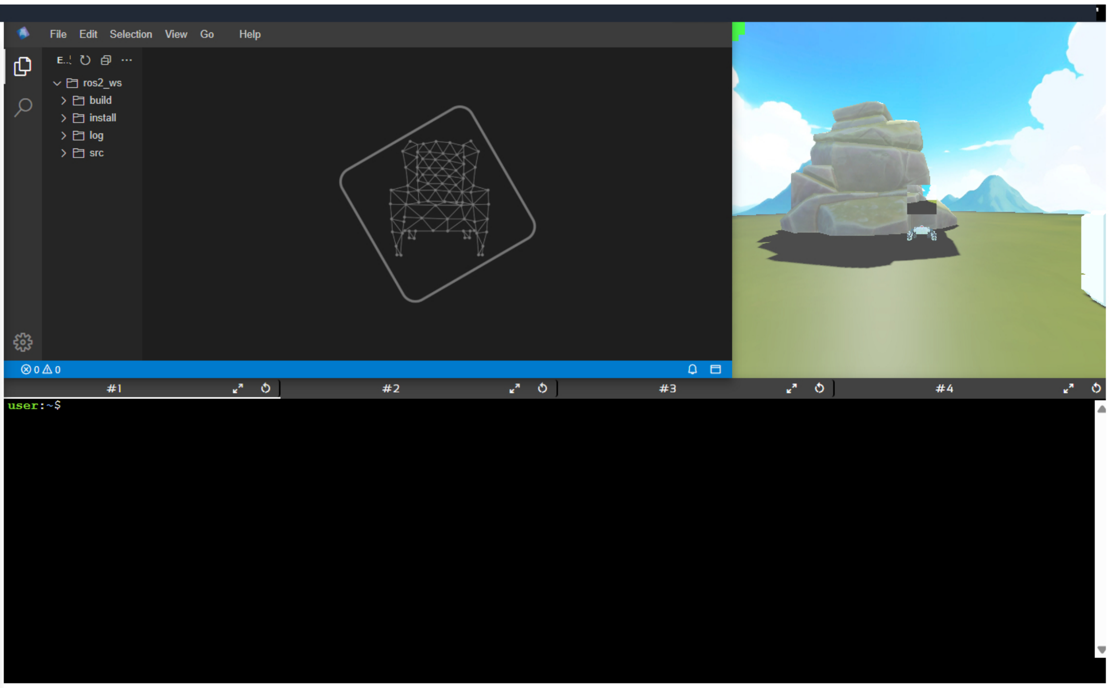
In [ ]:

```
ros2 topic list
```

You should get a result similar to the one in the image below:

Terminal #1 Output

```
/clock
/cmd_vel
/gazebo/link_states
/gazebo/model_states
/joint_states
/laser_scan
/leo/camera/camera_info
/leo/camera/image_raw
/leo/camera/image_raw/compressed
/leo/camera/image_raw/compressedDepth
/leo/camera/image_raw/theora
/odom
/parameter_events
/performance_metrics
/robot_description
```



```
user:~$
```

```
/rosout
/tf
/tf_static
```

- You have listed the `topics` currently available in your environment to work on.
- These `topics` are used and created by the simulation.

- Notes -

The **`ros2 topic list`** command prints a list of all available `Topics`.

- End of Notes -

Now try searching for a specific `topic`. To do this, use the following command:

Execute in Terminal #1

In [ ]:

```
ros2 topic list | grep cmd_vel
```

Terminal #1 Output

```
/cmd_vel
```

- It looks like you have found it!
- Now, try to get some information about the `topic`.
- To do that, we use the `ros2 topic info` command:

- Notes -

The **`ros2 topic info`** command is used to get information about a specific `Topic`. The command structure is as follows:

In [ ]:

```
ros2 topic info <topic_name>
```

- End of Notes -

Execute in Terminal #1

- In this case, the `topic_name` = `/cmd_vel`

In [ ]:

```
ros2 topic info /cmd_vel
```

Terminal #1 Output

```
Type: geometry_msgs/msg/Twist
Publisher count: 0
Subscription count: 2
```

Let's break down the output:

- **Type:** Refers to what we know as **ROS2** `interfaces` associated with the `Topic` with which you need to work with. `Interfaces` define the type of data transmitted through this `topic`. It could be **images**, **string**, **laser data** or whatever you need to transmit. In this case its `Twist` which is a kind of message used for **moving robots around.**

- `Publisher` **count:** Refers to the number of nodes that **publish data to this** `topic`. In those case there are **NONE**, which means that nothing is introducing data in this `topic`.

- `Subscription` **count:** Refers to the number of active nodes that **read data from this** `topic`. In this case we have **TWO** because each pair of wheels works independently, one is for the **front wheels** and the other for the **rear wheels**.

- Notes -

Don't worry about understanding `Publishers`, `Subscribers`, and `Interfaces` right now. These concepts will be explained in more detail later in this unit.

- End of Notes -

Okay, now try running the same command with different `topics` to see what differences you find:

Execute in Terminal #1

In [ ]:

```
ros2 topic info /laser_scan
```

Terminal #1 Output

```
Type: sensor_msgs/msg/LaserScan
Publisher count: 1
Subscription count: 0
```

- **Type:** In this case it's `LaserScan` data, which make sense because the data inside this `topic` is the readings of the **laser sensor**.

- `Publisher` **count:** In this case we have **ONE** which will probably be the **laser sensor** that is introducing the laser readings from the simulation.

- `Subscription` **count:** Refers to the number of active nodes that **read data form this** `topic`. In this case **NONE** because no one is using this laser data.

- This is the opposite case compared to the `topic`.

Try one more.

▶ Execute in Terminal #1

In [ ]:

```
ros2 topic info /leo/camera/image_raw
```

Terminal #1 Output

```
Type: sensor_msgs/msg/Image
Publisher count: 1
Subscription count: 0
```

- **Type:** In this case it's `Image` data, which makes sense because the data inside this `topic` is the image data from the Mars Rover camera.
- `Publisher` **count:** In this case we have **ONE** which will probably be the **camera sensor** that is introducing the camera images from the simulation.
- `Subscription` **count:** Refers to the number of active nodes that **read data form this** `topic`. In this case **NONE** because no one is using this image data.

- End of Example 3.1 -

24/09/2024