Understanding ROS 2 Topics

Estimated time to completion: **30 minutes**

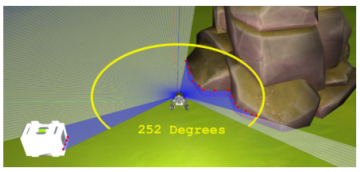
3.3 Explore the Mars Rover Topics

So far, so good! You've learned how to list the `topics` in your ROS 2 system, search for a specific `topic`, and gather information about it. Now, let's dive deeper into working with `topics`. For this, we'll explore the topics provided by the **Mars Rover** robot!

- Example 3.2 -

MISSION: Detect Obstacles for the Mars Rover

- To prevent the Mars Rover from colliding with obstacles during its missions, we need to detect nearby objects.
- A great option for this task is a laser sensor, which measures the distance to objects both in front of and on the sides of the robot.
- Here's its range:



- You can see that each laser beam will measures the distance to the first object detected.
- You can see where the distance is lower than the laser's maximum range with those **red dots**.
- All this information is **PUBLISHED**, or as it is called in ROS2, inside the `topic` `/laser_scan`.
- This laser data is published inside the `/laser_scan` `topic` so that any other system can use it.
- It uses an `interface` called `sensor_msgs/msg/LaserScan`, which we discovered previously using the `ros2 topic info /laser_scan` command.
- `sensor_msgs` is the package installed in the system that defines this `LaserScan`.
- We need to understand the data structure of this `LaserScan` `topic` `interface`.
- To do this, use the **ROS2 command** `ros2 interface show`:

In [ ]:

```
ros2 interface show <package_name>/msg/<interface_name>
```

- In our case:
  - `package_name` = `sensor_msgs`
  - `interface_name` = `LaserScan`
- Let's execute the command:

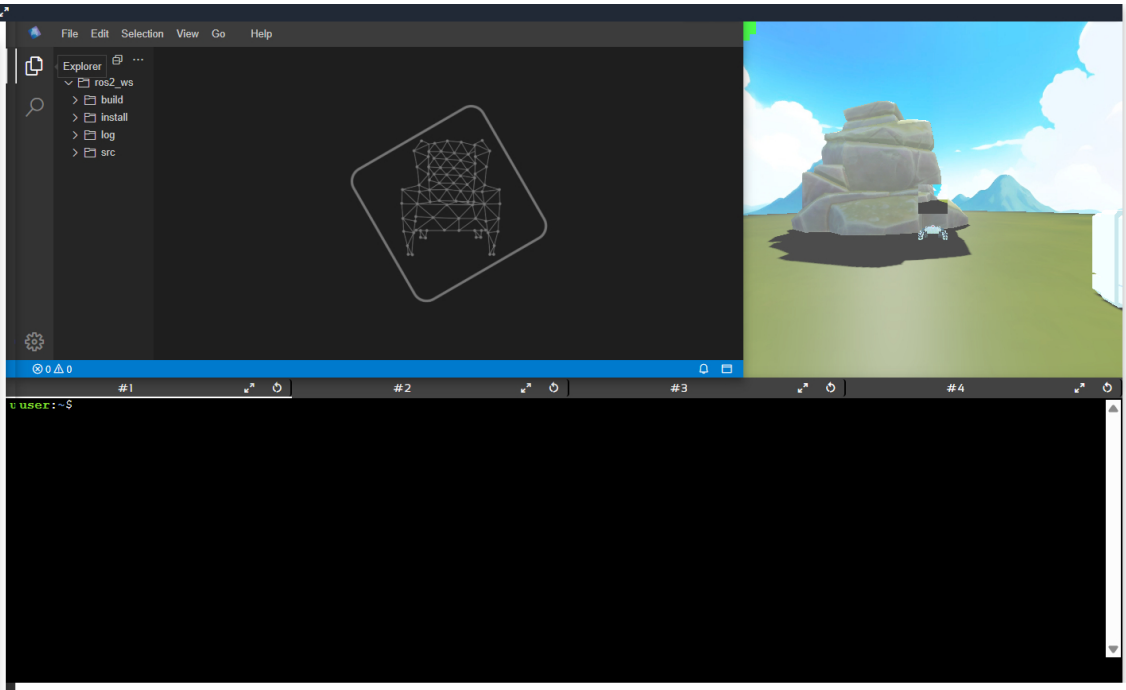
Execute in Terminal #1

In [ ]:

```
ros2 interface show sensor_msgs/msg/LaserScan
```

Terminal #1 Output

```
# Single scan from a planar laser range-finder
#
# If you have another ranging device with different
behavior (e.g. a sonar
# array), please find or create a different message,
since applications
# will make fairly laser-specific assumptions about
this data

std_msgs/Header header # timestamp in the header is
the acquisition time of
        builtin_interfaces/Time stamp
                int32 sec
                uint32 nanosec
        string frame_id
                        # the first ray in the
scan.
                        #
                        # in frame frame_id, an
gles are measured around
                        # the positive Z axis
(counterclockwise, if Z is up)
                        # with zero angle being
forward along the x axis

float32 angle_min           # start angle of the sc
an [rad]
float32 angle_max           # end angle of the scan
[rad]
float32 angle_increment     # angular distance betw
een measurements [rad]

float32 time_increment      # time between measurem
ents [seconds] - if your scanner
                        # is moving, this will
be used in interpolating position
                        # of 3d points
float32 scan_time           # time between scans [s
econds]

float32 range_min           # minimum range value
[m]
float32 range_max           # maximum range value
[m]

float32[] ranges            # range data [m]
                        # (Note: values < range
_min or > range_max should be discarded)
float32[] intensities       # intensity data [devic
e-specific units].  If your
```



```
                                # device does not provi
de intensities, please leave
                                # the array empty.
```

- The terminal shows the structure of a `LaserScan` message. Complicated? Let's break it down a bit.

Take note of the most critical information:

- `angle_min` : Refers to the minimum angle at which the laser operates. It can also be seen as the starting angle of the scan.
- `angle_max` : Refers to the maximum angle at which the laser operates. It can also be seen as the ending angle of the scan
- `angle_increment` : Refers to the angular distance between measurements [rads].

This information is useful because it will help you calculate the indices for each sensor position.

- For example, for this sensor, you can calculate the number of samples and their angle positions using these formulas:

$$number\_of\_samples = (angle\_max - angle\_min) / incremental\_angle$$

$$angle\_position(index) = angle\_min + (index * incremental\_angle)$$

Also, you have:

- `range_min` : Minimum range of the laser [meters].
- `range_max` : Maximum range of the laser [meters].
- `ranges` : An array containing laser distance detections of each of the laser samples [meters].

Each value in the `ranges` array represents the sensor's reading in a specific direction. Thus, each direction is represented by a position in this array.

- Let's see the real data **PUBLISHED** by our Mars Rover laser sensor in the `topic` `/laser_scan` .
- To do this we will use another ROS2 command named `ros2 topic echo` :

Execute in Terminal #1

In [ ]:

```
ros2 topic echo /laser_scan
```

- Your terminal may be flooded with many messages. Remember, you can press `Ctrl+C` to stop the command from running.
- Once you do that, look for a message similar to the following:

Terminal #1 Output

```
header:
  stamp:
    sec: 7587
    nanosec: 989000000
  frame_id: _laser_scan_frame
angle_min: -2.200000047683716
angle_max: 2.200000047683716
angle_increment: 0.022110553458333015
time_increment: 0.0
scan_time: 0.0
range_min: 0.10000000149011612
range_max: 20.0
ranges:
- .inf
- .inf
... [REMOVED FOR PREVETY]
- .inf
- .inf
- .inf
- 1.237236738204956
- 1.153326630592346 2
... [REMOVED FOR PREVETY]
- 0.7254517674446106
- 0.7442454695701599
- '...'
intensities:
- 0.0
... [REMOVED FOR PREVETY]
- 0.0
- 0.0
- '...'
---
```

- This is **REAL LASER DATA FROM THE Mars Rover laser sensor**!
- Let's comment on some values:
  - `ranges .inf` : `inf` means that the laser beam didn't detect anything beyond its **range_max**.
  - `angle_min: -2.200000047683716` , `angle_max: 2.200000047683716` , `angle_increment: 0.022110553458333015` : These values allow us to calculate the number of samples and angle direction of each of the samples:

$$number\_of\_samples = (angle\_max - angle\_min) / incremental\_angle = (2.2 - (-2.2))/0.02211 = 200 samples$$

$$angle\_position(index) = angle\_min + (index * incremental\_angle) = -2.2 + (index * 0.02211)$$

$$middle\_index\_sample = number\_of\_samples/2.0 = 100$$

$$angle\_position(index = middle\_index\_sample) = -2.2 + (100 * 0.02211) \approx 0.0$$

- In this case, we have **200 samples**.
- And if we want the sample right in the middle of the range, which is the **FRONT of the Mars Rover**, we get a direction of **0.0 degrees**.

- TIP -

- You can read only a **SINGLE latest ROS2** `interface` **message** from a `topic` using the following:

Execute in Terminal #1

In [ ]:

```
ros2 topic echo /laser_scan --once
```

- You can only echo a **certain field from the** `interface` of a `topic`.
- For example, we know that the `LaserScan` `interface` has an element named `ranges`, which is the only one we are really interested in because it contains the distance detection data.
- So, let's **ONLY echo** that field using the `--field` argument:

`--field FIELD`

- Echo a selected field of a message.
- Use '.' to select sub-fields.
- For example, to echo the seconds field of a sensor_msgs/msg/LaserScan message:
    - `ros2 topic echo /laser_scan --field header.stamp.sec`
- In our case, we want the `ranges` field

Execute in Terminal #1

In [ ]:

```
ros2 topic echo /laser_scan --field ranges
```

Terminal #1 Output

```
array('f', [inf, inf, inf, inf, inf, inf, inf, inf,
inf, inf, inf, inf, inf, inf, inf, inf, inf, inf, in
f, inf, inf, inf, inf, inf, inf, inf, inf, 3.8130645
751953125, 3.6558547019958496, 3.51468563079834, 3.3
907859325408936, 3.3040709495544434, 3.2232167720794
678, 3.1622393131256104, 3.1188008785247803, 3.12022
0899581909, 3.1231689453125, 3.1276540756225586, 3.1
33685350418091, 3.1412792205810547, 3.15045666694641
1, 3.1616013050079346, 3.1743814945220947, 3.0006661
415100098, 2.817243814468384, 2.667501926422119, 2.5
67586660385132, 2.550252676010132, 2.54551839828491
2, 2.5456557273864746, 2.5470497608184814, 2.5496921
53930664, 2.5535888671875, 2.5587515830993652, 2.578
0959129333496, 2.603687286376953, 2.631091833114624,
2.660393476486206, 2.689133644104004, 2.711207151412
964, 2.7349939346313477, 2.7606186866760254, 2.79371
64306640625, 2.8478903770446777, 2.903876304626465,
2.9602959156036377, 3.020455837249756, 3.10484743118
28613, 3.858429193496704, 3.9134936332702637, 3.9714
70832824707, 4.039212703704834, inf, inf, inf, inf,
inf, inf, inf, inf, inf, inf, inf, inf, inf, inf, in
f, inf, inf, inf, inf, inf, inf, inf, inf, inf, inf,
inf, inf, inf,inf, inf, inf, inf, inf, inf, inf, in
f, inf, inf, inf, inf, inf, inf, inf, inf, inf, inf,
inf, inf, inf, inf, inf, inf, inf, inf, inf, inf, in
f,inf, inf, inf, inf, inf, inf, inf, inf, inf, inf,
inf, inf, inf, inf, inf, inf, inf, inf, inf, inf, in
f, inf, inf, inf, inf, inf, inf, inf, inf,inf, inf,
inf, inf, inf, inf, inf, inf, inf, inf, inf, inf, in
f, inf, inf, inf, inf, inf, inf, inf, inf, inf, inf,
inf, inf, inf, inf, inf, inf,inf, 1.333944439888000
5, 1.2612237930297852, 1.1965770721435547, 1.1387636
66152954, 1.0867862701416016, 1.0398328304290771, 0.
9972357749938965, 0.9584415555000305, 0.922986805438
9954, 0.8904815316200256, 0.8605940937995911, 0.8330
419063568115])
```

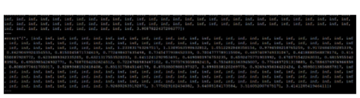
- You now **ONLY** see the `ranges` data.
- Let's **make the Mars Rover turn** using the **keyboard teleoperation node** and see how the values change.
- Press the keyboard key `j` to make the **Mars Rover turn**.

Execute in Terminal #2

In [ ]:

```
ros2 run teleop_twist_keyboard teleop_twist_keyboard
```

- You should see how the `inf` values get replaced by normal measurements, which makes sense when turning around.



- If you want to know all the optional arguments for `ros2 topic echo` or any other **ROS2 command**, just type the command and `--help`:

Execute in Terminal #1

In [ ]:

```
ros2 topic echo --help
```

- END TIP -

- Self-paced example 3.1 -

Investigate the structure of the camera `topic`

- We need to know how the **Mars Rover camera** `topic` `/leo/camera/image_raw` works.

- Execute the necessary commands to be able to:
  - Determine the structure of the `interface` message used by `/leo/camera/image_raw` `topic`.
  - See only the image `data` field. Be careful to `CTRL+C` to stop the infor stream because images hold a lot of data and in slow internet connections might cause lag.

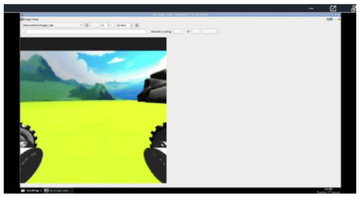
- You can visualize the **image** `topics` **data** graphically using this tool, which essentially performs a ROS2 `topic` echo and uses the image data to display it in a more comprehensible way than just an array of numbers:

▶ Execute in Terminal #1

In [ ]:

```
ros2 run rqt_image_view rqt_image_view /leo/camera/image_
```

- You should see the following:



- RESULTS expected -

▶ 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
```

▶ Execute in Terminal #1

In [ ]:

```
ros2 interface show sensor_msgs/msg/Image
```

Terminal #1 Output

```
# This message contains an uncompressed image
# (0, 0) is at top-left corner of image

std_msgs/Header header # Header timestamp should be
acquisition time of image
        builtin_interfaces/Time stamp
                int32 sec
                uint32 nanosec
        string frame_id
                             # Header frame_id shoul
d be optical frame of camera
                             # origin of frame shoul
d be optical center of cameara
                             # +x should point to th
e right in the image
                             # +y should point down
in the image
                             # +z should point into
to plane of the image
                             # If the frame_id here
and the frame_id of the CameraInfo
                             # message associated wi
th the image conflict
                             # the behavior is undef
ined

uint32 height                # image height, that i
s, number of rows
uint32 width                 # image width, that is,
number of columns

# The legal values for encoding are in file src/imag
e_encodings.cpp
# If you want to standardize a new string format, jo
in
# ros-users@lists.ros.org and send an email proposin
g a new encoding.

string encoding       # Encoding of pixels -- channe
l meaning, ordering, size
                      # taken from the list of strin
gs in include/sensor_msgs/image_encodings.hpp

uint8 is_bigendian    # is this data bigendian?
uint32 step           # Full row length in bytes
uint8[] data          # actual matrix data, size is
(step * rows)
```

▶ Execute in Terminal #1

In [ ]:

```
ros2 topic echo /leo/camera/image_raw --field data
```

Terminal #1 Output

```
...
255, 49, 221, 255, 49, 221, 255, 49, 221, 255, 49, 2
21, 255, 49, 221, 255, 49, 221, 255, 49, 221, 255, 4
9, 221, 255, 49, 220, 255, 48, 220, 255, 48, 220, 25
5, 48, 220, 255, 48, 220, 255, 48, 220, 255, 48, 22
0, 255, 48, 220, 255, 48, 220, 255, 48, 220, 255, 4
8, 220, 255, 48, 220, 255, 48, 220, 255, 48, 220, 25
5, 48, 220, 255, 48, 220, 255, 48, 220, 255, 48, 22
0, 255, 48, 220, 255, 48, 220, 255, 48, 220, 255, 4
8, 220, 255, 48, 220, 255, 48, 220, 255, 48, 220, 25
5, 48, 220, 255, 48, 220, 255, 48, 220, 255, 48, 22
0, 255, 48, 220, 255, 48, 220, 255, 48, 220, 255, 4
8, 220, 255, 48, 220, 255, 48, 220, 255, 48, 220, 25
5, 48, 220, 255, 48, 220, 255, 48, 220, 255, 48, 22
0, 255, 48, 220, 255, 48, 220, 255, 48, 220, 255, 4
8, 220, 255, 48, 220, 255, 48, 220, 255, 48, 220, 25
5, 48, 220, 255, 48, 220, 255, 48, 220, 255, 48, 22
0, 255, 48, 220, 255, 48, 220, 255, 48,220, 255, 48,
220, 255, 48, 220, 255, 48, 220, 255, 48, 220, 255,
48, 220, 255, 48, 220, 255, 48, 220, 255, 48, 220, 2
```

```
55, 48, 220, 255, 48, 220,255, 48, 220, 255, 48, 22
0, 255, 48, 220, 255, 48, 220, 255, 48, 220, 255, 4
8, 220, 255, 48, 220, 255, 48, 220, 255, 48, 220, 25
5, 48, 220, 255,48, 220, 255, 48, 131, 148, 45, 41,
41, 41, 41, 41, 41, 41, 41, 41, 41, 41, 41, 41, 41,
41])
```

- End Self-paced example 3.1 -

- Example 3.3 -

- Let's now have a look at the `topic` of `/cmd_vel`, which is used to move the Mars Rover:

Execute in Terminal #1

In [ ]:

```
ros2 topic echo /cmd_vel
```

Terminal #1 Output

- Strange. Nothing appears.
- Is something wrong?
- Okay. Don't panic.
- There is nothing wrong with the command you just typed.
- There is nothing broken in your workspace, either.
- What happens here is that at this moment, nobody is publishing any data to the `Topic` `/cmd_vel`.
- In the previous **two** `topics` `/laser_scan` and `/leo/camera/image_raw`, the sensors were **PUBLISHING** data into those `topics`.
- But now, no one is publishing, only listening or what is known as **SUBSCRIBING**, as we saw when we executed the command of `ros2 topic info /cmd_vel`.

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
```

- So, let's publish something to this `topic`.

- To publish something to this `topic`, you need to **first check what type of** `interface` **the** `Topic` **uses**.
- In this case, we know it's the `geometry_msgs/msg/Twist`
- We can check all the available `interfaces` using the command `ros2 interface list`:

Execute in Terminal #1

In [ ]:

```
ros2 interface list
```

Terminal #1 Output

- The output is much longer but we show here just a small part of it:

```
Messages:
    ackermann_msgs/msg/AckermannDrive
    ackermann_msgs/msg/AckermannDriveStamped
    action_msgs/msg/GoalInfo
..
    geometry_msgs/msg/Twist
..
    visualization_msgs/msg/MeshFile
    visualization_msgs/msg/UVCoordinate
Services:
    action_msgs/srv/CancelGoal
    cartographer_ros_msgs/srv/FinishTrajectory
...
    visualization_msgs/srv/GetInteractiveMarkers
Actions:
    action_tutorials_interfaces/action/Fibonacci
    control_msgs/action/FollowJointTrajectory
...
    turtlebot3_msgs/action/Patrol
    turtlesim/action/RotateAbsolute
```

- Notes -

The **`ros2 interface list`** command prints a list of all available `interfaces`.

- End of Notes -

You should find a similar result in the terminal where you ran the command. Looking at what the Terminal shows you, you will see that the `interfaces` are divided into the following groups:

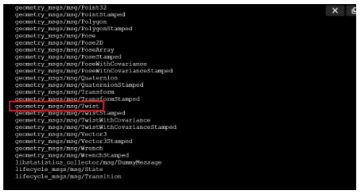
- **Messages:** Can be found as `.msg` files. They are simple text files that describe the fields of a ROS message. You can use them to generate source code for messages.
- **Services:** Can be found as `.srv` files. They are composed of two parts: a request and a response. Both are message declarations.
- **Actions:** Can be found as `.action` files. They are composed of three parts: a goal, a result, and feedback. Each part contains a message declaration.

You are now interested in messages, which are the `interfaces` used by `topics`, so let us go a little deeper into what they are.

- There are two ways to find the descriptions and definitions for messages:

- In the `.msg` files and the `msg/` directory of a ROS package. The `.msg` files are composed of two parts: **fields** and **constants**. You can find more information about this in the official ROS 2 documentation.

Resume what you were doing with the `cmd_vel Topic`. Then, looking at the list of `interfaces` you displayed, you will find the following:



As you may remember, when you were looking for information about the `/cmd_vel Topic`, you saw that this `Topic` uses a type called `geometry_msgs/Twist`. As you now know, it is a message-type `interface`.

- Now we need to get some data about this specific type.
- You can execute the `ros2 interface show` command:

Execute in Terminal #1

In [ ]:

```
ros2 interface show geometry_msgs/msg/Twist
```

Terminal #1 Output

```
# This expresses velocity in free space broken into it

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

What an exciting thing discovery! The last command output shows that the `Twist` message comprises two `Vector3` messages. It consists of a 3-dimensional vector for the linear velocity and another 3-dimensional vector for the angular velocity.

- Now we need an example of how to fill in this message so we can **publish** it to the `/cmd_vel topic`.
- For that we can use the command `ros2 interface proto`.
- This command shows you a prototype (example) of the `interface` that you can use.
- Execute the following line in your `Terminal 1`:

- Notes -

The **`ros2 interface proto`** structure is:

In [ ]:

```
ros2 interface proto <package_name>/msg/<interface_name>
```

- End of Notes -

Execute in Terminal #1

In [ ]:

```
ros2 interface proto geometry_msgs/msg/Twist
```

You should see a result similar to the following in your terminal:

Terminal #1 Output

```
"linear:
  x: 0.0
  y: 0.0
  z: 0.0
angular:
  x: 0.0
  y: 0.0
  z: 0.0
"
```

- Analyze the result obtained by executing this last command `ros2 interface proto`.
- You can see that the structure for writing a message to the `topic /cmd_vel` is composed of **two three-dimensional vectors**.
- One vector is called `linear` with positions for the axes (x, y, z), and another is called `angular` with positions for the same axes (x, y, z).
- Recall that using the `ros2 interface show` command, you verified that the Twist message is composed of two messages of type `Vector3`.
- You will use the `/cmd_vel Topic` to send a message composed of these velocity vectors to move the **Mars Rover**

- All right. You have enough information to start **publishing messages** to the `cmd_vel Topic`.
- To do this, you will use the `geometry_msgs/msg/Twist interface`.

To publish a message to a particular Topic, use the following command:

- Notes -

The **`ros2 topic pub`** command is used to publish messages to a `Topic`. The command structure is as follows:

In [ ]:

```
ros2 topic pub <topic_name> <interface_name> <message>
```

- End of Notes -

- In our case:
  - `topic_name` = `/cmd_vel`
  - `interface_name` = `geometry_msgs/msg/Twist`
  - **`message`** = `"{linear: {x: 0.1, y: 0.0, z: 0.0}, angular: {x: 0.0, y: 0.0, z: 1.0}}"`
- The message value was copied from the `ros2 interface proto geometry_msgs/msg/Twist` output.
- But now we set `linear.x = 1.0`, which will move the robot forward, and `angular.z = 1.0` will turn the robot. Combined, these should make the robot move forward while turning.

Execute in Terminal #1

In [ ]:

```
ros2 topic pub /cmd_vel geometry_msgs/msg/Twist "{linear:
```

You should see the robot doing the following:

- Now if we execute the `ros2 topic echo /cmd_vel`:

Execute in Terminal #2

In [ ]:

```
ros2 topic echo /cmd_vel
```

You should see the robot doing the following:

Terminal #2 Output

```
linear:
  x: 0.1
  y: 0.0
  z: 0.0
angular:
  x: 0.0
  y: 0.0
  z: 1.0
---
linear:
  x: 0.1
  y: 0.0
  z: 0.0
angular:
  x: 0.0
  y: 0.0
  z: 1.0
---
```

- You were able to publish a message to the `/cmd_vel` `Topic`.

You can also check if the `publisher/subscriber` count for the `/cmd_vel` `Topic` has increased.

Execute in Terminal #3

In [ ]:

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

In [ ]:

```
ros2 topic info /cmd_vel
```

Terminal #3 Output

```
Type: geometry_msgs/msg/Twist
Publisher count: 1Subscription count: 3
```

So you can confirm that you have correctly published to the `Topic` `/cmd_vel`. As you can see, the `Publisher` count is now 1, since you are publishing to the `Topic` with the **`ros2 topic pub`** command. Also, the Subscription count has increased to 3, because now you are also subscribed to the `Topic` with the **`ros2 topic echo`** command.

- All this information can be accessed by executing the `ros2 topic info` with the argument `verbose`:

Execute in Terminal #3

In [ ]:

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

In [ ]:

```
ros2 topic info /cmd_vel --verbose
```

Terminal #3 Output

```
Type: geometry_msgs/msg/Twist

Publisher count: 1

Node name: _ros2cli_1776
Node namespace: /
```

```
Topic type: geometry_msgs/msg/Twist
Endpoint type: PUBLISHER
[IRRELEVANT DATA at this level]

Subscription count: 3

Node name: four_diff_controller
Node namespace: /
Topic type: geometry_msgs/msg/Twist
Endpoint type: SUBSCRIPTION
[IRRELEVANT DATA at this level]

Node name: four_diff_controller2
Node namespace: /
Topic type: geometry_msgs/msg/Twist
Endpoint type: SUBSCRIPTION
[IRRELEVANT DATA at this level]

Node name: _ros2cli_1682
Node namespace: /
Topic type: geometry_msgs/msg/Twist
Endpoint type: SUBSCRIPTION
[IRRELEVANT DATA at this level]
```

- We have removed the extra information that is not relevant for now.
- You can see that there are **three** `topic subscribers` and **one** `topic publisher`.
  - `publisher`: `_ros2cli_1776`, which is a node name automatically assigned by the ROS2 `ros2 topic pub` command. This corresponds to your **publishing command**.
  - `subscribers`: `four_diff_controller`, `four_diff_controller2` and `_ros2cli_1682` (which corresponds to the `ros2 topic echo` command). The other two are the nodes of the **Mars Rover** in charge of **MOVING THE WHEEL MOTORS** in the simulation.

- You might have noticed that despite stopping the `ros2 topic pub` command, the robot **STILL MOVES**
- To stop the robot, use the following command:

Execute in Terminal #1

In [ ]:

```
ros2 topic pub --once /cmd_vel geometry_msgs/msg/Twist "{
```

- TIP -

- By now, you might have guessed that the command `ros2 run teleop_twist_keyboard teleop_twist_keyboard` is **PUBLISHING** in the `topic /cmd_vel` also.
- It's a very easy way to stop the Mars Rover because you only need to press the keyboard key `k`.
- Execute the following comands and see it for yourself:

Execute in Terminal #1

In [ ]:

```
ros2 run teleop_twist_keyboard teleop_twist_keyboard
```

Execute in Terminal #2

In [ ]:

```
ros2 topic echo /cmd_vel
```

Terminal #2 Output

- Depending on which keys you press, these values will be different:

```
linear:
  x: 0.0
  y: 0.0
  z: 0.0
angular:
  x: 0.0
  y: 0.0
  z: 1.0
---
linear:
  x: 0.5
  y: 0.0
  z: 0.0
angular:
  x: 0.0
  y: 0.0
  z: 0.0
---
linear:
  x: 0.0
  y: 0.0
  z: 0.0
angular:
  x: 0.0
  y: 0.0
  z: -1.0
---
```

Execute in Terminal #3

In [ ]:

```
ros2 topic info /cmd_vel --verbose
```

Terminal #3 Output

```
Type: geometry_msgs/msg/Twist

Publisher count: 1

Node name: teleop_twist_keyboard
Node namespace: /
Topic type: geometry_msgs/msg/Twist
Endpoint type: PUBLISHER
[IRRELEVANT DATA at this level]

Subscription count: 3

Node name: four_diff_controller
Node namespace: /
Topic type: geometry_msgs/msg/Twist
Endpoint type: SUBSCRIPTION
[IRRELEVANT DATA at this level]

Node name: four_diff_controller2
Node namespace: /
Topic type: geometry_msgs/msg/Twist
Endpoint type: SUBSCRIPTION
[IRRELEVANT DATA at this level]

Node name: _ros2cli_3180
Node namespace: /
```

```
Topic type: geometry_msgs/msg/Twist
Endpoint type: SUBSCRIPTION
[IRRELEVANT DATA at this level]
```

- Here, the **publishing node** is `teleop_twist_keyboard`, which is the name of the node launched with the command `ros2 run teleop_twist_keyboard teleop_twist_keyboard`.

- END TIP -

- End of Example 3.3 -

24/09/2024