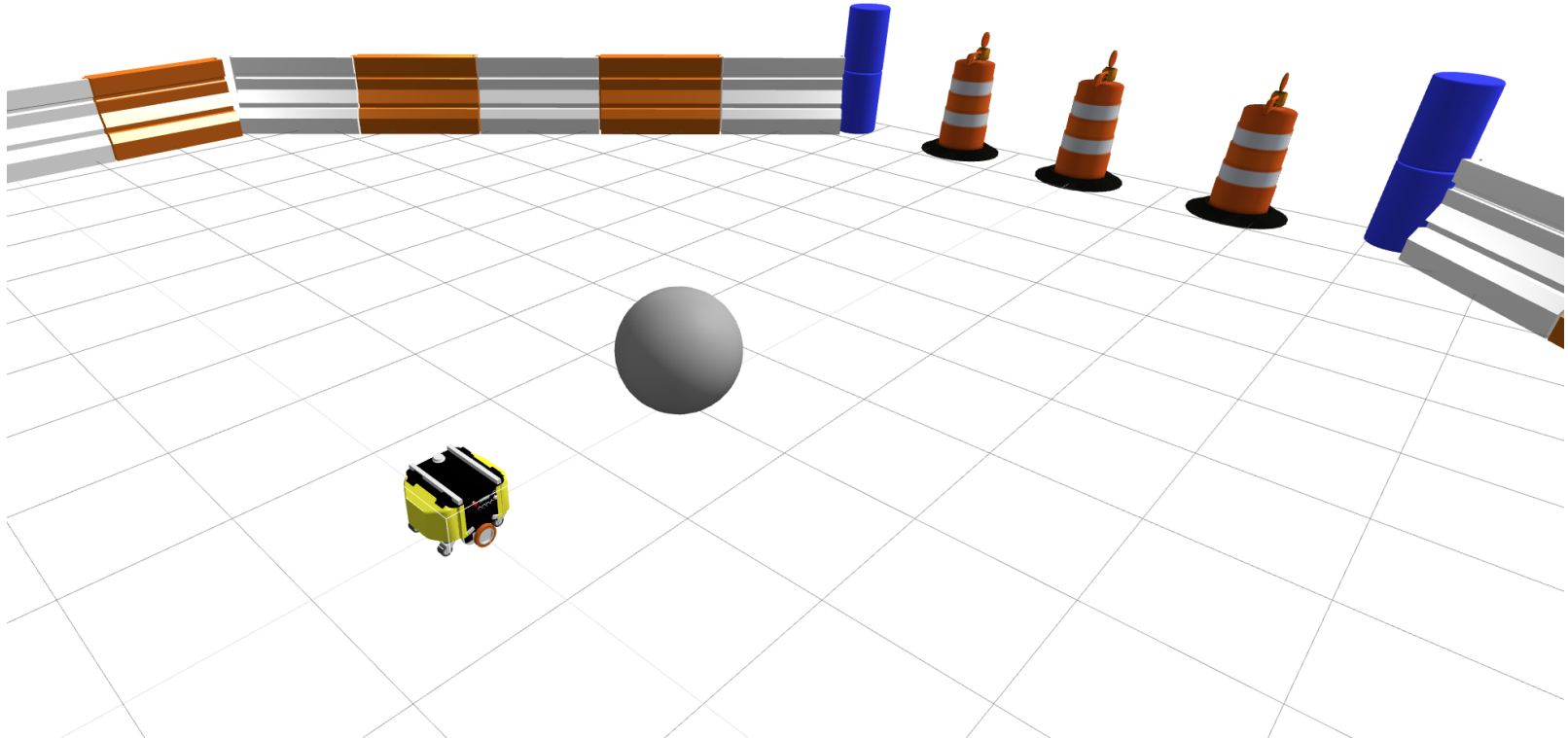


ROS2 Basics in 5 Days (Python)

Unit 7 ROS2 Debugging Tools



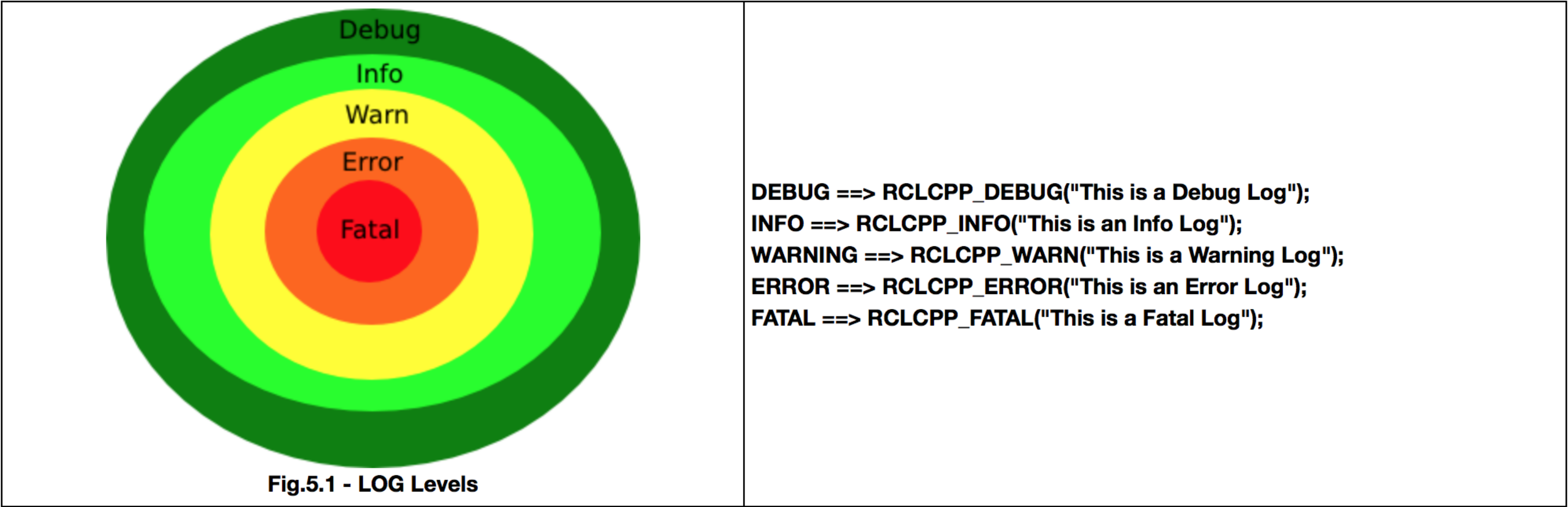
What will you learn in this unit?

- Add debugging ROS logs
- Basic use of RVIZ2 debugging tool
- view_frames tool
- ROS2 Doctor

Knowing how to transform your thoughts and experience into real tasks is one of the most complex but crucial aspects of robotics. In robotics programs, there is one constant: nothing works as it should. Since reality is far more complex, you'll need tools to figure out what's going on and where the problem lies. Therefore, debugging and visualization tools are critical in robotics, especially when dealing with complex data formats like **images, laser scans, pointclouds, or kinematic data**.

7.1 ROS2 Debugging Messages

Logs can be displayed on the screen, but they can also be saved in the ROS frame for sorting, filtering, and other uses. There are logging levels in systems, as seen in the image below. There are **five levels** in the case of ROS2 logs. Each level has a series of sub-levels. For example, if you use the **Error level**, the log will display the Error and **Fatal messages**. If your level is **Warning**, you'll see all of the messages for the levels **Warning, Error, and Fatal**.



- Exercise 7.1 -

- Create a new package named `logs_test` . When creating the package, add `rcipy` as dependencies.
- Inside the `src` folder of the package, create a new file named `logger_example.py` . Inside this file, copy the code shown below.
- Create a launch file for starting the new created node.
- Do the necessary modifications to your `setup.py` file, and compile the package.
- Execute the launch file.

```
In [ ]: import rclpy
from rclpy.node import Node

class LogDemo(Node):
    def __init__(self):
        # call super() in the constructor in order to initialize the Node object
        # the parameter we pass is the node name
        super().__init__('logger_example')
        # Logger Level configuration
        rclpy.logging.set_logger_level('logger_example', rclpy.logging.LoggingSeverity.DEBUG)
        # create a timer sending two parameters:
        # - the duration between 2 callbacks (0.2 seconds)
        # - the timer function (timer_callback)
        self.create_timer(0.2, self.timer_callback)

    def timer_callback(self):
        # print a ROS2 Log info
        self.get_logger().info("Moe yo Byakugan! Kore ga watashi no nindō yo")
        # print a ROS2 Log debugging
        self.get_logger().debug("No shinobi nearby!")
        # print a ROS2 Log warning
        self.get_logger().warn("Pain attacked the village!")
        # print a ROS2 Log error
        self.get_logger().error("I have no more chakra!")
        # print a ROS2 Log fatal
        self.get_logger().fatal("Don't die naruto-kun!")

def main(args=None):
    # initialize the ROS communication
    rclpy.init(args=args)
    # declare the node constructor
    log_demo = LogDemo()
    # pause the program execution, waits for a request to kill the node (ctrl+c)
    rclpy.spin(log_demo)
    # shutdown the ROS communication
    rclpy.shutdown()

if __name__ == '__main__':
    main()
```

You should see all of the ROS2 logs in the current nodes running in the system.

```
user:~/ros2_ws$ ros2 launch logs_test logger_example_launch_file.launch.py
[INFO] [launch]: All log files can be found below /home/user/.ros/log/2021-05-16-20-31-49-029166-1_xterm-93171
[INFO] [launch]: Default logging verbosity is set to INFO
[INFO] [logger_example-1]: process started with pid [93173]
[logger_example-1] [INFO] [1621197109.701735270] [logger_example]: Moe yo Byakugan! Kore ga watashi no nindō yo
[logger_example-1] [DEBUG] [1621197109.703387963] [logger_example]: No shinobi nearby!
[logger_example-1] [WARN] [1621197109.704526243] [logger_example]: Pain attacked the village!
[logger_example-1] [ERROR] [1621197109.705924037] [logger_example]: I have no more chakra!
[logger_example-1] [FATAL] [1621197109.706997122] [logger_example]: Don't die naruto-kun!
[logger_example-1] [INFO] [1621197109.886134159] [logger_example]: Moe yo Byakugan! Kore ga watashi no nindō yo
[logger_example-1] [DEBUG] [1621197109.886956529] [logger_example]: No shinobi nearby!
[logger_example-1] [WARN] [1621197109.887944037] [logger_example]: Pain attacked the village!
[logger_example-1] [ERROR] [1621197109.888756758] [logger_example]: I have no more chakra!
[logger_example-1] [FATAL] [1621197109.889633794] [logger_example]: Don't die naruto-kun!
[logger_example-1] [INFO] [1621197110.085986802] [logger_example]: Moe yo Byakugan! Kore ga watashi no nindō yo
[logger_example-1] [DEBUG] [1621197110.086656136] [logger_example]: No shinobi nearby!
[logger_example-1] [WARN] [1621197110.087523172] [logger_example]: Pain attacked the village!
[logger_example-1] [ERROR] [1621197110.088355377] [logger_example]: I have no more chakra!
[logger_example-1] [FATAL] [1621197110.089015405] [logger_example]: Don't die naruto-kun!
[logger_example-1] [INFO] [1621197110.289521897] [logger_example]: Moe yo Byakugan! Kore ga watashi no nindō yo
[logger_example-1] [DEBUG] [1621197110.293468900] [logger_example]: No shinobi nearby!
[logger_example-1] [WARN] [1621197110.294473191] [logger_example]: Pain attacked the village!
[logger_example-1] [ERROR] [1621197110.295040879] [logger_example]: I have no more chakra!
[logger_example-1] [FATAL] [1621197110.296763195] [logger_example]: Don't die naruto-kun!
```

- End of Exercise 7.1 -

- Exercise 7.2 -

1. Change the LOG level in the previous code **logger_example.py** and see how the different messages are printed or not, depending on the level selected.
2. Remember that you will need to recompile the package each time you modify the code.
3. The line where you change the LOG level is the following:

```
In [ ]: rclpy.logging.set_logger_level('logger_name', rclpy.logging.LoggingSeverity.<LOG_LEVEL>)
```

- End of Exercise 7.2 -

7.2 Visualize Complex Data and RVIZ2

Here you are! **HollyMolly!** You have reached the highest point in terms of debugging tools in ROS - **RVIZ2**.

RVIZ is a visualization tool that lets you view images, pointclouds, lasers, kinematic transformations, robot models, and more. The list goes on and on. It's even possible to create your own markers. It's one of the reasons ROS has been so well received. It wasn't easy to figure out what the robot was seeing before RVIZ. This is the main idea:

- **RVIZ is NOT a simulation. I repeat: It's NOT a simulation.**
- RVIZ represents what is being published in the topics by the simulation or the real robot.
- RVIZ is a complex tool, and it would take you a whole course to master it. Here, you will get a glimpse of what it can give you.

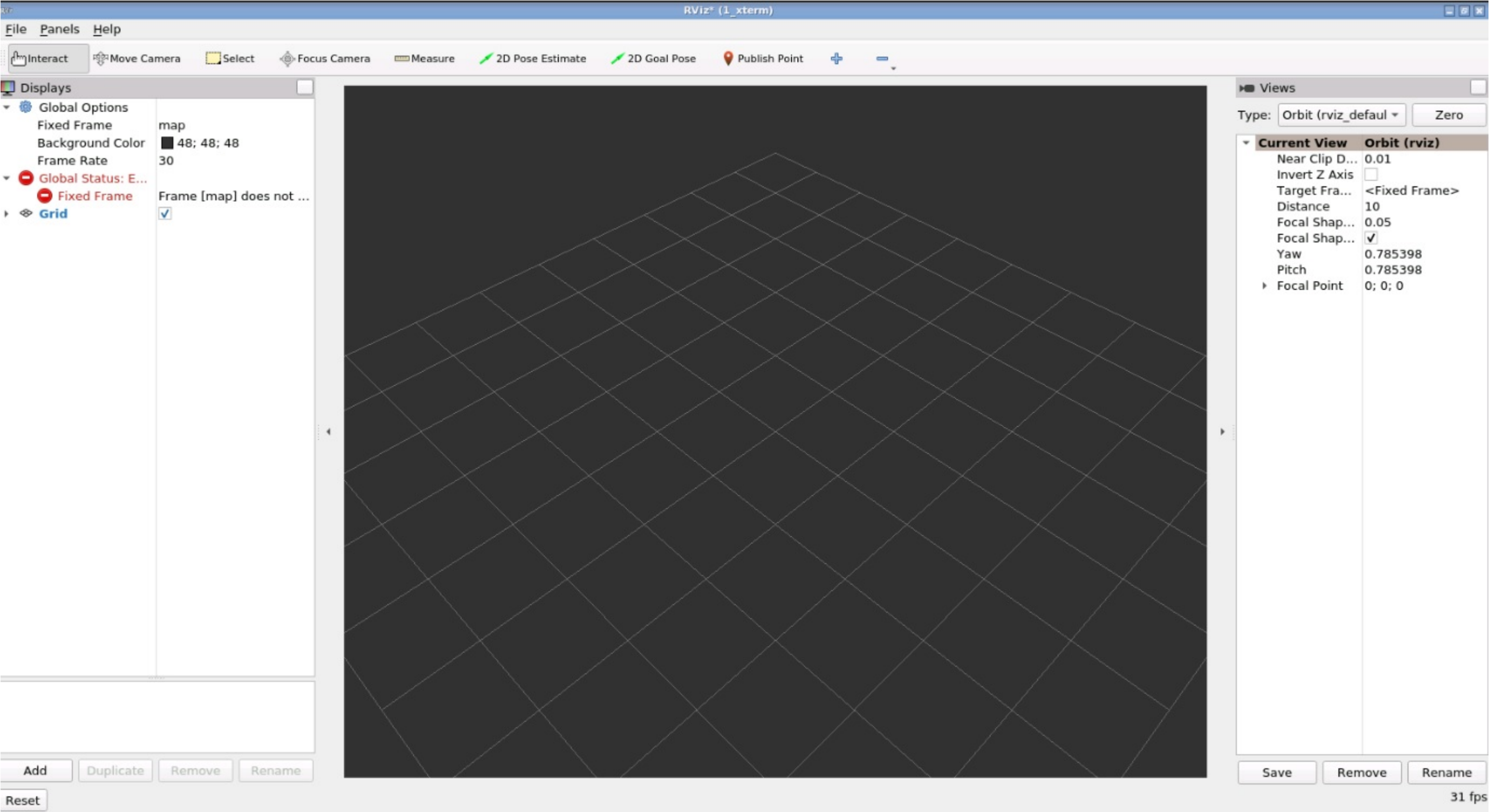
- Example 7.1 -

1. **Type the following** command into Shell #1:

In []:

rviz2

You should see a black canvas appear on top of everything else on the screen. Wait a couple of seconds until Rviz has loaded.



- Notes -

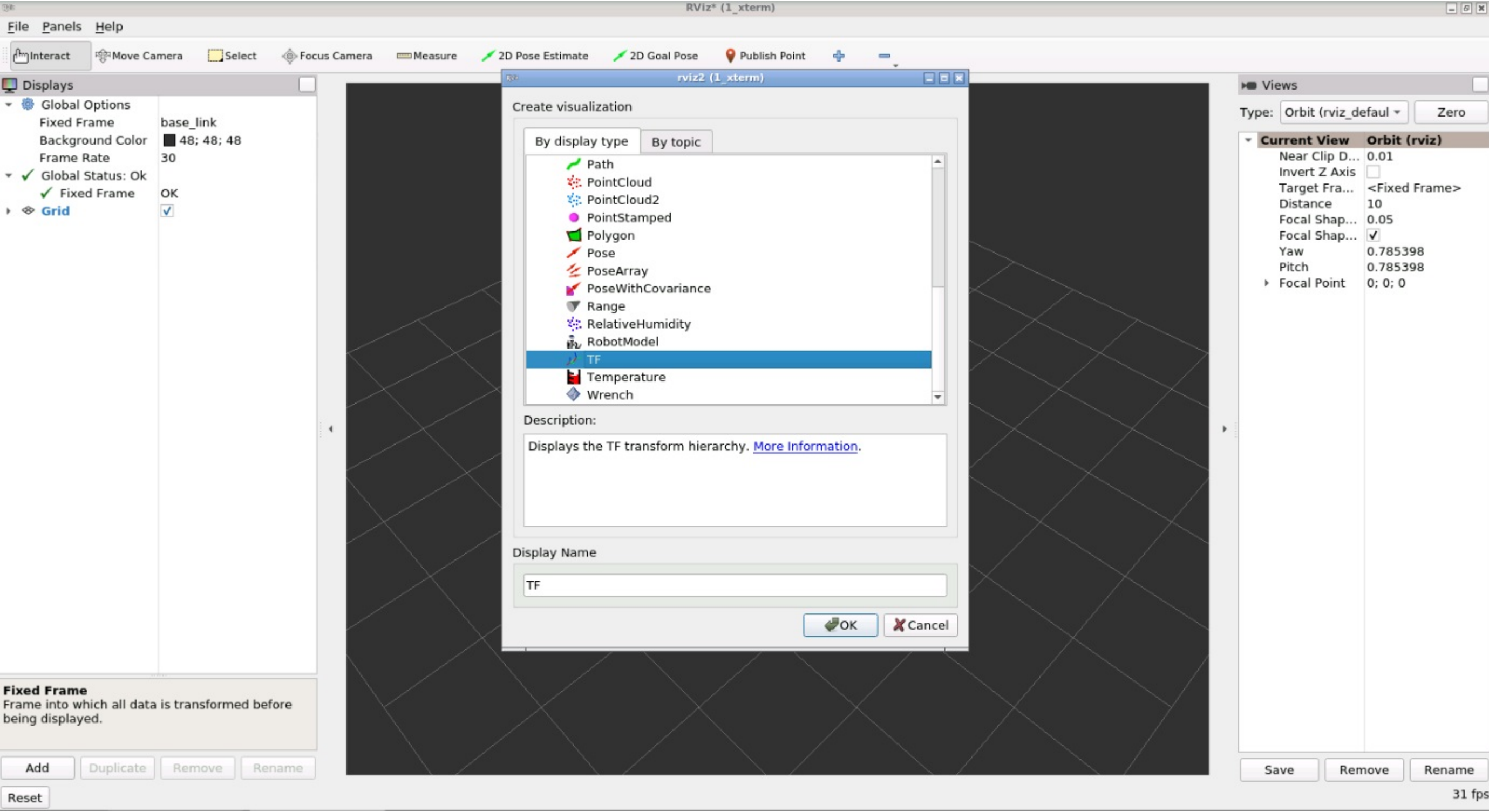
Note: Double-click at the top of the window to maximize it if you don't see the lower part of RVIZ2 (the Add button, etc.). Then you'll be able to see it.

- End of Notes -

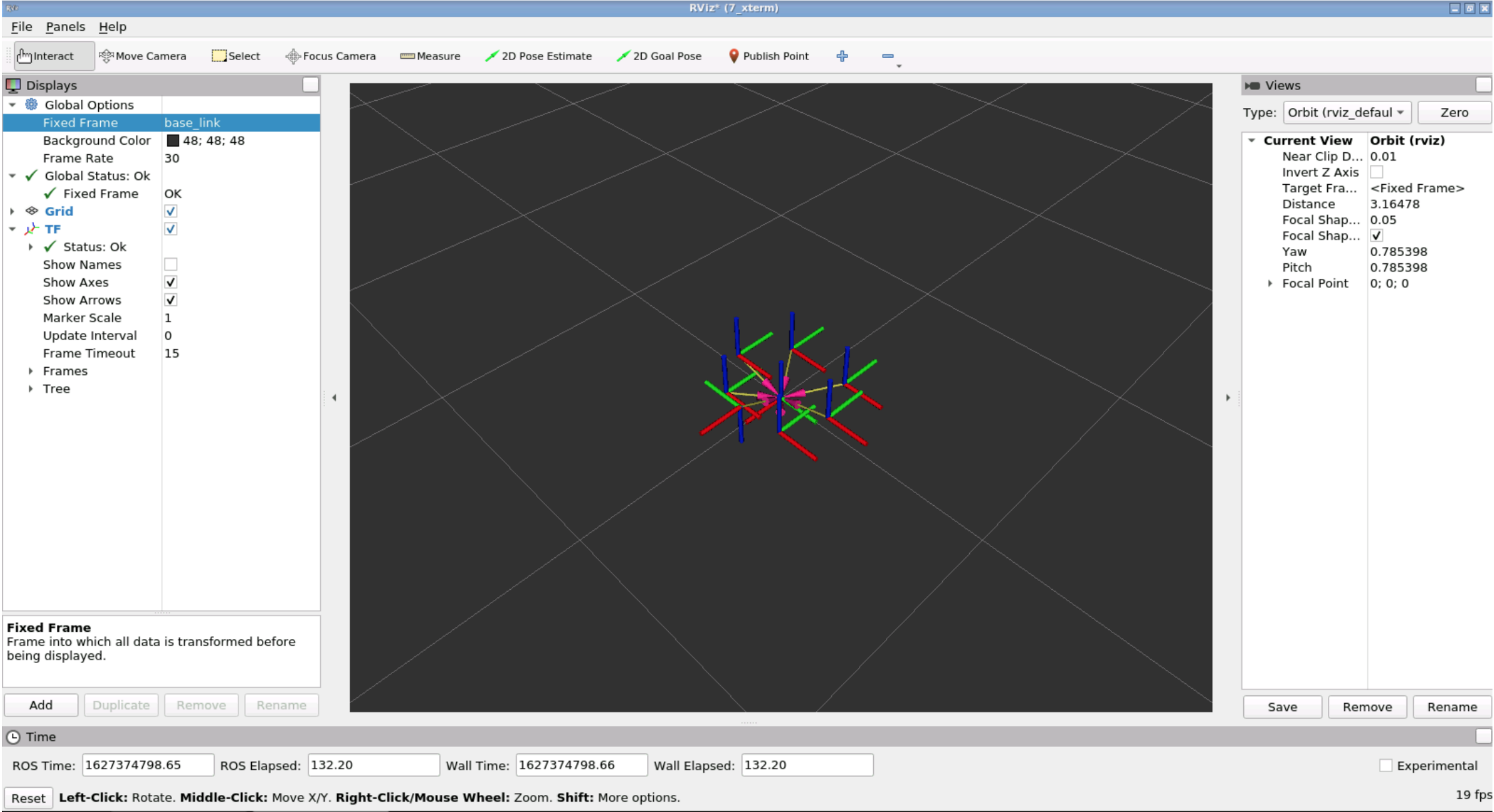
2. To begin enjoying RVIZ, consider the following factors:

- **Central Panel:** This is where the data will be shown. It's a 3D space that you can rotate (**LEFT-CLICK PRESSED**), translate (**CENTER MOUSE BUTTON PRESSED**), and zoom in/out (**LEFT-CLICK PRESSED**).
- **LEFT Displays Panel:** Here, you **manage/configure** all the elements you wish to visualize in the central panel. You only need to use two elements:
 - **In Global Options**, you must choose the Fixed Frame that best fits your needs for data visualization. It is the frame of reference to which all data will be compared.
 - **The Add button.** Clicking here will give you all of the types of elements that can be represented in RVIZ.

3. Go to RVIZ2 and add a TF element. Click "Add" and select the element TF in the list of elements provided, as shown in the image below.



4. Go to the RVIZ2 left panel, under Global Options, click on the drop down for the **Fixed Frame** and select **base_link**. Also ensure that the **TF** element checkbox is checked. You should see all of the robot's reference frame X,Y,Z axes represented in the central area with the grid after a few seconds of loading.



Press "Add" and select **RobotModel**.

From the RobotModel options, set the "Description Source" to **File**.



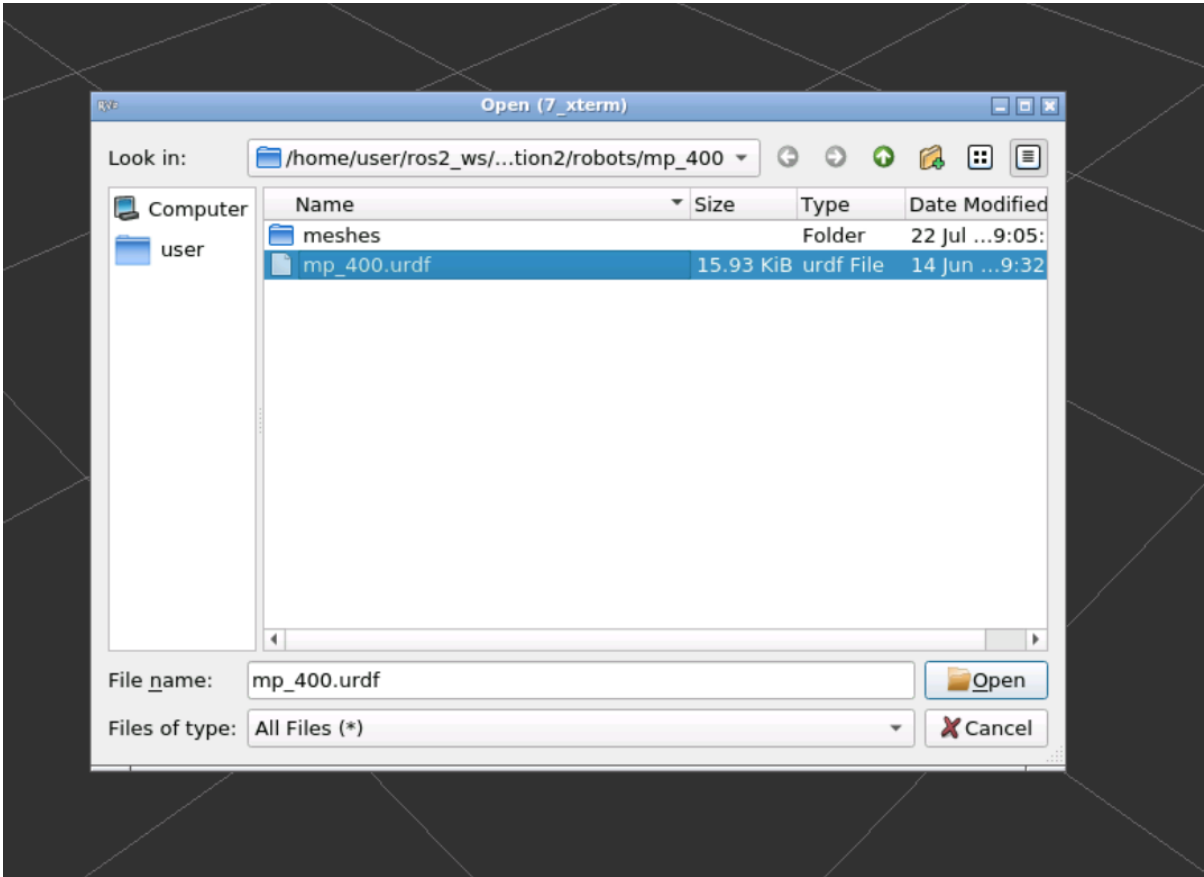
- Notes -

The path of the URDF file to load is the following:

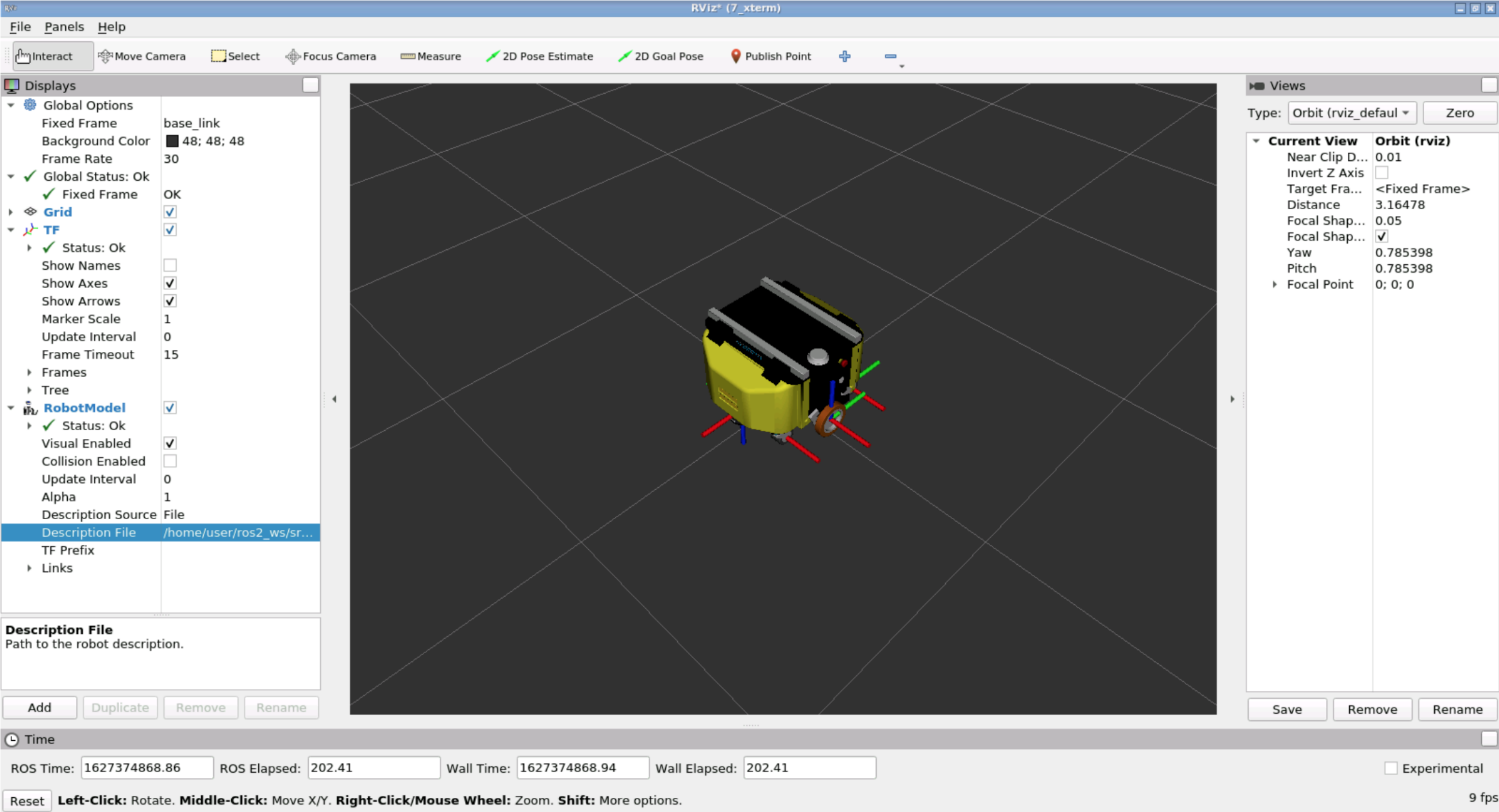
```
In [ ]: /home/simulations/ros2_sims_ws/src/neobotix_ros2/neo_simulation2/robots/mp_400
```

- Notes -

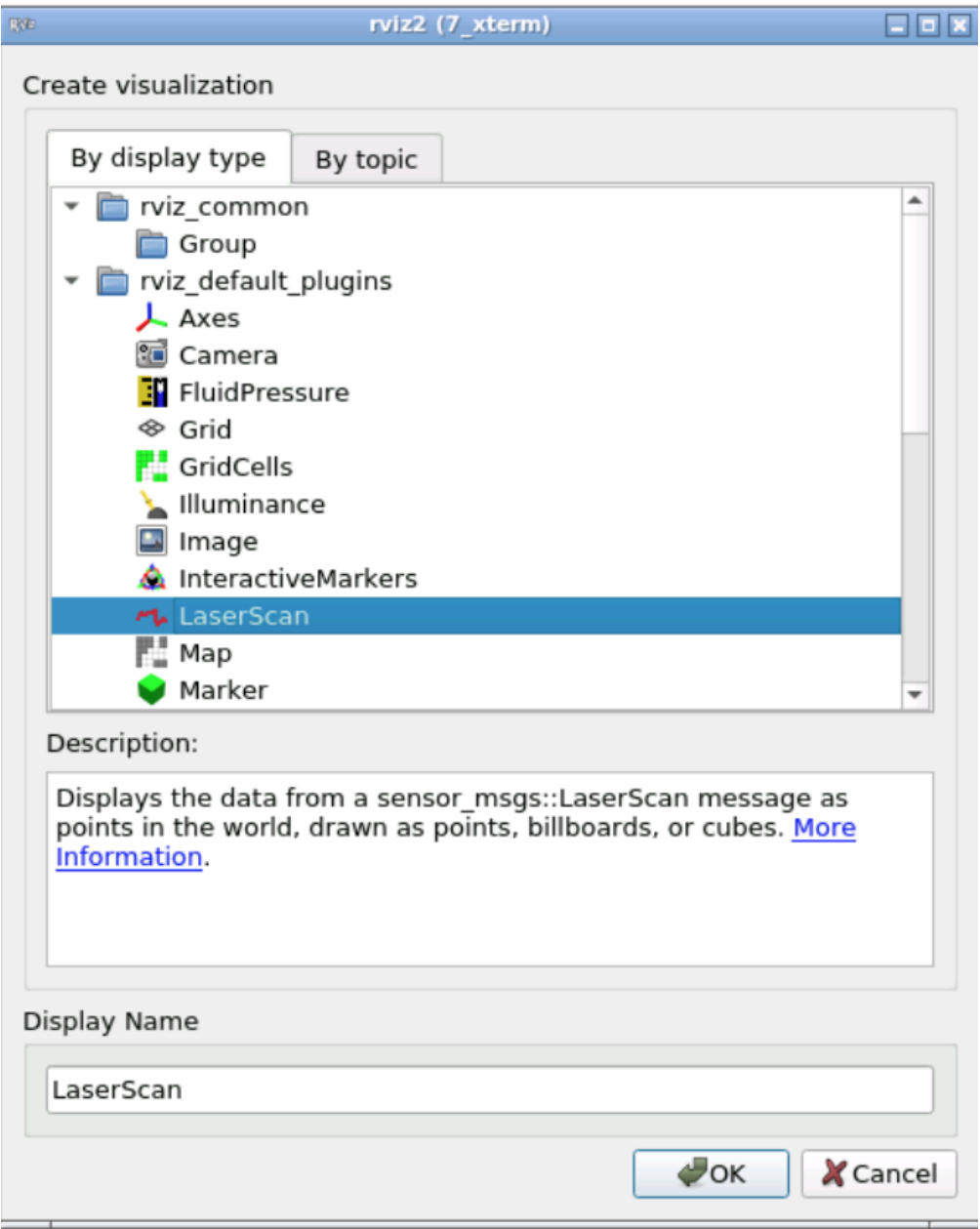
5. In the "Description Source" option, select the file named **mp_400.urdf**.



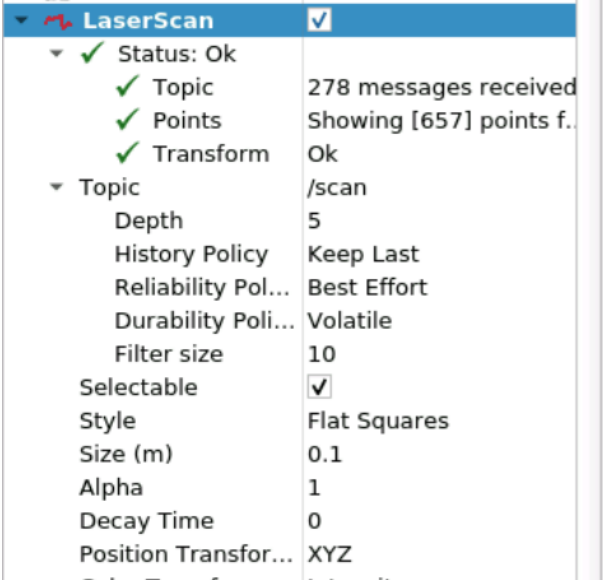
You should now see the 3D model of the robot, as shown in the image below:



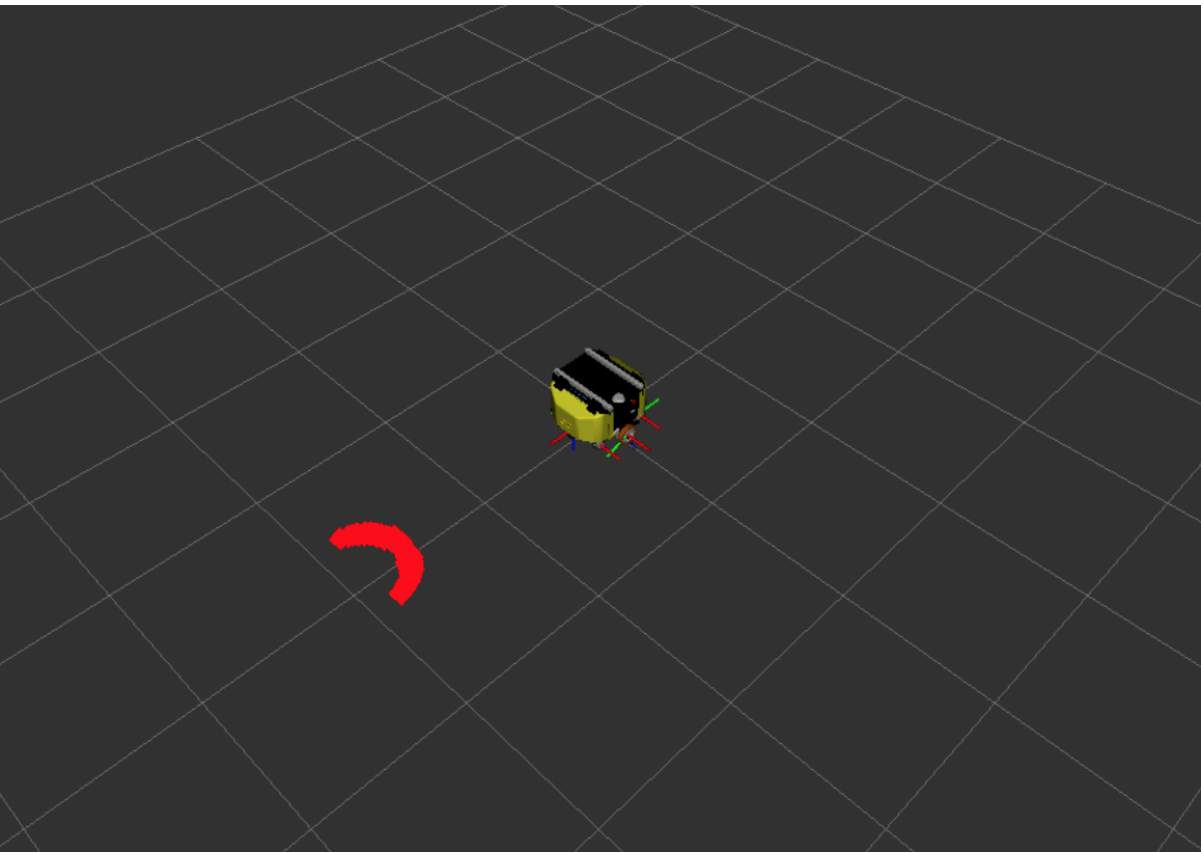
6. Let's now add a sensor to visualize it in RViz. Go to the **Add** button again and select the **LaserScan** option:



Make sure to set the Topic name to **/scan** and the Reliability Policy to **Best Effort**.



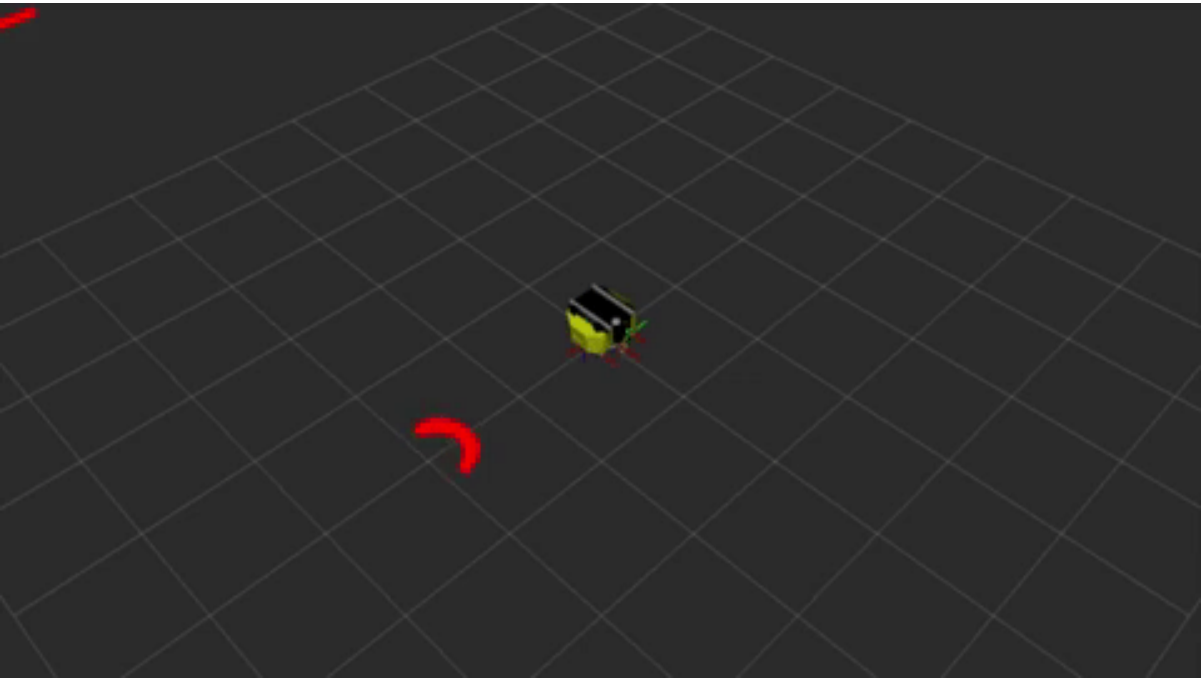
You should see something like this:



7. Go to Shell #2 and enter the command to move the robot:

```
In [ ]: ros2 topic pub /cmd_vel geometry_msgs/msg/Twist "linear:
  x: 0.2
  y: 0.0
  z: 0.0
angular:
  x: 0.0
  y: 0.0
  z: 0.2
"
```

You should see something similar to the image below in RVIZ2:



In the GIF above, you can see all of the MP-400 robot's transition elements in real-time.

- End of Example 7.1 -

7.3 tf2_tools view_frames

Another very useful tool is the **view_frames** program. This program allows you to visualize a tree diagram showing you the relationship between the different reference frames of your robot.

- Example 7.2 -

Since the program will produce a PDF file in the directory where it is run, we suggest that you first move outside of the ROS workspace:

```
In [ ]: cd
```

This command will return you to your home directory.

Now, execute the following instruction to have access to the ROS 2 commands:

```
In [ ]: source /opt/ros/humble/setup.bash
```

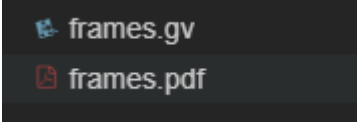
```
In [ ]: ros2 run tf2_tools view_frames
```

Type the following to list all files in the current directory:

```
In [ ]: ls
```

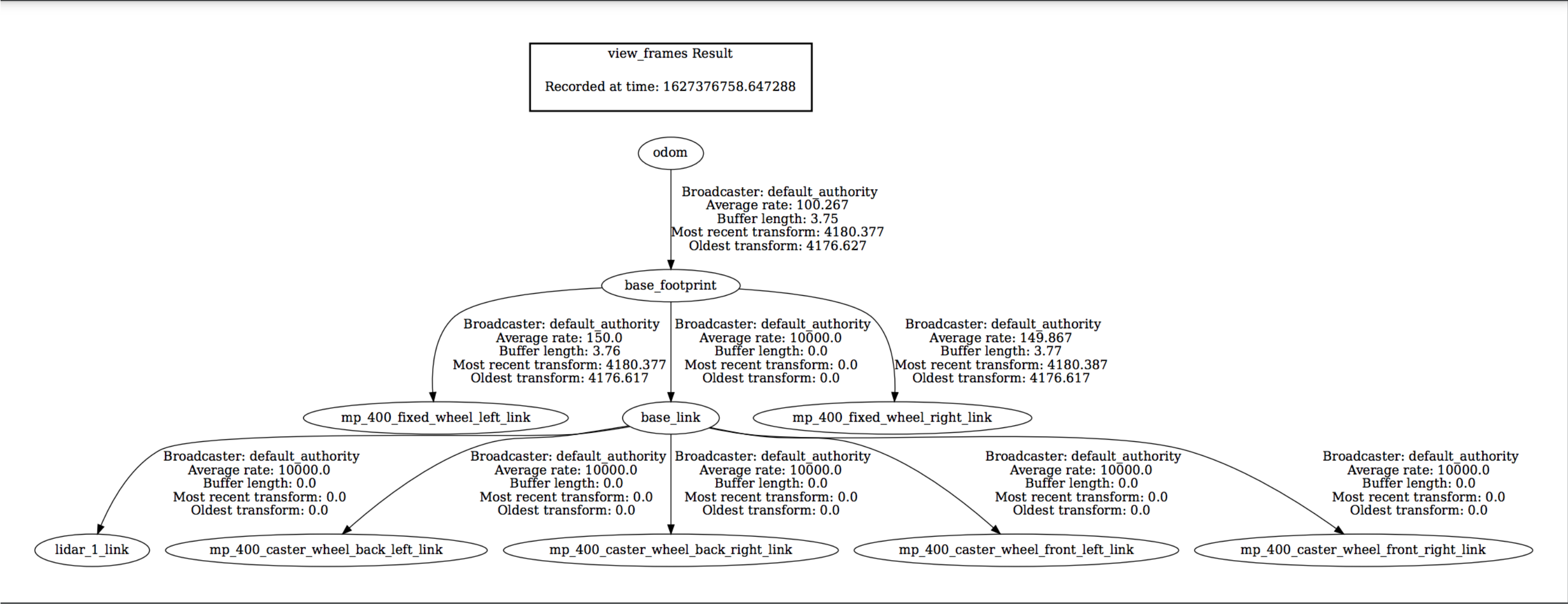
You should get:

Using the IDE, you will also see that two new files have been created like the ones you see below:



Now, download the pdf file. You can do so by performing a right-click.

When you open it, you can visualize the tree diagram mentioned earlier.



- End of Example 7.2 -

This tool is useful because it allows you to see the connection between each frame and determine if a transform is not well executed. For example, there may be no relationship between a generated map and the robot that wants to navigate, or a sensor and its base, etc.

If you are a bit confused about reference frames or TF's, don't stress yourself. Reference frames is a bit broad subject and unfortunately it is impossible to discuss all these topics here in detail. However, if you are interested in learning more about TF's, we highly recommend that you take this course: [TF ROS2 \(https://app.theconstructsim.com/#/Course/92\)](https://app.theconstructsim.com/#/Course/92)

7.4 ROS2 Doctor

What happens when your ROS2 setup is not working as expected? Here you use ROS2's great tool, `ros2 doctor`.

This powerful tool reviews the total ROS2 setup from the platform, versions, network, environment, etc. In addition to giving you a proper diagnosis with precautions, errors, and possible reasons for problems.

Don't forget that `ros2doctor` belongs to the `ros2cli` package, so you will need to have it installed to use it.

7.4.1 Check Your Entire Setup

- Example 7.3 -

This is the first and most significant level of *ros2doctor* tool checking. You only need to go to Shell #1 and execute **ros2 doctor**.

```
In [ ]: ros2 doctor
```


WOOOOOOW! I told you that is a complete tool to diagnose. If the ROS2 setup is in perfect shape, it will **pass all checks** as stated in the **last line of output** in the shell.

You can disregard the `UserWarning` messages. Don't worry about it. It means that something is not configured as a default or in the ideal way or maybe the last version. Why? Because in this platform, the configuration is designed to be the best for your learning. ;-)

This is what the general structure for a `UserWarning` looks like:

For example:

If something is wrong, what does the output look like?

Well, it depends on what is wrong. You may receive output similar to the message below:

- End of Example 7.3 -


7.4.2 Check an specific System

Can ROS2 Doctor help me with a node? OF COURSE! Let's see.

- Example 7.4 -

In []:

ros2 doctor



Look at the final output. It is telling you what is happening with some topics.

- End of Example 7.4 -

7.4.3 Get a Full Report with ROS2 Doctor

If you want to know more details to analyze issues, a `--report` is the best option.

- Example 7.5 -

To get a full report, run the following command in Shell #1:

In []:

ros2 doctor --report



You will see that the report is categorized into 7 sections:

- NETWORK CONFIGURATION
- PACKAGE VERSION
- PLATFORM CONFIGURATION
- QOS COMPATIBILITY LIST
- RMW MIDDLEWARE
- ROS 2 INFORMATION
- TOPIC LIST

Let's review each category:

NETWORK CONFIGURATION

You can see all the information related to the system's network configuration.

PACKAGE VERSION

These are packages that can be updated to the last version.

PLATFORM CONFIGURATION

This is information about the platform you are using.

QOS COMPATIBILITY LIST

This is information about the QoS setting of your system.

RMW MIDDLEWARE

This is information about the RMW MIDDLEWARE that defines an interface of middleware primitives used by the higher level ROS API's. If you want to know more about this, you can follow this [live_class](https://app.theconstructsim.com/#/LiveClass/a9fcaaa4-64a1-4e15-b92b-d11076325bb2) (<https://app.theconstructsim.com/#/LiveClass/a9fcaaa4-64a1-4e15-b92b-d11076325bb2>).

ROS2 INFORMATION

This is information about the ROS2 that you are using.

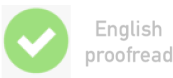
TOPIC LIST

This is information about the topics that are working with the subscribers and publishers.

- End of Example 7.5 -

You did it!

You made it to the end of this course!



English proofread