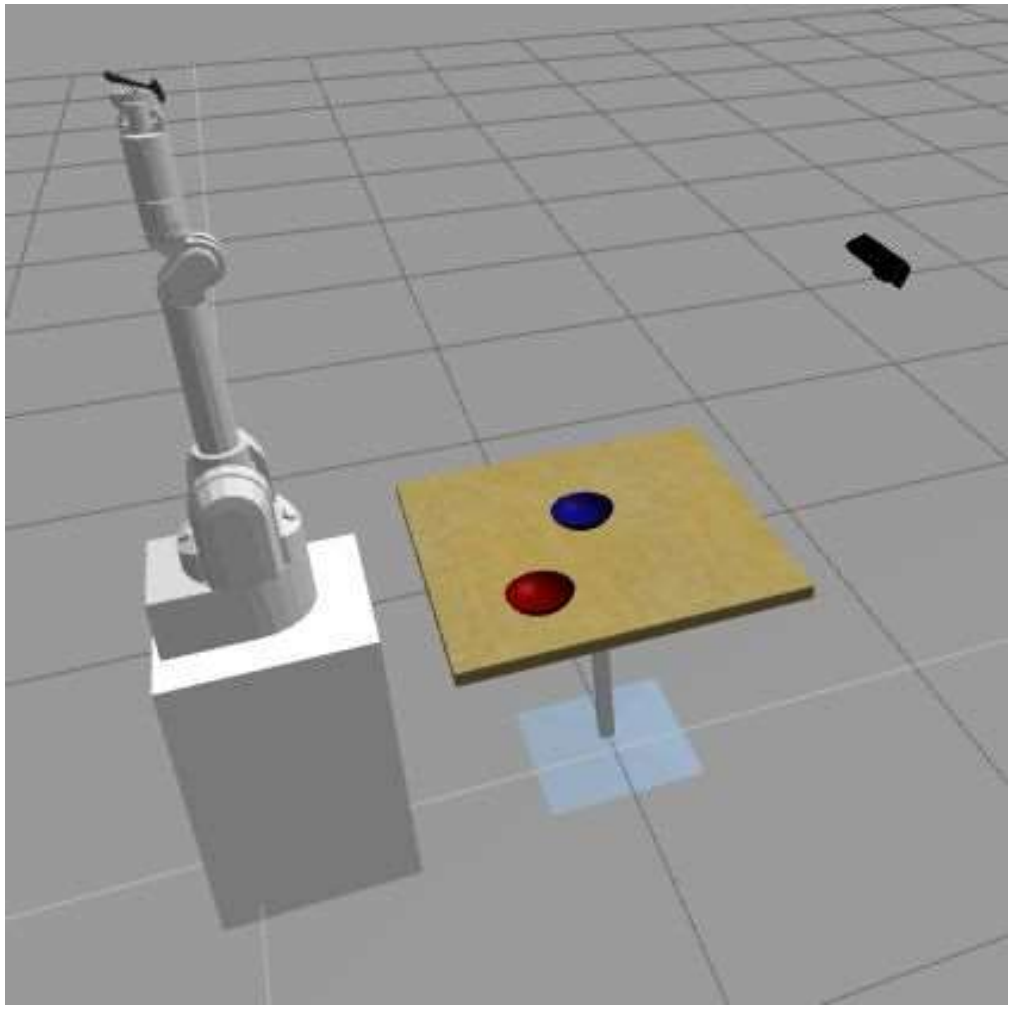
ROS Basics in 5 days

Unit 10　Debugging Tools



- Summary -

Estimated time of completion: 1.5 hours

What will you learn with this unit?

- Add Debugging ROS logs
- Filter ROS logs
- Record and replay sensory data
- Plot Topic Data
- Draw connections between different nodes of your system
- Basic use of RViz debugging tool

- End of Summary -

One of the most difficult, but important, parts of robotics is: **knowing how to turn your ideas and knowledge into real projects**. There is a constant in robotics projects: **nothing works as in theory**. Reality is much more complex and, therefore, you need tools to discover what is going on and find where the problem is. That's why debugging and visualization tools are essential in robotics, especially when working with complex data formats such as **images**, **laser-scans**, **pointclouds** or **kinematic data**.

Examples are shown in {Fig-10.i} and {Fig-10.ii}.

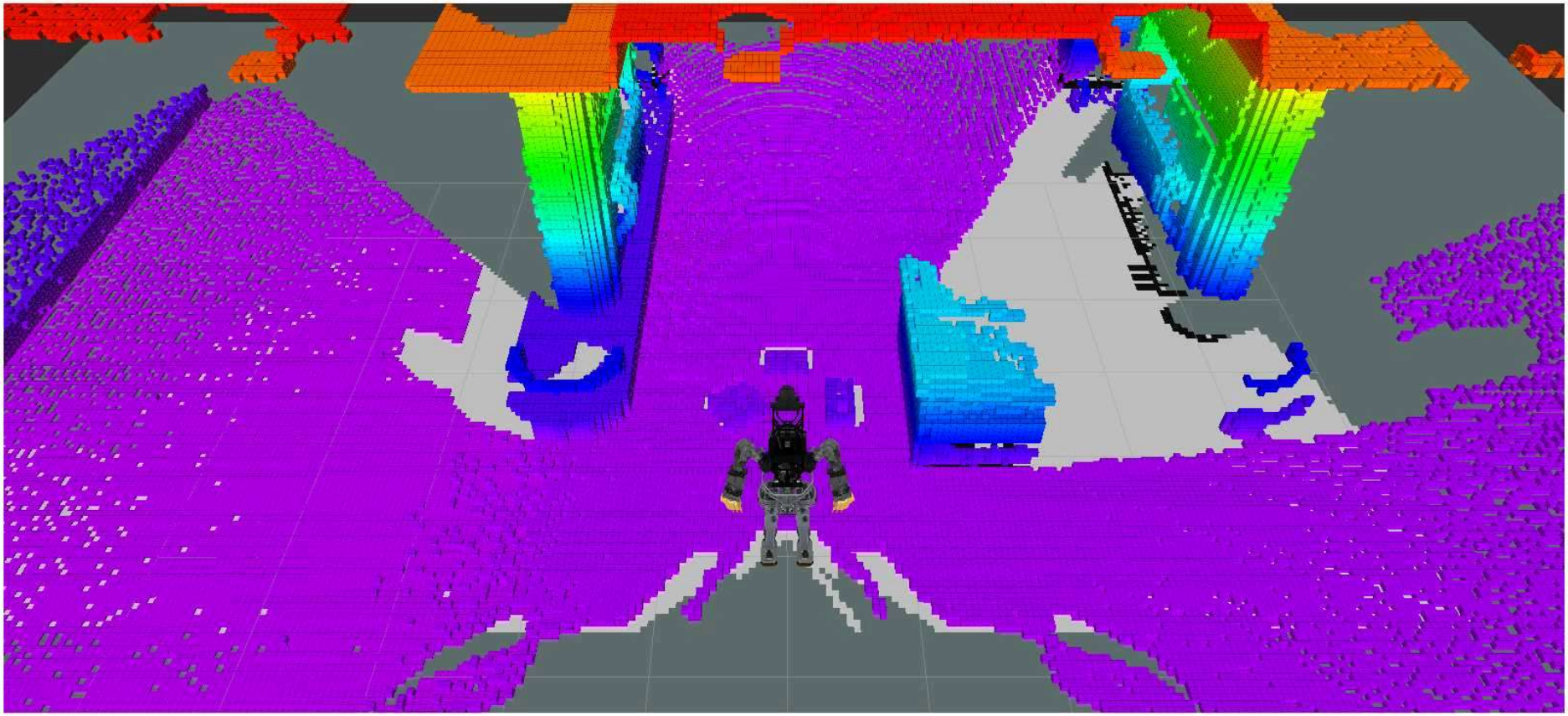


Fig.10.i - Atlas Laser

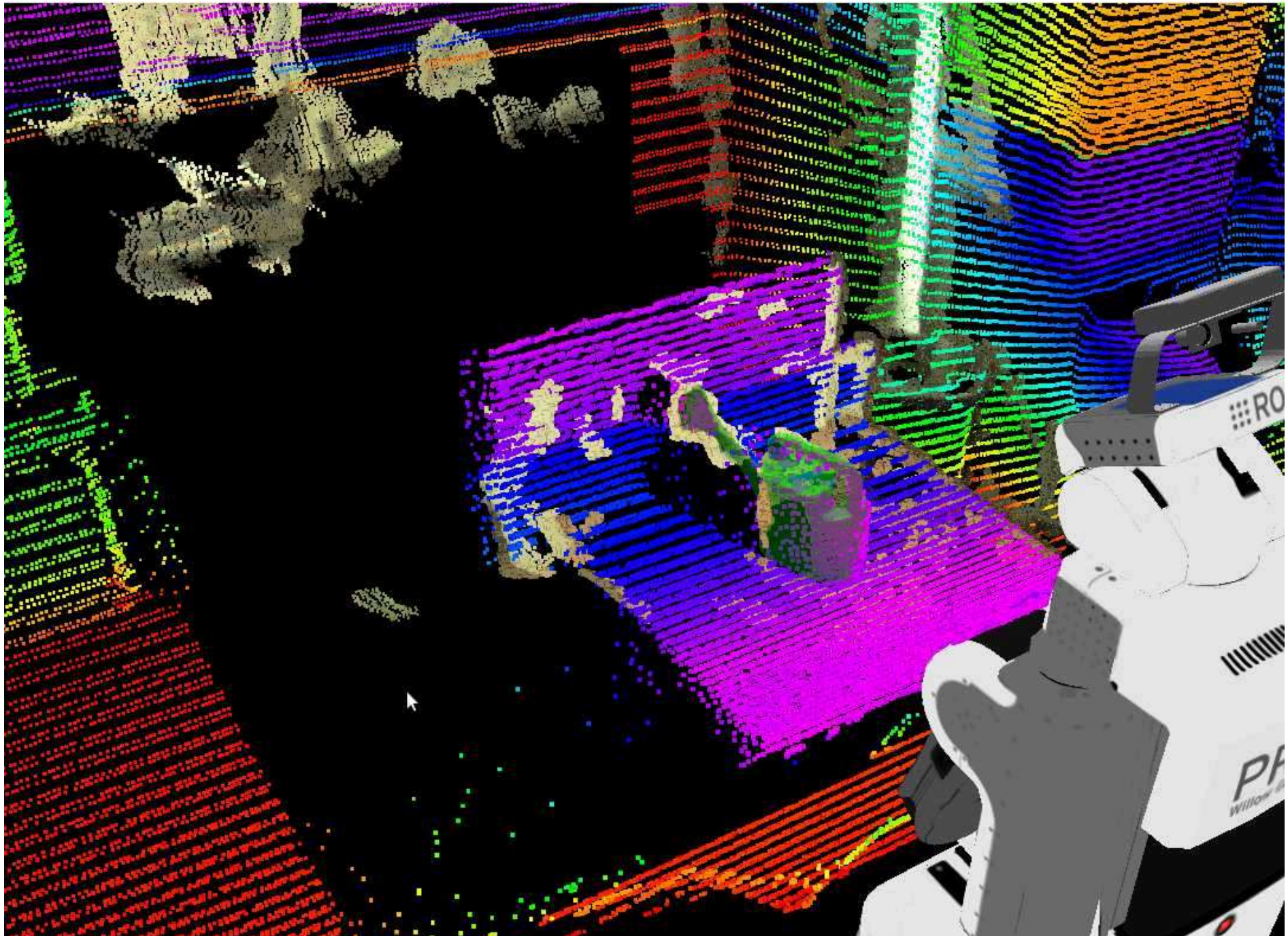


Fig.10.ii - PR2 Laser and PointCloud

So here you will be presented with the most important tools for debugging your code and visualizing what is really happening in your robot system.

10.1 ROS Debugging Messages and Rqt-Console

You have used *print()* during this course to print information about how your programs are doing. **Prints** are the *Dark Side of the Force*, so from now on, you will use a more *Jedi* way of doing things. **LOGS** are the way. Logs allow you to print them on the screen, but also to store them in the ROS framework, so you can classify, sort, filter, or else.

In logging systems, there are always levels of logging, as shown in {Fig-10.1}. In ROS logs case, there are **five** levels. Each level includes deeper levels. So, for example, if you use **Error** level, all the messages for **Error** and **Fatal** will be shown. If your level is **Warning**, then all the messages for levels **Warning**, **Error** and **Fatal** will be shown.

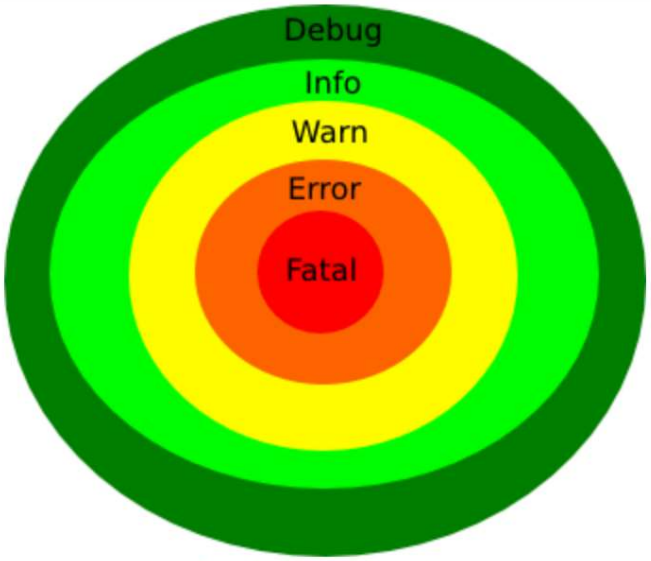


Fig.5.1 - LOG Levels

Use the Python module *rospy* to access the logging functionality in Python.
DEBUG ==> rospy.logdebug(msg, *args)
***INFO* ==> *rospy.loginfo(msg,* args)**
WARNING ==> rospy.logwarn(msg, *args)
***ERROR* ==> *rospy.logerr(msg,* args)**
FATAL ==> rospy.logfatal(msg, *args)

- Example 10.1 -

First, create the package **my_log_print_example**, executing the following comands in the Web Shell:

▶ Execute in Shell #1

In [ ]:
```
cd ~/catkin_ws/src
```

In [ ]:
```
catkin_create_pkg my_log_print_example rospy std_msgs
```

Now create the **logger_example.py** script inside it:

▶ Execute in Shell #1

In [ ]:
```
cd ~/catkin_ws/src/my_log_print_example
```

In [ ]:
```
mkdir scripts
```

In [ ]:
```
cd scripts
```

In [ ]:
```
touch logger_example.py
```

In [ ]:
```
chmod +x logger_example.py
```

Python Program {10.1}: logger_example.py

In [ ]:

```
#! /usr/bin/env python

import rospy
import random
import time

# Options: DEBUG, INFO, WARN, ERROR, FATAL
rospy.init_node('log_demo', log_level=rospy.DEBUG)
rate = rospy.Rate(0.5)

#rospy.loginfo_throttle(120, "DeathStars Minute info: "+str(time.time()))

while not rospy.is_shutdown():
    rospy.logdebug("There is a missing droid")
    rospy.loginfo("The Emperors Capuchino is done")
    rospy.logwarn("The Revels are coming time "+str(time.time()))
    exhaust_number = random.randint(1,100)
    port_number = random.randint(1,100)
    rospy.logerr(" The thermal exhaust port %s, right below the main port %s", exhaust_number, port_number)
    rospy.logfatal("The DeathStar Is EXPLODING")
    rate.sleep()
    rospy.logfatal("END")
```

END Python Program {10.1}: logger_example.py

▶ Execute in Shell #1

In [ ]:

```
rosrun my_log_print_example logger_example.py
```

A good place to read all of the logs issued by all of the ROS Systems is: `/rosout`

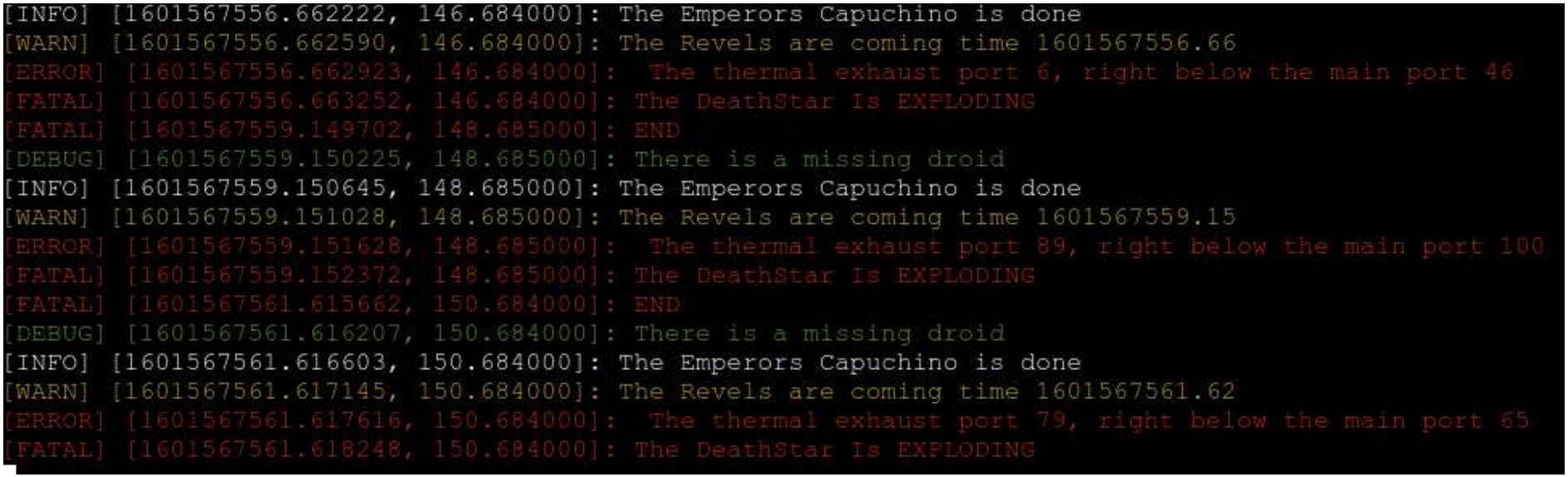
Go to the Terminal and type the following command:

▶ Execute in Shell #2

In [ ]:
```
rostopic echo /rosout
```

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

But also you can just read them in the output of the script when launched:



- End of Example 10.1 -

- Exercise 10.1 -

1. Inside the newly created package **my_log_print_example**, create a launch file named **start_logger_example.launch** that starts the Python script introduced above logger_example.py.

1. Change the LOG level in the `rospy.init_node()` and see how the different messages are printed or not in the **/rosout** topic, depending on the level selected.

- End of Exercise 10.1 -

As you can see, with only one node publishing every 2 seconds, the amount of data is big. Now imagine **ten** nodes, publishing image data, laser data, using the actions, services, and publishing debug data of your DeepLearning node. It's really difficult to get the logging data that you want.

That's where **rqt_console** comes to the rescue.

Type in the Terminal #1 the roslaunch command for launching the Exercise 10.1 launch.

▶ Execute in Shell #1

In [ ]:
```
roslaunch my_log_print_example start_logger_example.launch
```

And type in the Terminal #2 : `rqt_console` , to activate the GUI log printer.

▶ Execute in Shell #2

In [ ]:
```
rqt_console
```

You will get a window similar to {Fig-10.2} in the black canvas that should pop up.

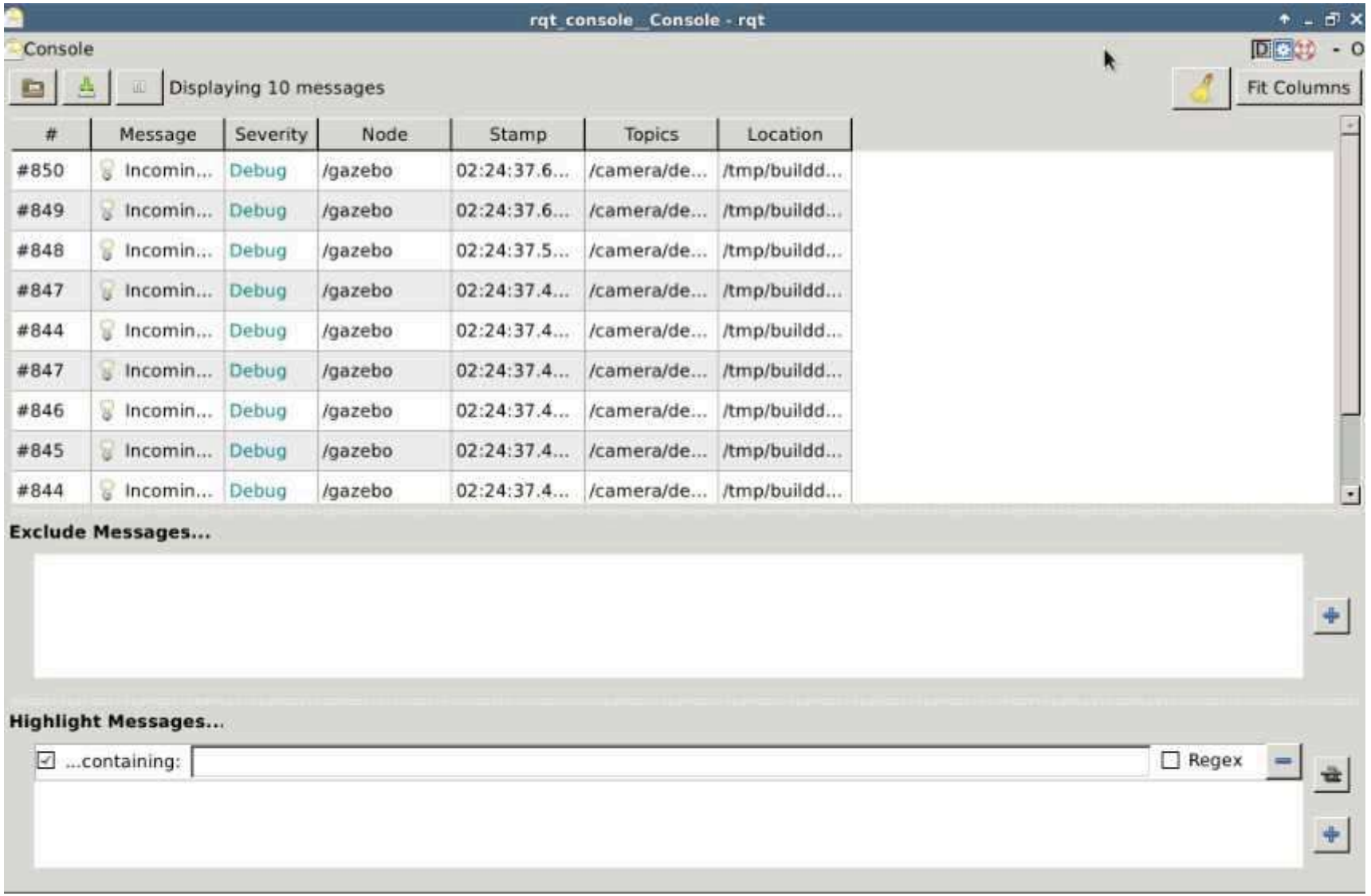


Fig.10.2 - Rqt Console

The rqt_console window is divided into three subpanels.

- The first panel outputs the logs. It has data about the message, severity/level, the node generating that message, and other data. Is here where you will extract all your logs data.
- The second one allows you to filter the messages issued on the first panel, excluding them based on criteria such as: node, severity level, or that it contains a certain word. To add a filter, just press the plus sign and select the desired one.
- The third panel allows you to highlight certain messages, while showing the other ones.

You have to also know that clicking on the tiny white gear on the right top corner, you can change the number of messages shown. Try to keep it as low as possible to avoid performance impact in your system.

Filter the logs so that you only see the Warning and Fatal messages of the node from exercice 10.1.

You should see something like {Fig-10.3}:

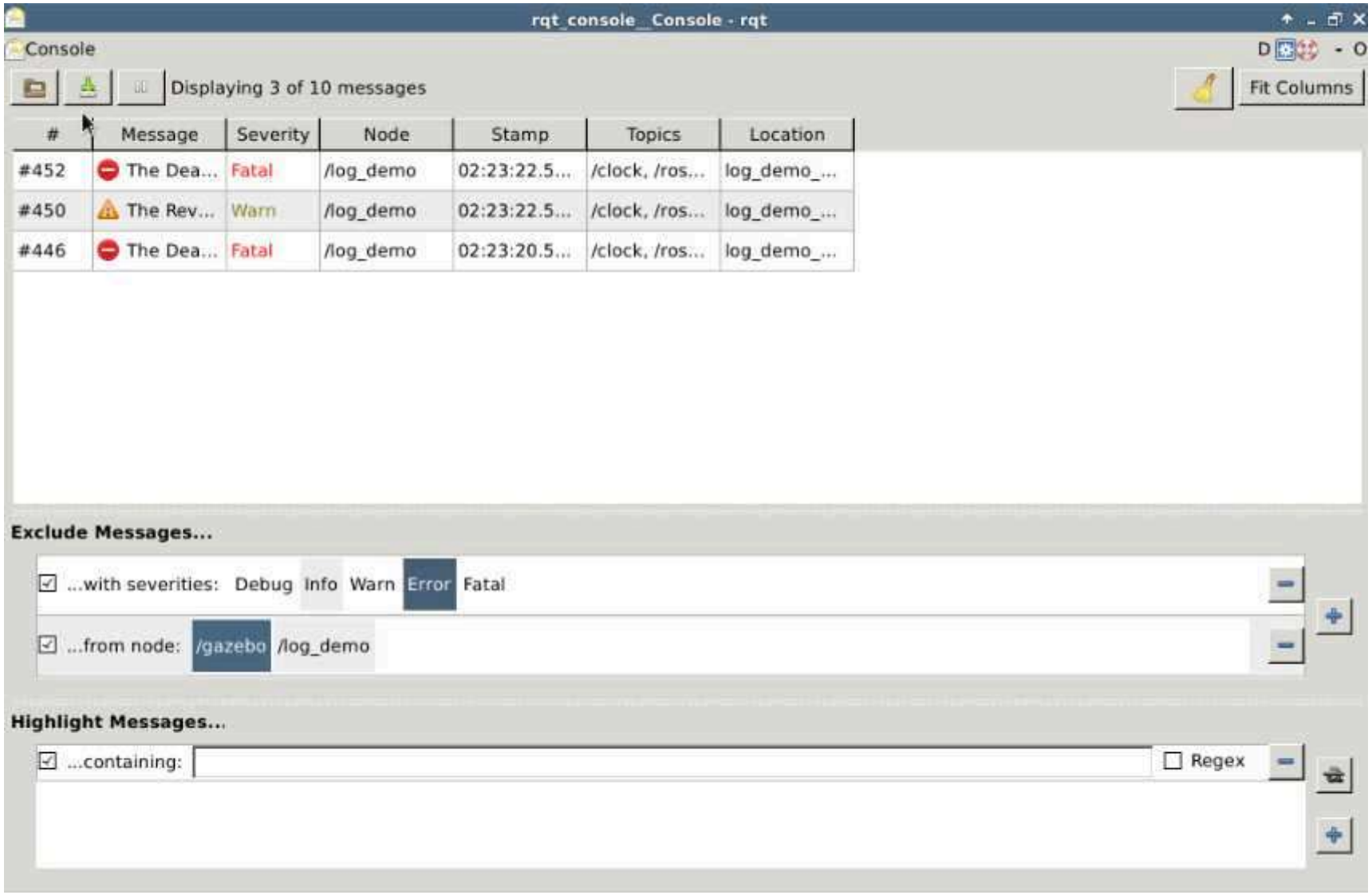


Fig.10.3 - Rqt Console Filter

10.2 Plot topic data and Rqt Plot

This is a very common need in any scientific discipline, but especially important in robotics. You need to know if your inclination is correct, your speed is the right one, the torque readings in an arm joint is above normal, or the laser is having anomalous readings. For all these types of data, you need a graphic tool that makes some sense of all the data you are receiving in a fast and real-time way. Here is where **rqt_plot** comes in handy.

To Continue you should have stopped the Exercise 10.1 node launched in the Terminal #1

- **Go to the Terminal and type the following command to start moving the robot arm:**

▶ Execute in Shell #1

In [ ]:
```
roslaunch iri_wam_aff_demo start_demo.launch
```

- **Go to another Terminal and type the following command to see the positions and the effort made by each joint of the robot arm:**

▶ Execute in Shell #2

In [ ]:
```
rostopic echo /joint_states -n1
```

As you can probably see, knowing what's happening in the robots joints with only arrays of numbers is quite daunting.

So let's use the ***rqt_plot*** command to plot the robot joints array of data.

Go to the graphical interface and type in a terminal the following command to open the rqt_plot GUI:

Remember to hit [CTRL]+[C] to stop the rostopic echo

▶ Execute in Shell #2

In [ ]:
```
rqt_plot
```

You will get a window similar to {Fig-10.4}:

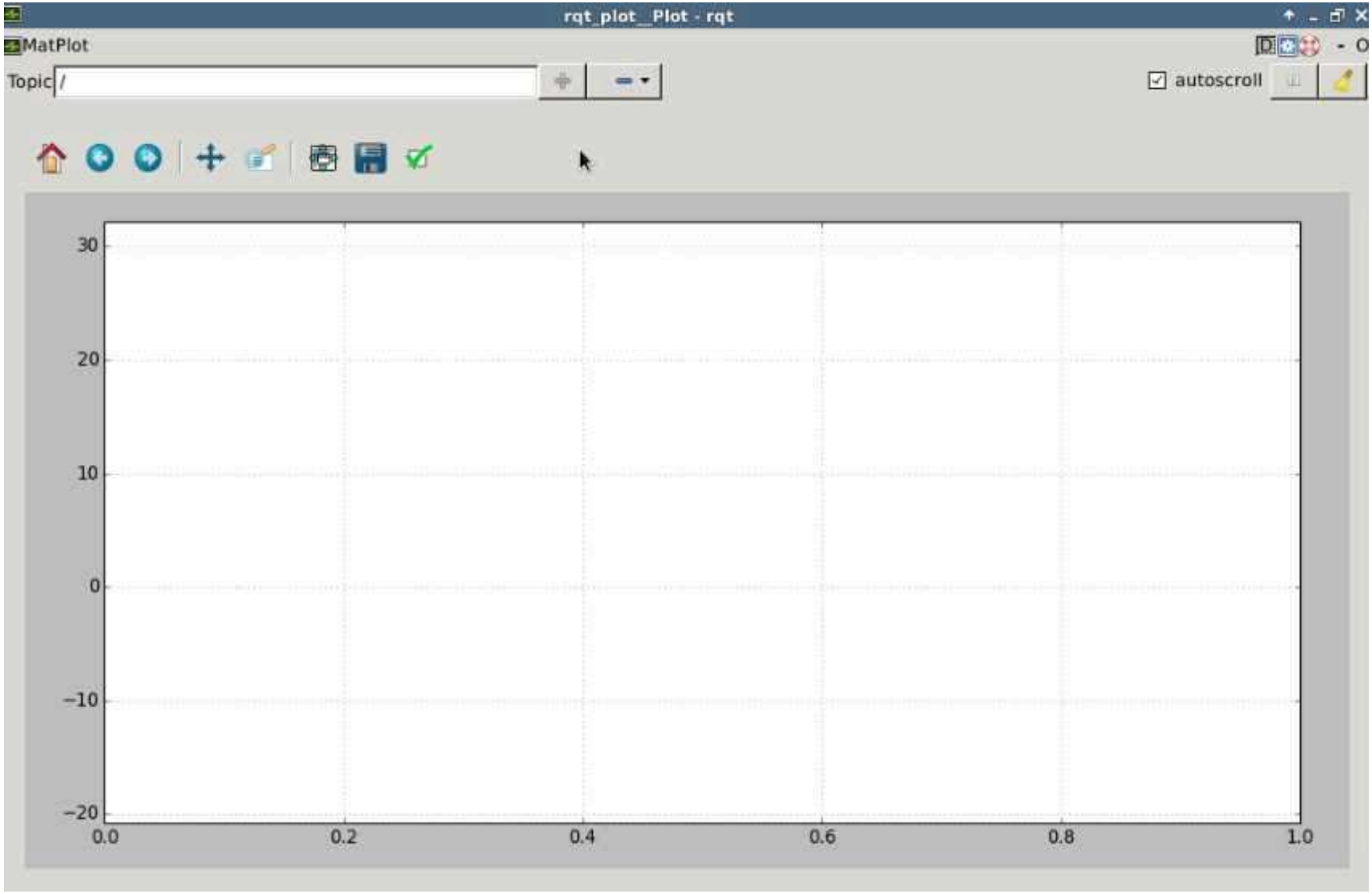


Fig.10.4 - Rqt Plot

In the *topic* input located in the top-left corner of the window, you have to write the topic structure that leads to the data that you want to plot. Bear in mind that in order to be plotted, the topic has to publish a number. Once written, you press the ***PLUS SIGN*** to start plotting the Topic.

In the case that we want to plot the robot joints, we need to plot the topic */joint_states*, which has the following structure (that you can already get by extracting the topic message type with *rostopic info* , and afterwards using *rosmsg show* command from **Unit 3**):

In [ ]:
```
std_msgs/Header header
string[] name
float64[] position
float64[] velocity
float64[] effort
```

Then, to plot the velocity of the first joint of the robot, we would have to type */joint_states/velocity[0]*.

You can add as many plots as you want by pressing the "plus" button.

- Exercise 10.2 -

- Plot in rqt_plot the effort made by the four first joints of the robot while it moves.

- End of Exercise 10.2 -

10.3 Node Connections and Rqt graph

Is your node connected to the right place? Why are you not receiving data from a topic? These questions are quite normal as you might have experienced already with ROS systems. **Rqt_graph** can help you figure that out in an easier way. It displays a visual graph of the nodes running in ROS and their topic connections. It's important to point out that it seems to have problems with connections that aren't topics.

Go to the graphical interface and type in a terminal the following command to open the rqt_graph GUI:

Remember to have in the Terminal #1 the roslaunch iri_wam_aff_demo start_demo.launch

▶ Execute in Shell #2

In [ ]:
```
rqt_graph
```

You will get something like the following image:

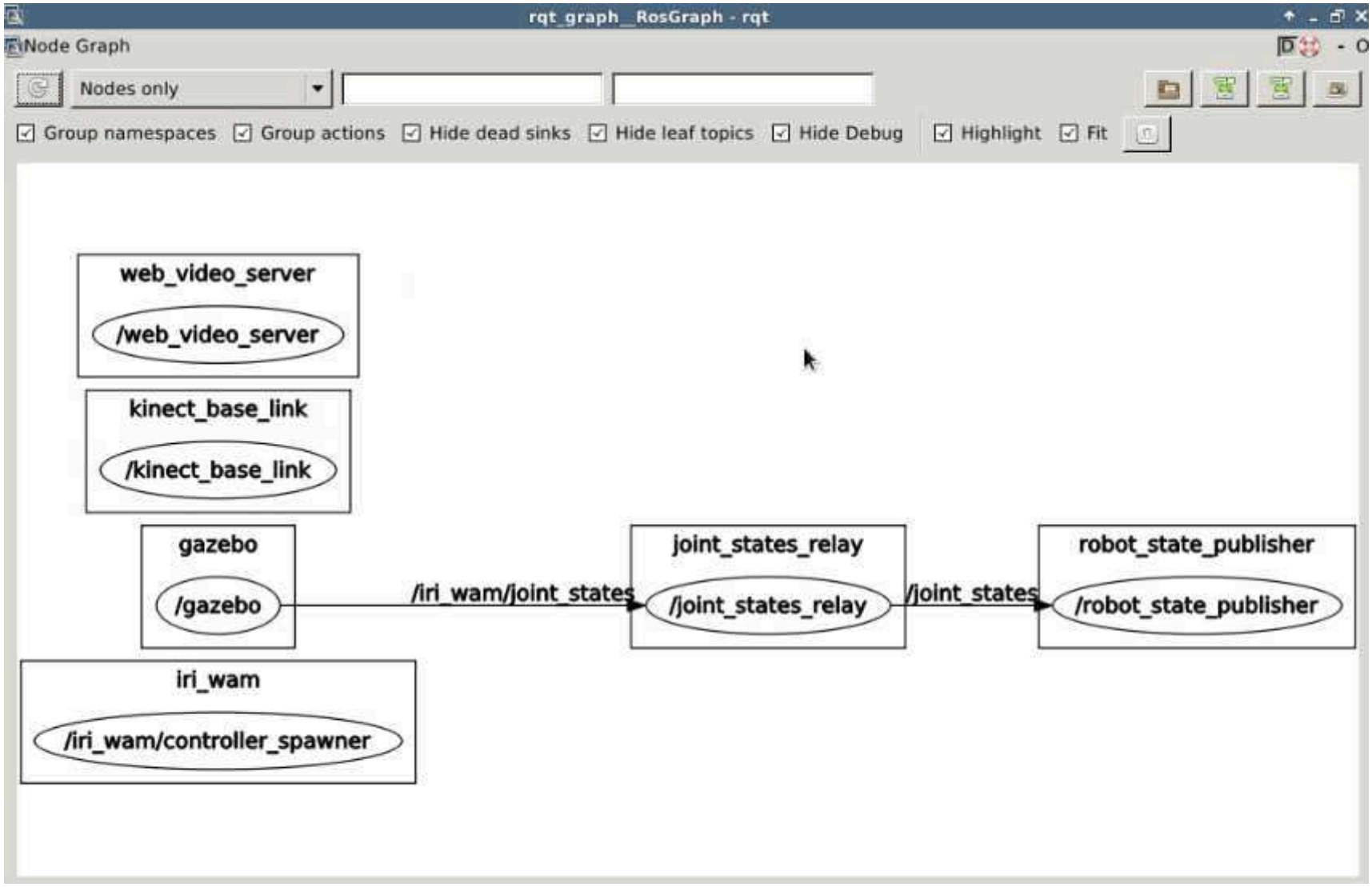


Fig.10.5 - Rqt-Graph Result

In the diagram {Fig-10.5}, you will be presented with all of the nodes currently running, connected by the topics they use to communicate with each other. There are two main elements that you need to know how to use:

- The **refresh button**: Which you have to press any time you have changed the nodes that are running:

- The **filtering options**: These are the three boxes just beside the refresh button. The first element lets you select between only nodes or topics. The second box allows you to filter by names of nodes.

Here is an example where you filter to just show the */gazebo* and the */joint_states_relay* {Fig-10.6}:

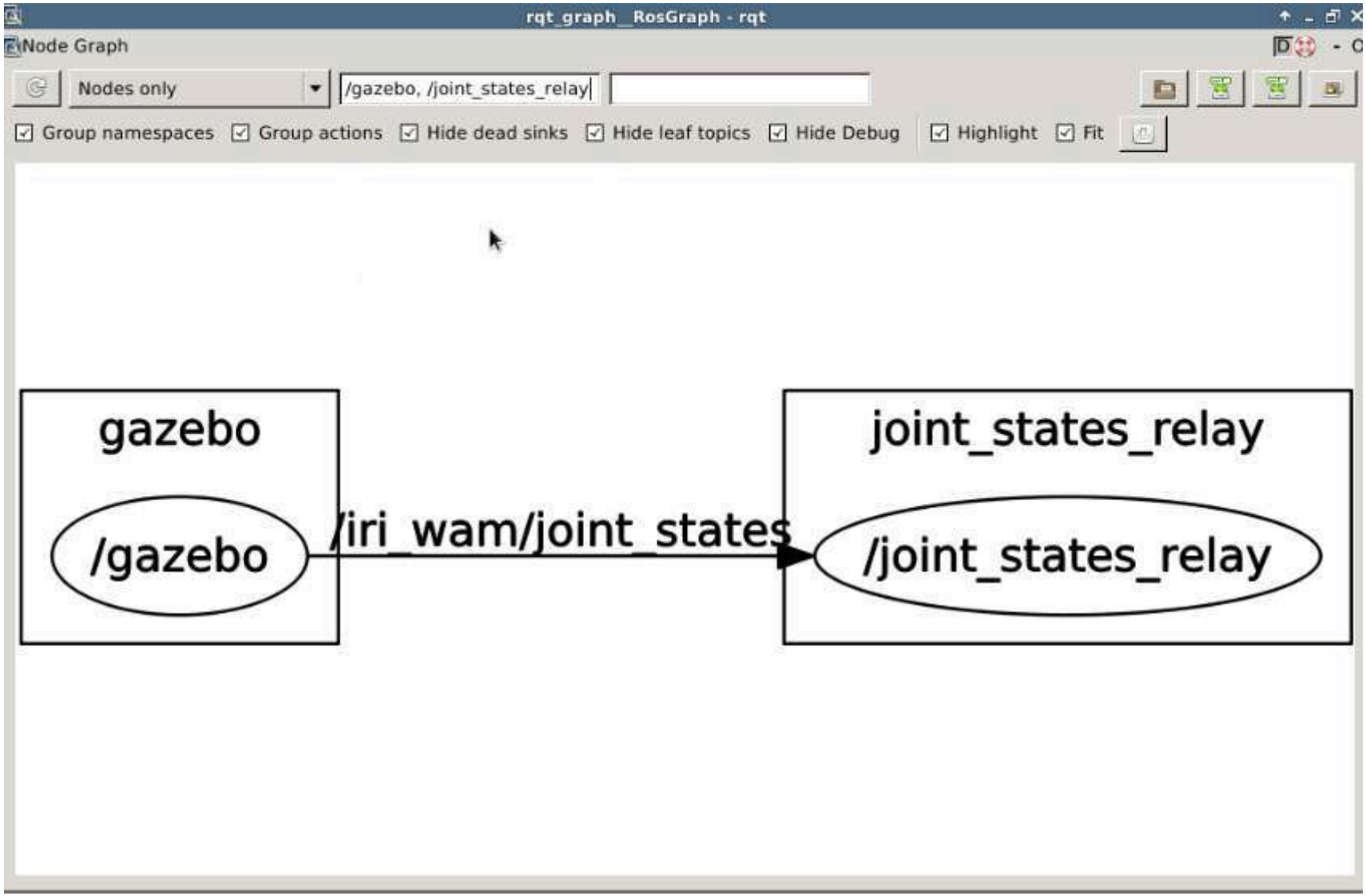


Fig-10.6 - Rqt-Graph Result Filtered /gazebo and /joint_states_relay

- Exercise 10.3 -

- Create a package that launches a simple Topic publisher and a subscriber, and filter the ros_graph output to show you only the two nodes and the topic you are interested in.

- End of Exercise 10.3 -

You should get something like this {Fig-10.7}:

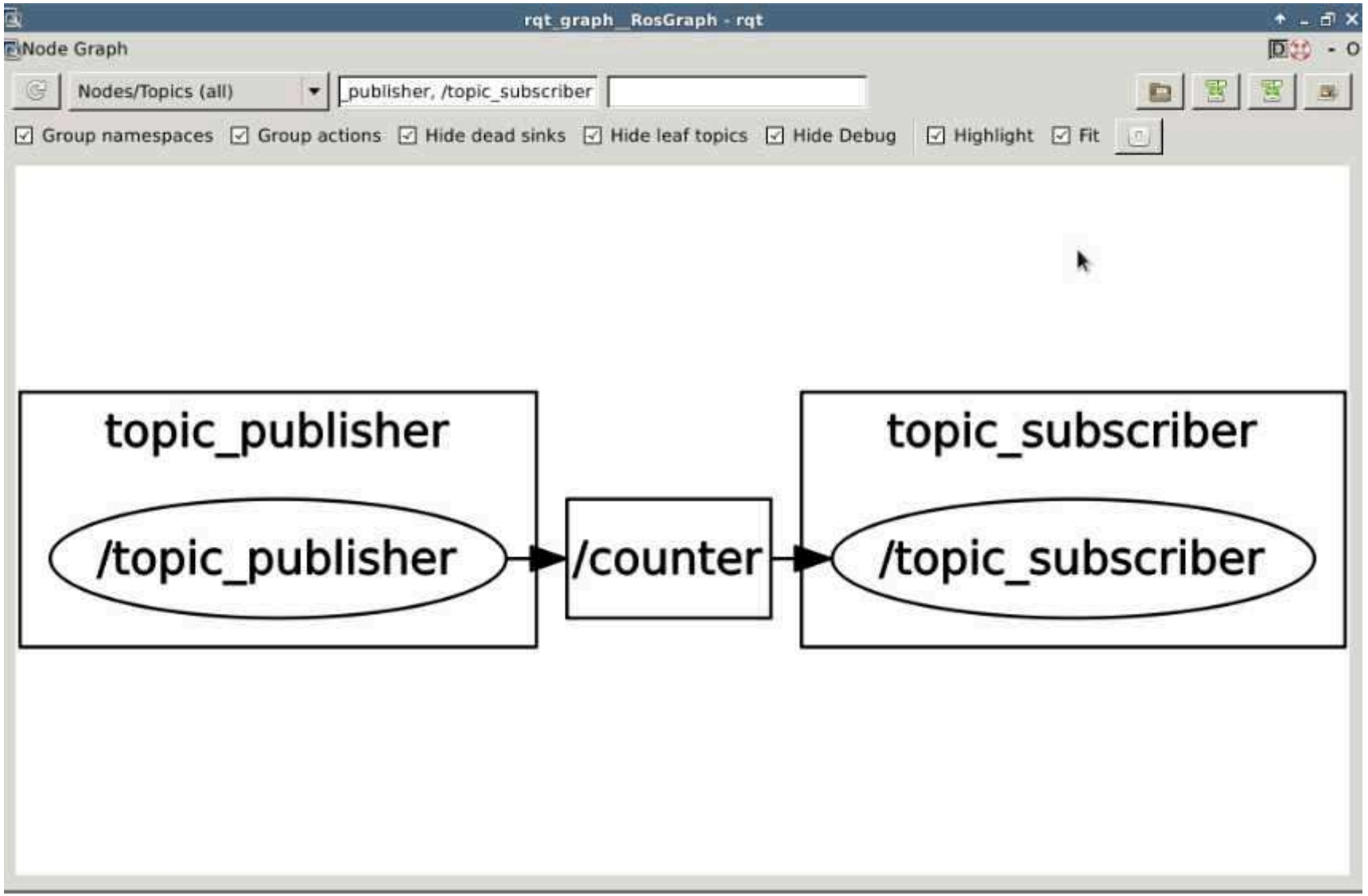


Fig-10.7 - Rqt-Graph from Exercise 10.7

10.4 Record Experimental Data and Rosbags

One very common scenario in robotics is the following:

You have a very expensive real robot, let's say R2-D2, and you take it to a very difficult place to get, let's say The Death Star. You execute your mission there and you go back to the base. Now, you want to reproduce the same conditions to improve R2-D2s algorithms to open doors. But you don't have the DeathStar nor R2-D2. How can you get the same exact sensory readings to make your test? Well, you record them, of course! And this is what **rosbag** does with all the ROS topics generated. It records all of the data passed through the ROS topics system and allows you to replay it any time through a simple file.

The commands for playing with **rosbag** are:

- To **Record** data from the topics you want:

  rosbag record -O name_bag_file.bag name_topic_to_record1 name_topic_to_record2 ... name_topic_to_recordN

- To **Extract** general information about the recorded data:

  rosbag info name_bag_file.bag

- To **Replay** the recorded data:

  rosbag play name_bag_file.bag

  Replaying the data will make the rosbag publish the same topics with the same data, at the same time when the data was recorded.

- Example 10.2 -

1- Go to the Terminal and type the following command to make the robot start moving if you don't haven't already:

▶ **Execute in Shell #1**

In [ ]:
```
roslaunch iri_wam_aff_demo start_demo.launch
```

2- Go to another Terminal and go to the *src* directory. Type the following command to record the data from the */laser_scan*:

▶ Execute in Shell #2

In [ ]:
```
roscd; cd ../src
```

In [ ]:
```
rosbag record -O laser.bag laser_scan
```

This last command will start the recording of data.

3- After 30 seconds or so, a lot of data have been recorded, press **[CTRL]+[C]** in the rosbag recording Terminal #2 to stop recording. Check that there has been a laser.bag file generated and it has the relevant information by typing:

▶ Execute in Shell #2

In [ ]:
```
rosbag info laser.bag
```

4- Once checked, CTRL+C on the **start_demo.launch** Terminal #1 to stop the robot.

5- Once the robot has stopped, type the following command to replay the *laser.bag* (the *-l* option is to loop the *rosbag* infinitely until you CTRL+C):

▶ Execute in Shell #2

In [ ]:
```
rosbag play -l laser.bag
```

6- Go to Terminal #3 and type the following command to read the ranges[100] of Topic */laser_scan* :

▶ Execute in Shell #2

In [ ]:
```
rostopic echo /laser_scan/ranges[100]
```

7- Type the following command in another Terminal , like Terminal #4. Then, go to the graphical interface and see how it plots the given data with rqt_plot:

▶ Execute in Shell #4

In [ ]:
```
rqt_plot /laser_scan/ranges[100]
```

Is it working? Did you find something odd?
There are various things that are wrong, but it's important that you memorize this.

1) The first thing you notice is that when you echo the topic ***/laser_scan*** topic, you get sudden changes in values.

Try getting some more information on who is publishing in the ***/laser_scan*** topic by writing in the Terminal:

Remember to hit [CTRL]+[C] if there was something running there first.

▶ Execute in Shell #3

In [ ]:
```
rostopic info /laser_scan
```

You will get something similar to this:

Shell #3 Output

```
Type: sensor_msgs/LaserScan

Publishers:
 * /gazebo (http://ip-172-31-27-126:59384/)
 * /play_1476284237447256367 (http://ip-172-31-27-126:41011/)

Subscribers: None
```

As you can see, **TWO** nodes are publishing in the **/laser_scan** topic: **gazebo** (the simulation) and **play_x** (the rosbag play).

This means that not only is rosbag publishing data, but also the simulated robot.

So the first thing to do is to **PAUSE** the simulation, so that gazebo stops publishing laser readings.

For that, you have to execute the following command:

▶ Execute in Shell #3

In [ ]:
```
rosservice call /gazebo/pause_physics "{}"
```

And to **UnPAUSE** it again, just for you to know, just:

▶ Execute in Shell #3

In [ ]:
```
rosservice call /gazebo/unpause_physics "{}"
```

With this, you should have stopped all publishing from the simulated robot part and only left the rosbag to publish.

NOTE: When you execute the command to pause Gazebo physics, the node **/gazebo** will stop publishing into the **/laser_scan** topic. However, if you execute a *rostopic info* command and check the laser topic, you will still see the **/gazebo** node as a Publisher, which may cause confusion. This happens because ROS hasn't updated the data yet. But do not worry, because **/gazebo** will not be publishing into the laser topic.

Now you should be able to see a proper **/laser_scan** plot in the rqt_plot.

Still nothing?

2) Check the time you have in the rqt_plot. Do you see that, at a certain point, the time doesn't keep on going?

That's because you have stopped the simulation so that the time is not running anymore, apart from the tiny time frame in the rosbag that you are playing now.

Once rqt_plot reaches the maximum time, it stops. It doesn't return to the start, and therefore, if the values change outside the last time period shown, you won't see anything.

Take a look also at the rosbag play information of your currently running rosbag player

Shell #2 Output

```
[ INFO ] [1471001445.5575545086]: Opening laser.bag

Waiting 0.2 seconds after advertising topics... done.

Hit space to toggle paused, or 's' to stop.
[RUNNING] Bag Time: 61.676140 Duration: 41.099140 / 41.452000
```

In this example, you can see that the current time is **41.099** of **41.452 seconds** recorded. And the rosbag time, therefore, will reach a maximum of around **62** seconds.

62 seconds, in this case, is the maximum time the rqt_plot will show, and will start around 20 seconds.

Therefore, you have to always **CLEAR** the plot area with the **clear button** in rqt_plot.

By doing a **CLEAR**, you should get something similar to {Fig-10.8}:

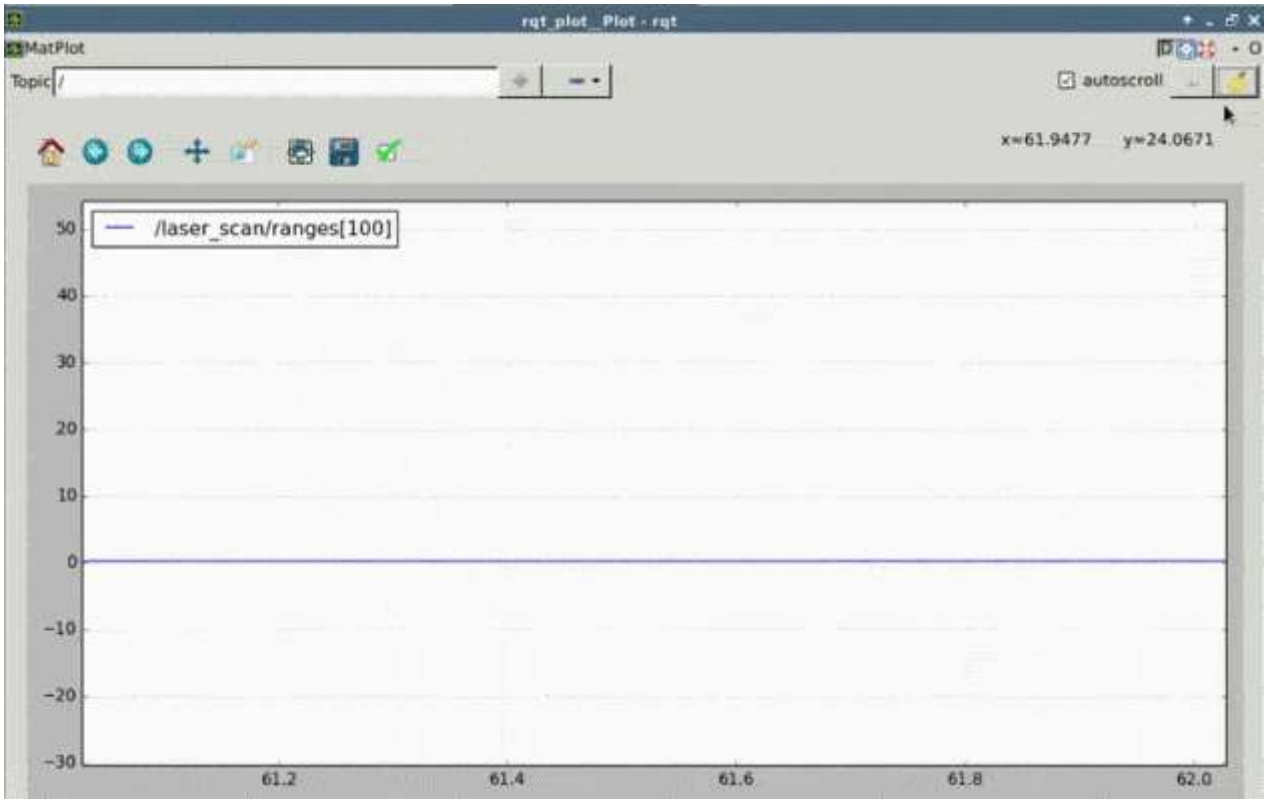


Fig-10.8 - RosBag Rqt-Plot

To summarize:

To use rosbag files, you have to make sure that the original data generator (real robot or simulation) is NOT publishing. Otherwise, you will get really weird data (the collision between the original and the recorded data). You have to also keep in mind that if you are reading from a rosbag, time is finite and cyclical, and therefore, you have to clear the plot area to view all of the time period.

rosbag is especially useful when you don't have anything in your system (neither real nor simulated robot), and you run a bare **roscore**. In that situation, you would record all of the topics of the system when **you do have** either a simulated or real robot with the following command:

In [ ]:
```
rosbag record -a
```

This command will record **ALL** the topics that the robot is publishing. Then, you can replay it in a bare roscore system and you will get all of the topics as if you had the robot.

Before continuing any further, please check that you've done the following:

- You have stopped the rosbag play by going to the Terminal #2 where it's executing and [CTRL]+[C]
- You have unpaused the simulation to have it working as normal:

▶ Execute in Shell #3

In [ ]:
```
rosservice call /gazebo/unpause_physics "{}"
```

- End of Example 10.2 -

10.5 RViz

And here you have it. The **HollyMolly!** The Milenium Falcon! The most important tool for ROS debugging....**RVIZ**.

RVIZ is a tool that allows you to visualize *Images, PointClouds, Lasers, Kinematic Transformations, RobotModels*...The list is endless. You even can define your own markers. It's one of the reasons why ROS got such a great acceptance. Before RVIZ, it was really difficult to know what the Robot was perceiving. And that's the main concept:

RVIZ is **NOT** a simulation. I repeat: It's **NOT** a simulation.
RVIZ is a representation of what is being published in the topics, by the simulation or the real robot.

RVIZ is a really complex tool and it would take you a whole course just to master it. Here, you will get a glimpse of what it can give you.

Remember that you should have unpaused the simulations and stopped the rosbag as described in the rosbag section.

1- Type in Terminal #2 the following command:

▶ Execute in Shell #2

```
In [ ]: rosrun rviz rviz
```

2- Wait a few seconds to see the RVIZ GUI:

You will be greeted by a window like {Fig-10.9}:

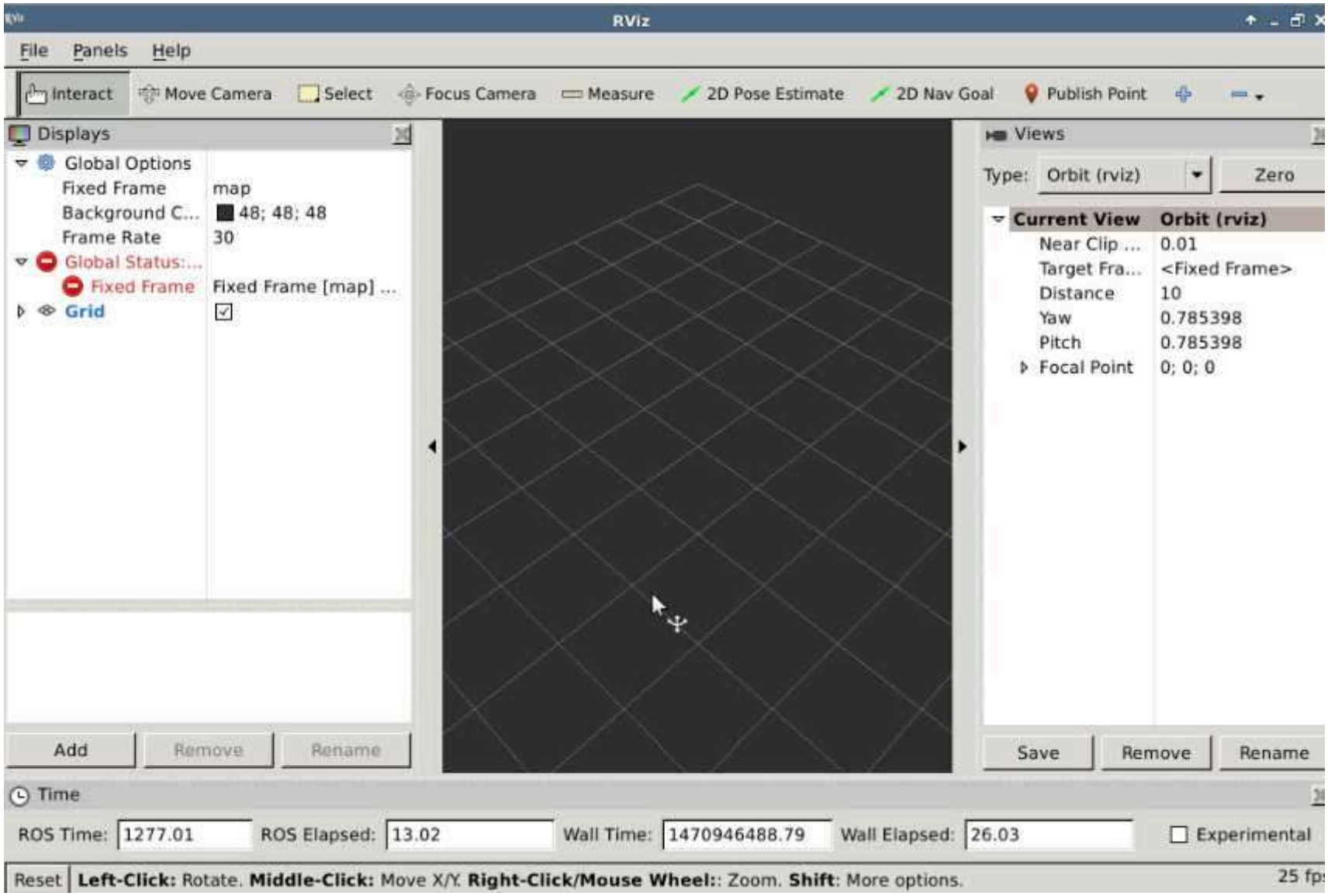


Fig-10.9 - RVIZ Starting Window

NOTE: In case you don't see the lower part of Rviz (the Add button, etc.), double-click at the top of the window to maximize it. Then you'll see it properly.

You need only to be concerned about a few elements to start enjoying RVIZ.

- **Central Panel**: Here is where all the magic happens. Here 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 (RIGHT-CLICK PRESSED).
- **Left Displays Panel**: Here is where you manage/configure all the elements that you wish to visualize in the central panel. You only need to use two elements:
- In **Global Options**, you have to select the **Fixed Frame** that suits you for the visualization of the data. It is the reference frame from which all the data will be referred to.
- The **Add button**. Clicking here you get all of the types of elements that can be represented in RVIZ.

Go to RVIZ in the graphical interface and add a TF element. For that, click "Add" and select the element TF in the list of elements provided, as shown in {Fig-10.10}.

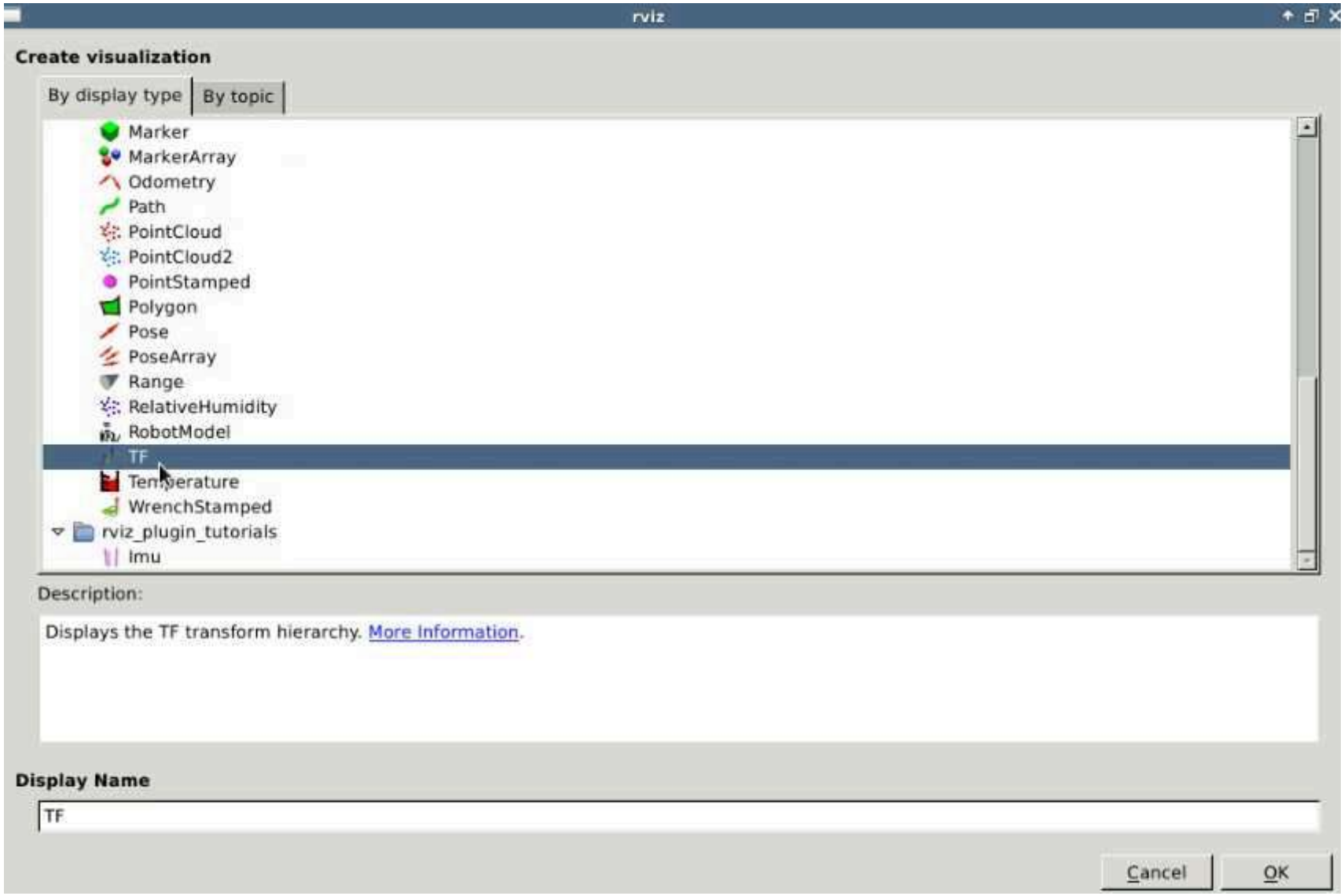


Fig-10.10 - RVIZ Add element

- Go to the RVIZ Left panel, select as Fixed Frame the ***iri_wam_link_footprint*** and make sure that the **TF** element checkbox is checked. In a few moments, you should see all of the Robots Elements Axis represented in the CENTRAL Panel.
- Now, go to a Terminal #1 and enter the command to move the robot:

▶ Execute in Shell #1

In [ ]:
```
roslaunch iri_wam_aff_demo start_demo.launch
```

You should see something like this:

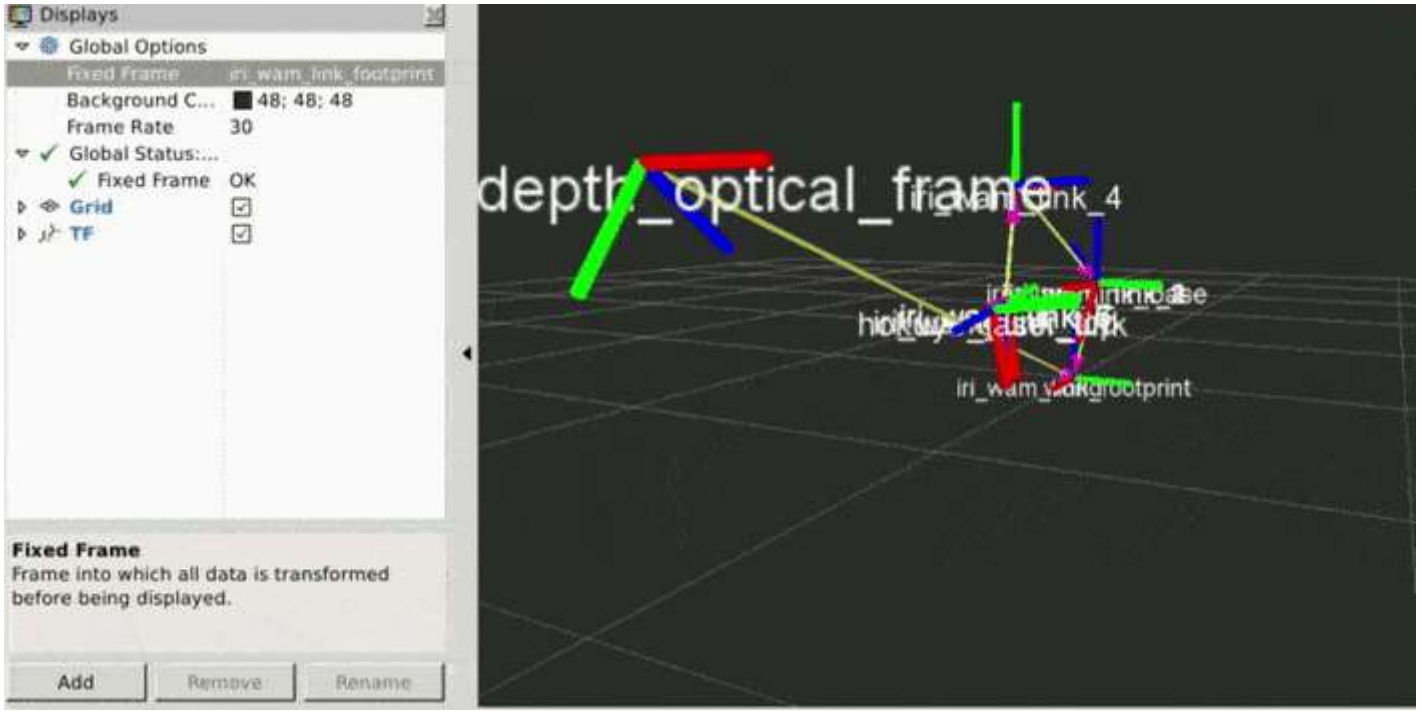


Fig-10.11 - RVIZ TF

In {Fig-10.11}, you are seeing all of the transformations elements of the IRI Wam Simulation in real-time. This allows you to see exactly what joint transformations are sent to the robot arm to check if it's working properly.

- Now press "Add" and select *RobotModel*, as shown in {Fig-10.12}

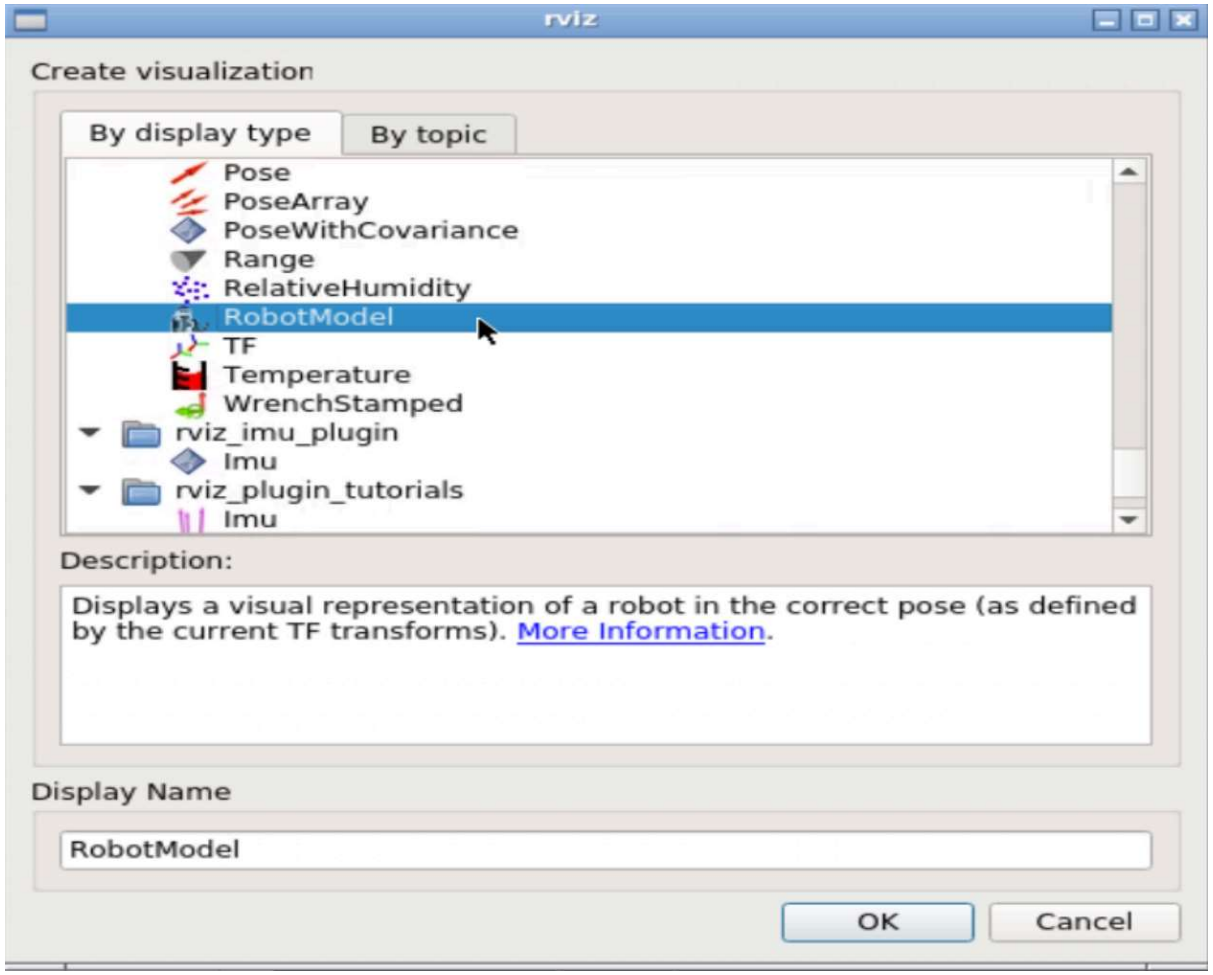


Fig-10.12 - RVIZ Add Robot Model

You should see now the 3D model of the robot, as shown in {Fig-10.13}:

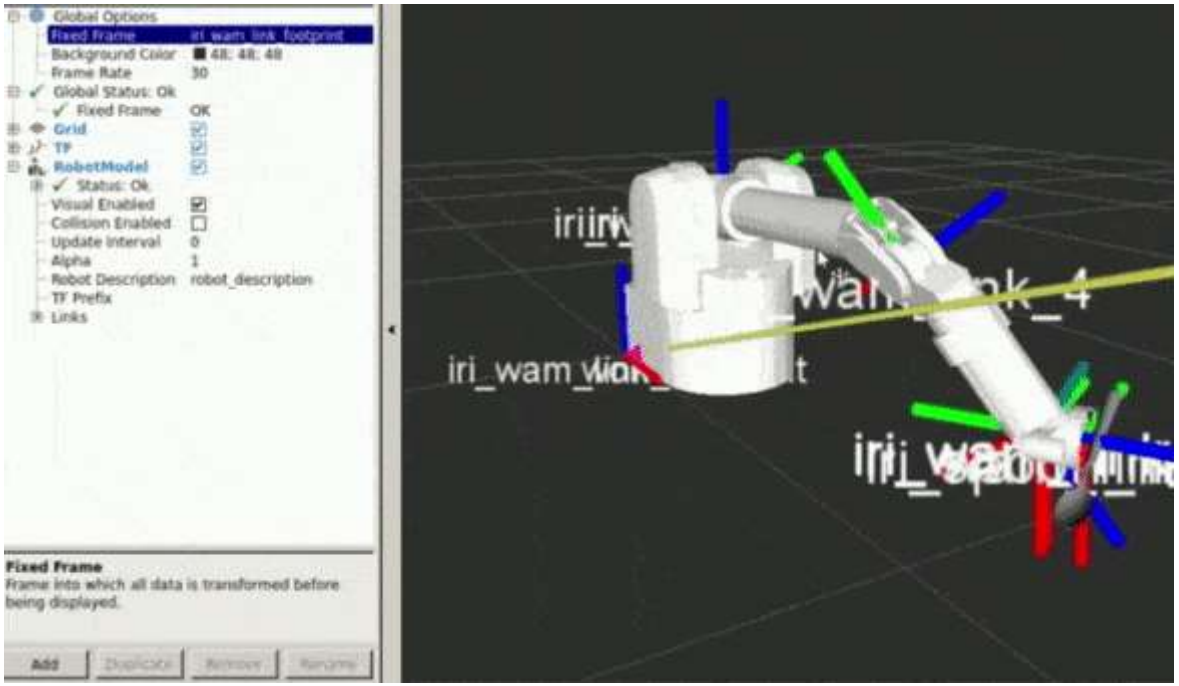


Fig-10.13 - RVIZ Robot Model + TF

Why can't you see the table? Or the bowl? Is there something wrong? Not at all!

Remember: RVIZ is **NOT** a simulation, **it represents what the TOPICS are publishing**. In this case the models that are represented are the ones that the *RobotStatePublisher node* is publishing in some ROS topics. There is NO node publishing about the bowl or the table.

Then how can you see the object around? Just like the robot does, through cameras, lasers, and other topic data.

Remember: RVIZ shows what your robot is perceiving, nothing else.

- Exercise 10.4 -

Add to RVIZ the visualization of the following elements:

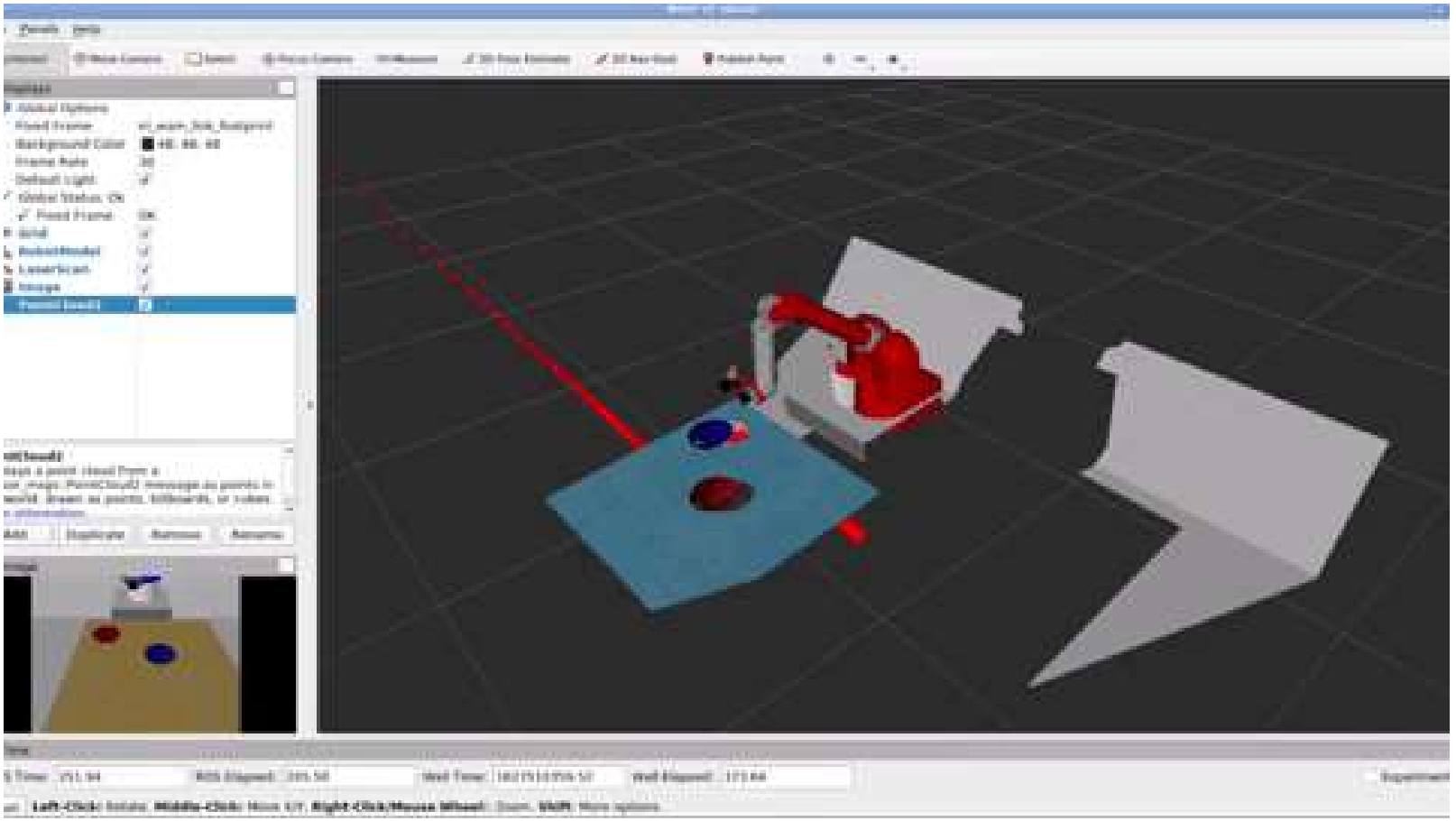
- What the RGB camera from the Kinect is seeing. **TIP: The topic it has to read is /camera/rgb/image_raw. It might take a while to load the images, so just be patient.**
- What the Laser mounted at the end effector of the robot arm is registering. **TIP: You can adjust the appearance of the laser points through the element in the LEFT PANEL.**
- What the PointCloud Camera / Kinect mounted in front of the robot arm is registering. **TIP: You can adjust the appearance of the pointcloud points through the element in the LEFT PANEL. You should select points for better performance.**

TIP: You should have a similar result as the one depicted beneath:
Notice that activating the pointcloud has a huge impact on the system performance. This is due to the huge quantity of data being represented. It's highly recommended to only use it with a high-end graphics card.

Play around with the type of representations of the laser, size, and so on, as well as with the pointcloud configuration.

- End of Exercise 10.4 -

- Expected Result for Exercise 10.4 -



- End of Expected Result -

Congratulations! Now you are ready to debug any AstroMech out there!

10.6 Additional information to learn more

roswtf: http://wiki.ros.org/roswtf (http://wiki.ros.org/roswtf)

Ros Logging System: http://wiki.ros.org/rospy/Overview/Logging (http://wiki.ros.org/rospy/Overview/Logging)

rqt_console: http://wiki.ros.org/rqt_console (http://wiki.ros.org/rqt_console)

rqt_plot: http://wiki.ros.org/rqt_plot (http://wiki.ros.org/rqt_plot)

rqt_graph: http://wiki.ros.org/rqt_graph (http://wiki.ros.org/rqt_graph)

Rosbag: http://wiki.ros.org/rosbag (http://wiki.ros.org/rosbag)

Rviz: http://wiki.ros.org/rviz (http://wiki.ros.org/rviz)