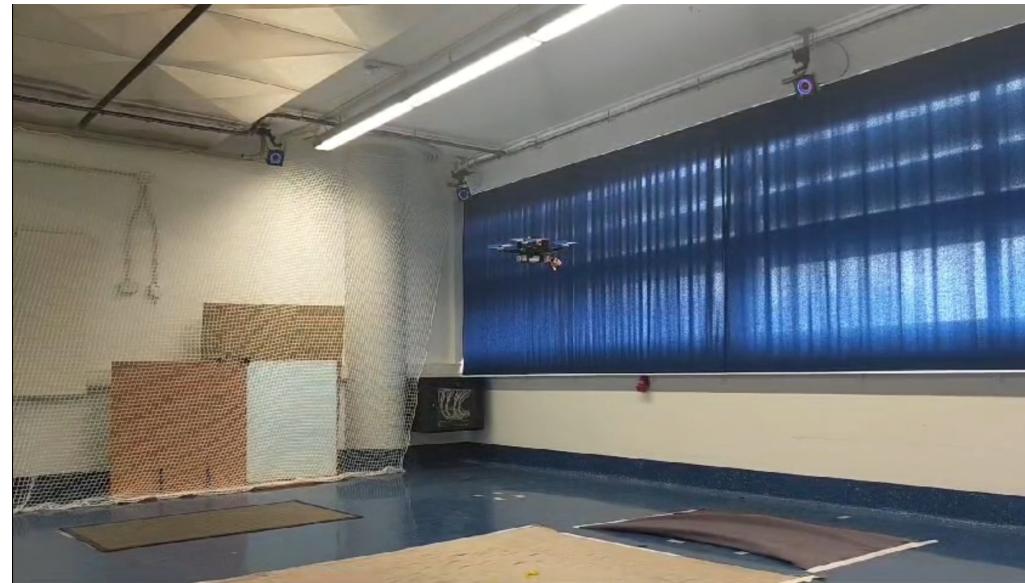


ROS 2 Workshop

Marcelo Jacinto
[\(mjacinto@isr.tecnico.ulisboa.pt\)](mailto:mjacinto@isr.tecnico.ulisboa.pt)

Motivation

ROS 2 is useful if you ever dream of ruling the world with an army of drones...



Why ROS 2

2 > 1



What is ROS 2

Not an operating system! (they use the name as clickbait)

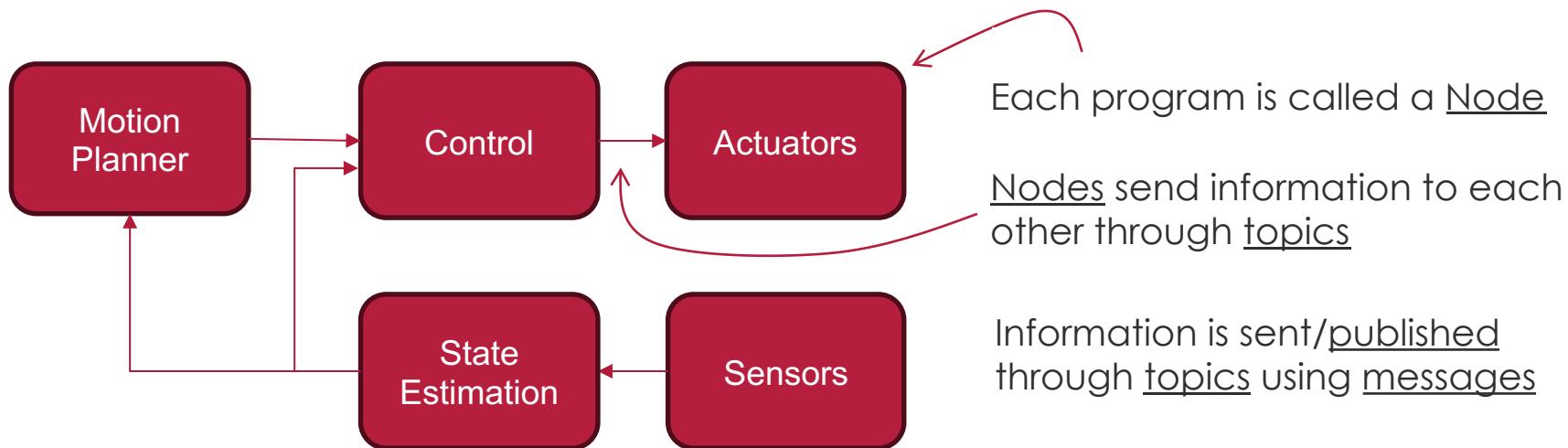
“The Robot Operating System (ROS) is a set of software libraries and tools for building robot applications.”

ROS 2 provides:

- a set of useful programs that run in your computer
- a communication system for programs to share data with each other

Communication Structure

Problem Formulation/Motivation: You want to design a Guidance, Control and Navigation (GNC) system for a drone, such that each block of the system is implemented as an individual program (written in Python or C++).



Installing ROS 2

Distro	Release date	Logo	EOL date
Iron Irwini	May 23rd, 2023		November 2024
Humble Hawkbill	May 23rd, 2022		May 2027
Galactic Geochelone	May 23rd, 2021		December 9th, 2022
Foxy Fitzroy	June 5th, 2020		June 20th, 2023

Follow the steps on the links bellow:

Ubuntu 22.04LTS

Official Documentation:

<https://docs.ros.org/en/humble/Installation/Ubuntu-Install-Debians.html>

Ubuntu 20.04LTS

Official Documentation:

<https://docs.ros.org/en/foxy/Installation/Ubuntu-Install-Debians.html>

Installation tutorials on YouTube: <https://youtu.be/0aPbWsyENA8?t=122&si=CbwthPBpd0aOQdI->

After Installing ROS 2

Run the following line on the terminal

```
source /opt/ros/humble/setup.bash >> ~/.bashrc  
    ↘ foxy (if using Ubuntu 20.04LTS)
```

Tell your terminal where
ROS 2 is installed

Also install colcon build tools

```
sudo apt install python3-colcon-common-extensions
```

```
echo "source /usr/share/colcon_argcomplete/hook/colcon-argcomplete.bash" >> ~/.bashrc  
echo "source /usr/share/colcon_cd/function/colcon_cd.sh" >> ~/.bashrc  
echo "export _colcon_cd_root=/opt/ros/humble/" >> ~/.bashrc
```

↘ foxy (if using Ubuntu 20.04LTS)

Convenience tools such as
auto-complete in the terminal

Getting Started

Create a folder/workspace where the code will live.

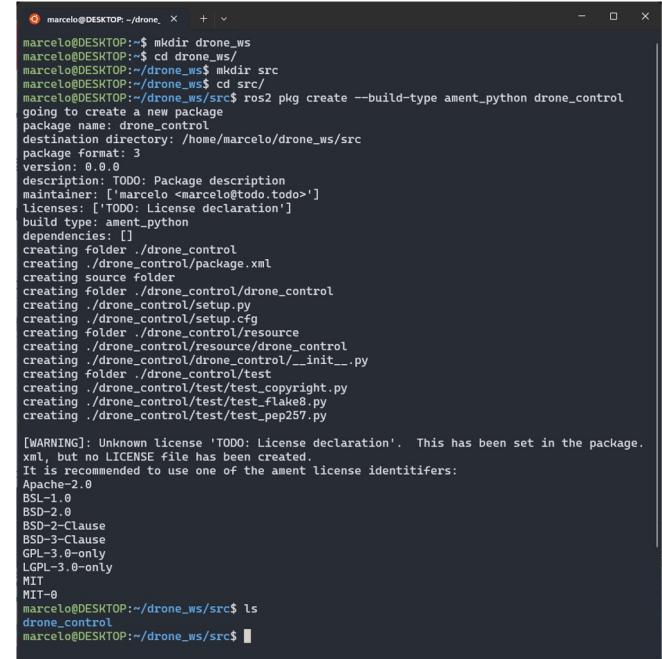
```
mkdir drone_ws
```

Inside the workspace create a src folder/directory

```
cd drone_ws  
mkdir src
```

Inside the src directory create a Python code package

```
cd src  
ros2 pkg create --build-type ament_python drone_control
```



The terminal window shows the following command sequence:

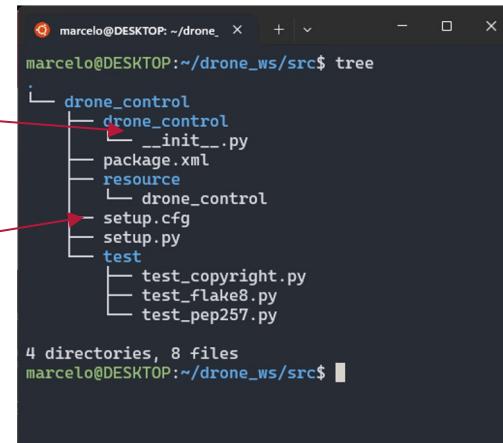
```
marcelo@DESKTOP:~/drone_ws$ mkdir drone_ws  
marcelo@DESKTOP:~/drone_ws$ cd drone_ws/  
marcelo@DESKTOP:~/drone_ws$ mkdir src  
marcelo@DESKTOP:~/drone_ws$ cd src/  
marcelo@DESKTOP:~/drone_ws/src$ ros2 pkg create --build-type ament_python drone_control  
going to create a new package  
package name: drone_control  
destination directory: /home/marcelo/drone_ws/src  
package_format: 3  
version: 0.0.0  
description: TODO: Package description  
maintainer: ['marcelo <marcelo@todo.todo>']  
licenses: ['TODO: License declaration']  
build type: ament_python  
dependencies: []  
creating folder ./drone_control  
creating ./drone_control/package.xml  
creating source Folder  
creating folder ./drone_control/drone_control  
creating ./drone_control/setup.py  
creating ./drone_control/setup.cfg  
creating folder ./drone_control/resource  
creating ./drone_control/resource/drone_control  
creating ./drone_control/drone_control/_init_.py  
creating folder ./drone_control/test  
creating ./drone_control/test/test_copyright.py  
creating ./drone_control/test/test_flake8.py  
creating ./drone_control/test/test_pep257.py
```

[WARNING]: Unknown license 'TODO: License declaration'. This has been set in the package.xml, but no LICENSE file has been created.
It is recommended to use one of the ament license identifiers:
Apache-2.0
BSD-1.0
BSD-2.0
BSD-2-Clause
BSD-3-Clause
GPL-3.0-only
LGPL-3.0-only
MIT
MIT-0
marcelo@DESKTOP:~/drone_ws/src\$ ls
drone_control
marcelo@DESKTOP:~/drone_ws/src\$ █

Folder/Package Structure (Python)

Write your code here

Inform ROS 2 where
your “main” is in the
setup.py file



```
marcelo@DESKTOP:~/drone_ws/src$ tree
.
└── drone_control
    ├── __init__.py
    ├── package.xml
    ├── resource
    │   └── drone_control
    ├── setup.cfg
    ├── setup.py
    └── test
        ├── test_copyright.py
        ├── test_flake8.py
        └── test_pep257.py

4 directories, 8 files
marcelo@DESKTOP:~/drone_ws/src$
```

Creating a Node

Python file
for node
+
Python file
for logic

Timer to periodically
compute the control signal

The screenshot shows a terminal window titled "drone_control (WSL: Ubuntu-22.04)". The code is a Python script named "drone_controller_node.py". It imports various ROS 2 libraries and defines a class "DroneControllerNode" that inherits from "drone.SimulationNode". The class reads parameters like mass, frequency, and controller gains from the configuration. It then creates a controller object using these parameters. A timer is set to periodically update the control signal based on the current state and a target waypoint. Publishers and subscribers are also created for odometry, state, and waypoint topics.

```
drone_controller_node.py
# Import the ROS 2 libraries
import rclpy
from rclpy.node import Node
from rclpy.qos import qos_profile_sensor_data, qos_profile_system_default

# Import the ROS 2 message to receive the current state of the drone
from nav_msgs.msg import Odometry

# Import my custom message to send the control to apply to the drone
from drone_msgs.msg import Control, WaypointReference

# Create a class that inherits the Drone object
class DroneControllerNode(Node):

    def __init__(self):
        super().__init__("drone_simulation_node",
                         allow_undeclared_parameters=True,
                         automatically_declare_parameters_from_overrides=True)

    # Get the mass, inertia and frequency from the configurations
    mass = self.get_parameter("drone_controller.mass").get_parameter_value().double_value
    frequency = self.get_parameter("drone_controller.frequency").get_parameter_value().double_value

    # Get the gains of the controller
    kp_pos = self.get_parameter("drone_controller.position.kp").get_parameter_value().double_array_value
    kd_pos = self.get_parameter("drone_controller.position.kd").get_parameter_value().double_array_value
    kp_theta = self.get_parameter("drone_controller.attitude.kp").get_parameter_value().double_value
    kd_theta = self.get_parameter("drone_controller.attitude.kd").get_parameter_value().double_value

    # Log the parameters
    self.get_logger().info("Drone mass: {}".format(mass))
    self.get_logger().info("Drone position gains: {}.".format(kp_pos, kd_pos))
    self.get_logger().info("Drone attitude gains: {}.".format(kp_theta, kd_theta))
    self.get_logger().info("Drone frequency: {}.".format(frequency))

    # Create the object that actually simulates the dynamics of a drone
    self.controller = Controller(kp_pos, kd_pos, kp_theta, kd_theta, mass)

    # Set the current state of the system
    self.x = None
    self.y = None
    self.theta = None
    self.x_dot = None
    self.y_dot = None
    self.theta_dot = None

    # Get the desired waypoint
    self.x_target = None
    self.y_target = None
    self.theta_target = None

    # Publisher to the control inputs of the system
    self.control_publisher = self.create_publisher(Control, "input", qos_profile_sensor_data)

    # Subscription to the current state of the system and for the references waypoint for the controller to track
    self.state_subscriber = self.create_subscription(Odometry, "state", self.state_callback, qos_profile_system_default)
    self.waypoint_subscriber = self.create_subscription(WaypointReference, "waypoint", self.waypoint_callback, qos_profile_system_default)

    # Create a timer that will perform the update step of the simulation
    self.timer = self.create_timer(1.0/frequency, self.update_control)
```

ROS 2 library imports

Reading parameters
dynamically

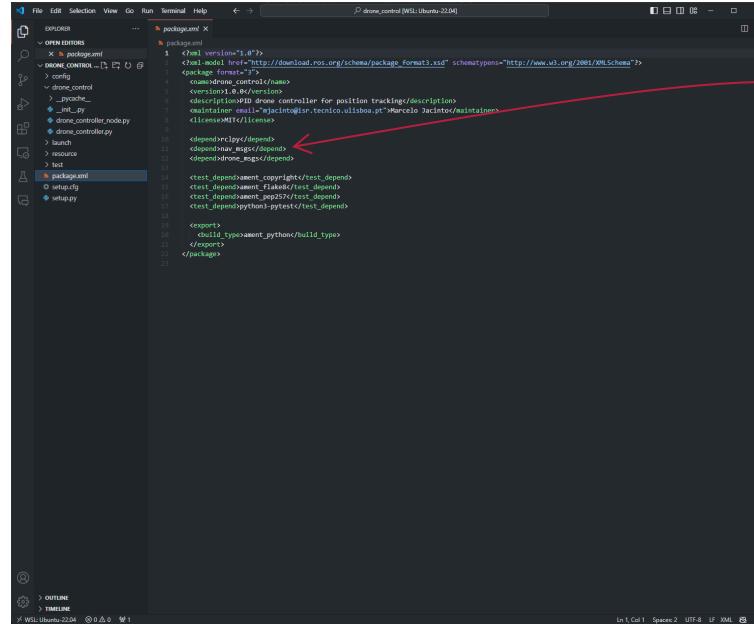
Creating the
publishers/subscribers

Creating a Node (2)

Subscriber callbacks
are called when a new
message arrives

Function called by the timer to compute and publish the control signal

Creating a Node (3)

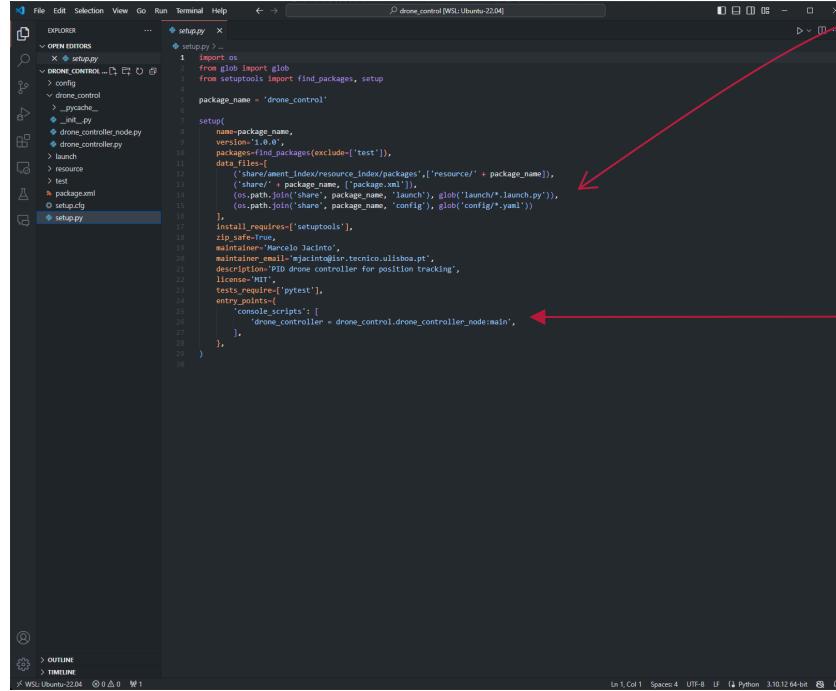


The screenshot shows a code editor interface with a dark theme. On the left is the Explorer sidebar, which lists several files and folders: package.xml, package.srv, drone_control_node.py, drone_controller_node.py, setup.py, and setup_dx.dg. The main area displays the content of package.xml. A red arrow points from the text "Add the used packages to the package.xml file" to the line in the XML code where dependencies are listed.

```
<package>
  <name>drone_control</name>
  <version>0.0.1</version>
  <description>PID drone controller for position tracking</description>
  <maintainer email="mjacinto@isr.tecnico.ulisboa.pt">Marcelo Jacinto</maintainer>
  <license>MIT</license>
  <dependencies>
    <depend name="rospy"/>
    <depend name="nav_msgs"/>
    <depend name="drone_msgs"/>
  </dependencies>
  <test_depend name="rostest"/>
  <test_depend name="rostest_pytest"/>
  <test_depend name="pep8"/>
  <test_depend name="python-pytest"/>
  <export>
    <build_type>ament_python</build_type>
  </export>
</package>
```

Add the used
packages to the
package.xml file

Creating a Node (4)



```
File Edit Selection View Go Run Terminal Help < -> drone_control [WSL:Ubuntu-22.04] setup.py X setup.py ... OPEN EDITORS X setup.py ... setup() 1 import os 2 from glob import glob 3 from setuptools import find_packages, setup 4 5 package_name = 'drone_control' 6 7 setup( 8     name=package_name, 9     version='0.0.1',10     packages=find_packages(exclude=['test']),11     data_files=[12         ('share/ament_index/resource_index/packages', ['resource/' + package_name]),13         ('share/' + package_name, ['package.xml']),14         (os.path.join('share', package_name, 'launch'), glob('launch/*.launch.py')),15         (os.path.join('share', package_name, 'config'), glob('config/*.yaml'))16     ],17     install_requires=['setuptools'],18     zip_safe=False,19     maintainer='Marcelo Jacinto',20     maintainer_email='mjacinto@isr.tecnico.ulisboa.pt',21     description='PID drone controller for position tracking',22     license='Apache Software License 2.0',23     tests_require=['pytest'],24     entry_points={25         'console_scripts': [26             'drone_controller = drone_control.drone_controller_node:main',27         ],28     },29 )
```

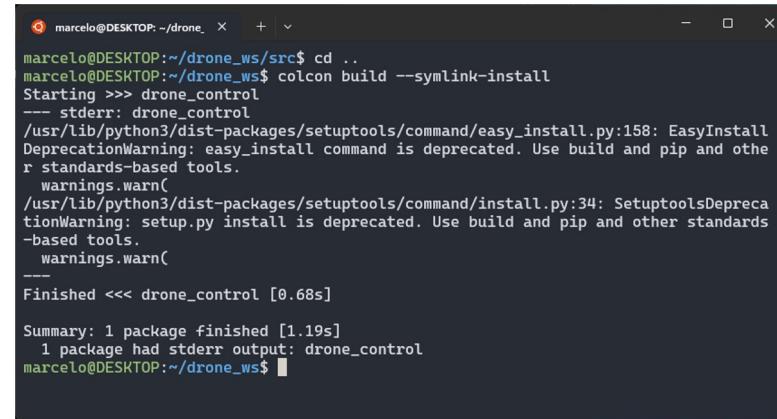
Index configuration
and launch directories
here

Setup your python
executable nodes here

Compiling/Indexing the Package

To compile all the packages, run (at the workspace level):

```
colcon build --symlink-install
```



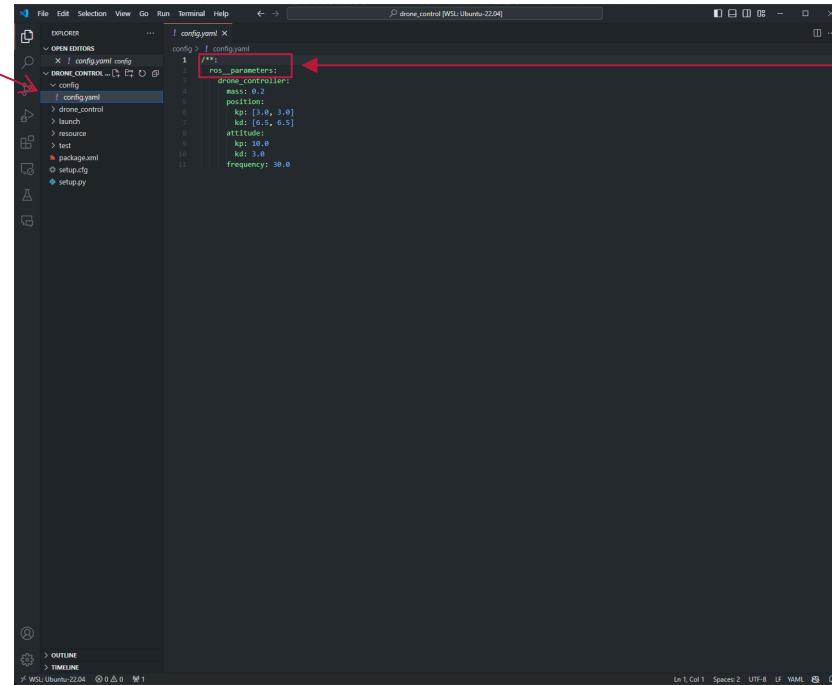
A terminal window titled "marcelo@DESKTOP: ~/drone_ws/" showing the command "colcon build --symlink-install". The output includes several deprecation warnings from Python's setuptools module, such as "DeprecationWarning: easy_install command is deprecated. Use build and pip and other standards-based tools." and "SetupToolsDeprecationWarning: setup.py install is deprecated. Use build and pip and other standards-based tools.". The terminal ends with a summary: "Summary: 1 package finished [1.19s]" and "1 package had stderr output: drone_control".

If you get these
warnings, just run
the following lines

```
PYTHONWARNINGS="ignore:setup.py install is deprecated::setuptools.command.install" >> ~/.bashrc  
export PYTHONWARNINGS >> ~/.bashrc
```

Dynamic Parameters

Create a configuration directory with a YAML file



- Use `/**` such that all the parameters are available to all nodes

Launch Files

Specify the yaml file
that contains
parameters

Return the node and arguments to be evaluated at runtime

```
File Edit Selection View Go Run Terminal Help < -> drone_control [WSL:Ubuntu-22.04]
explorer ... control.launchpy x
OPEN EDITORS
  X W: control.launchpy launch
  E: dronecontrol, WSL:UBUNTU-2...
  config
    config.yaml
  drone_control
  launch
  control.launchpy
  resource
  test
  package.xml
  setup.cfg
  setup.py

control.launchpy
  ...
  launch > control.launchpy # generate.launch_description
  launch > control.launchpy # generate.launch_description
  import os
  from ament_index_python.packages import get_package_share_directory
  from launch import LaunchDescription
  from launch.substitutions import LaunchConfiguration
  from launch.actions import DeclareLaunchArgument
  from launch_ros.actions import Node
  ...
  def generate_launch_description():
  ...
  # ----- DECLARE THE LAUNCH ARGUMENTS -----
  ...
  id_namespace_and_id_of_the_vehicle_is_generated_by_the_launch_file
  id_arg = DeclareLaunchArgument('vehicle_id', default_value='1', description='Drone ID in the network')
  namespace_arg = DeclareLaunchArgument('vehicle_ns', default_value='vehicle', description='Namespace to append to every topic and node name')
  ...
  # Get the name of the yaml configuration file either from the package or an external source
  controller_yaml_arg = DeclareLaunchArgument(
    'controller_yaml',
    default_value=os.path.join(get_package_share_directory('drone_control'), 'config', 'config.yaml'),
    description='The configurations for the controller')
  ...
  simulator_node = Node(
    package='drone_control',
    name='drone_simulator',
    ...
    LaunchConfiguration('vehicle_ns'),
    LaunchConfiguration('vehicle_id')),
    executable='drone_controller',
    name='drone_controller',
    output='screen',
    emulate_tty=True,
    parameters=[
      LaunchConfiguration('controller_yaml'),
      {
        'vehicle_id': LaunchConfiguration('vehicle_id'),
        'vehicle_ns': LaunchConfiguration('Vehicle_ns')
      }
    ]
  )
  ...
  # Return the node to be launched by ROS2
  return LaunchDescription([
    id_arg,
    namespace_arg,
    controller_yaml_arg,
    ...
    simulator_node])
```

Define some parameters directly in the launch file

Inject the parameters from the yaml file in the node

Compiling/Indexing the Package (2)

To compile all the packages, run (at the workspace level):

```
colcon build --symlink-install
```



Important! Otherwise, every time a launch file or configuration files is changed, you must re-compile the code!

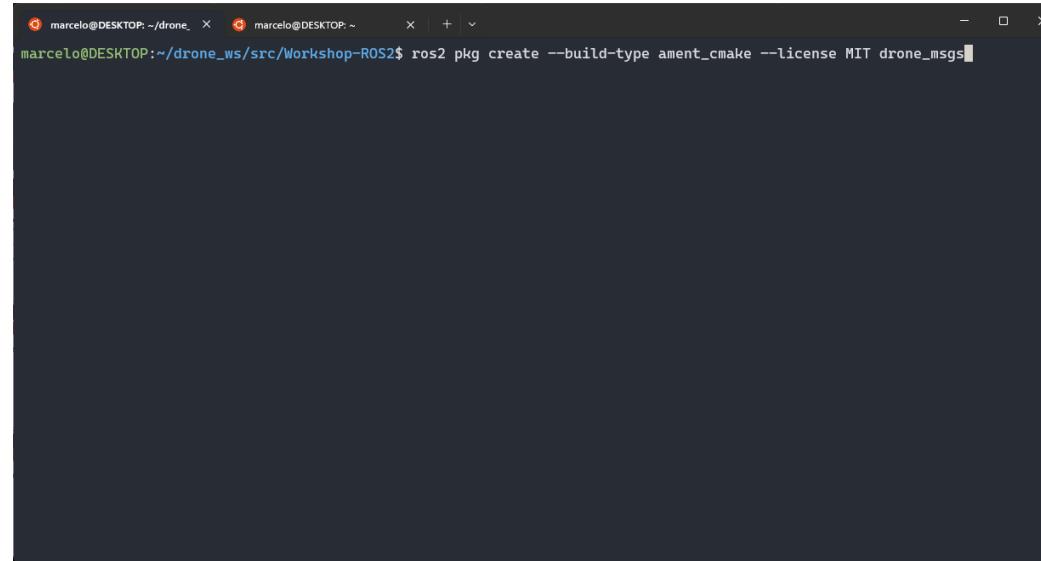
To compile a specific package, run (at the workspace level):

```
colcon build --symlink-install --packages-select <my_package_name>
```

To clear all the compiled packages, run (at the workspace level):

```
rm -rf install build install
```

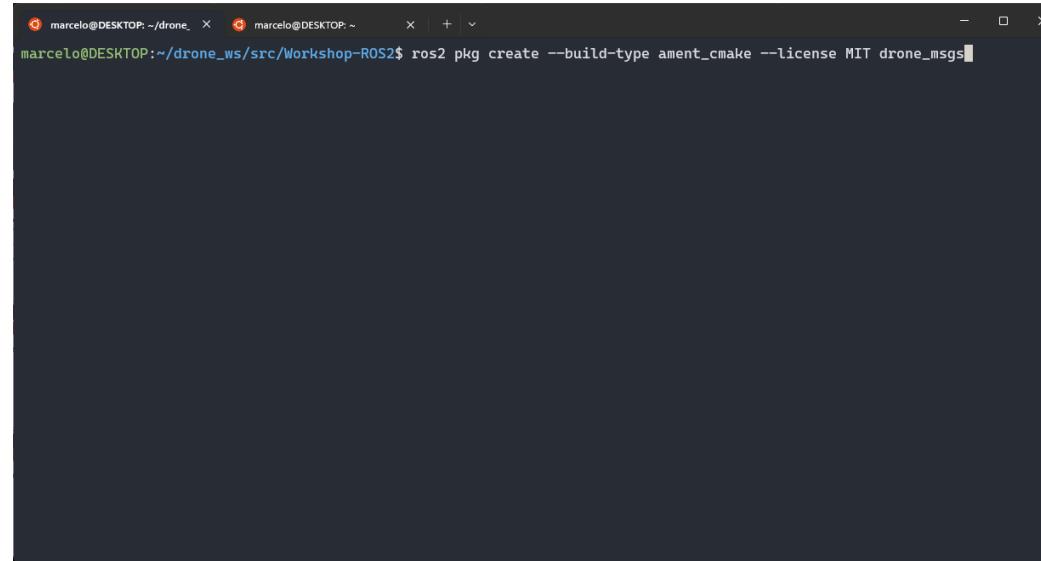
Creating a Messages Package



A screenshot of a terminal window titled "marcelo@DESKTOP: ~/drone_". The window contains a single command: "ros2 pkg create --build-type ament_cmake --license MIT drone_msgs". The terminal has two tabs open, both titled "marcelo@DESKTOP: ~".

```
ros2 pkg create --build-type ament_cmake --license MIT drone_msgs
```

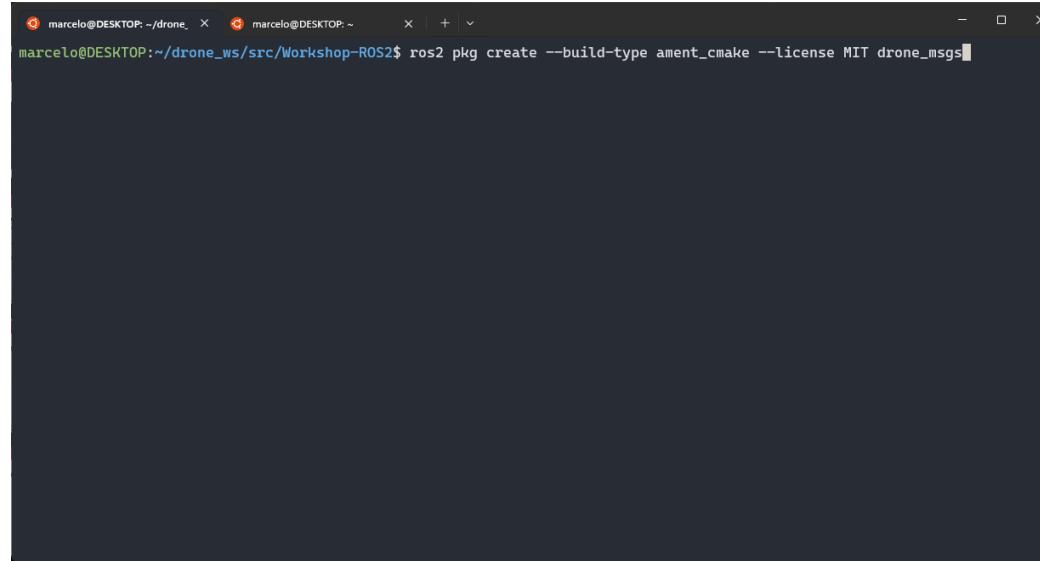
Creating a Messages Package



A screenshot of a terminal window titled "marcelo@DESKTOP: ~/drone_". The window shows a single command being typed: "ros2 pkg create --build-type ament_cmake --license MIT drone_msgs". The terminal has a dark background and light-colored text.

Ros2 pkg create –build-type ament_cmake --license MIT drone_msgs

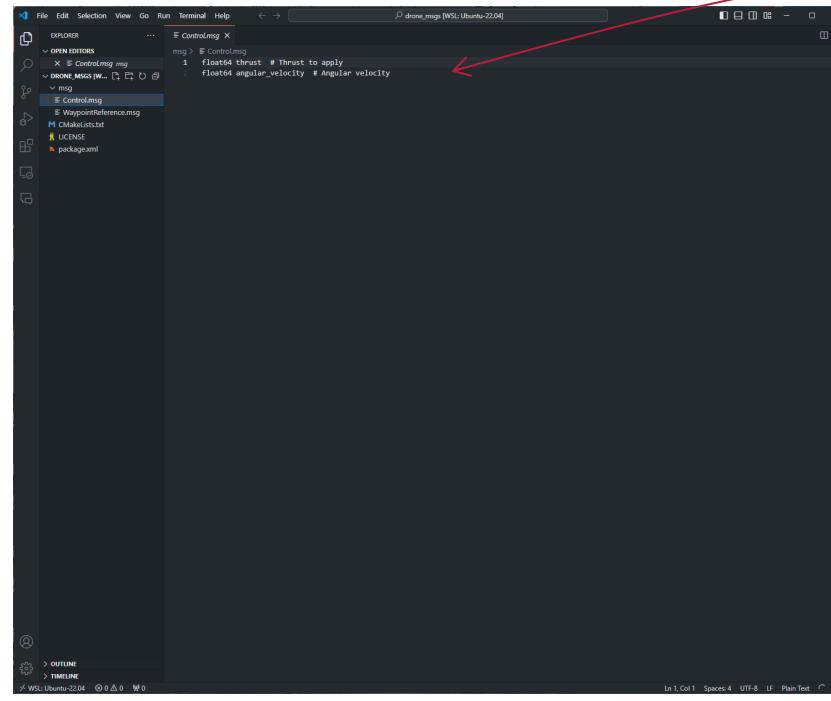
Creating a Messages Package



A screenshot of a terminal window titled "marcelo@DESKTOP: ~/drone_". The window shows a single command being typed: "ros2 pkg create --build-type ament_cmake --license MIT drone_msgs". The terminal has a dark background and light-colored text.

Ros2 pkg create –build-type ament_cmake --license MIT drone_msgs

Messages Provided by ROS 2



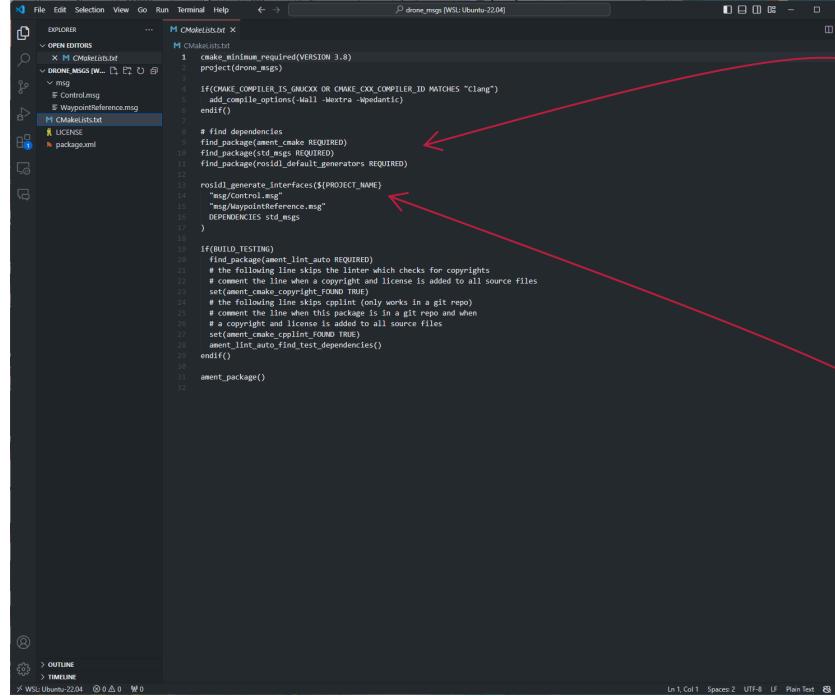
A screenshot of the Visual Studio Code (VS Code) interface, showing the code editor with a ROS 2 message definition. The file is named `Control.msg` and contains the following code:

```
msg > Controlmsg
1 float64 thrust # Thrust to apply
1 float64 angular_velocity # Angular velocity
```

A red arrow points from the text "Message definition" on the right side of the slide to the word "Controlmsg" in the code editor.

Message definition

Messages Provided by ROS 2



```
File Edit Selection View Go Run Terminal Help ↵ → drone_msgs (WSL:Ubuntu-22.04) □ □ □ □ □ ...  
OPEN EDITORS ...  
EXPLORER ...  
CMakeLists.txt  
drone_msgs (wsl:Ubuntu-22.04)  
msg  
Controls.msg  
WaypointReference.msg  
CMakeLists.txt  
LICENSE  
package.xml  
  
1 #ake_minimum_required(VERSION 3.0)  
2 project(drone_msgs)  
3  
4 if(CMAKE_COMPILER_IS_GCCXX OR CMAKE_CXX_COMPILER_ID MATCHES "Clang")  
5   add_compile_options(-Wall -Wextra -Wpedantic)  
6 endif()  
7  
8 # Find dependencies  
9 find_package(ament_cmake REQUIRED)  
10 find_package(std_msgs REQUIRED)  
11 find_package(rosidl_default_generators REQUIRED)  
12  
13 rosIDL_generate_interfaces(${PROJECT_NAME})  
14   "msg/Control.msg"  
15   "msg/WaypointReference.msg"  
16 DEPENDENCIES std_msgs  
17  
18 if(BUILD_TESTING)  
19   if(packageament_lint_auto REQUIRED)  
20     # The following line skips the linter which checks for copyrights  
21     # comment the line when a copyright and license is added to all source files  
22     set(ament_cmake_copyright_FOUND TRUE)  
23     # the following line skips cpplint (only works in a git repo)  
24     # comment the line when the package is in a git repo and when  
25     # a copyright and license is added to all source files  
26     set(ament_cmake_cpplint_FOUND TRUE)  
27     ament_lint_auto_find_test_dependencies()  
28   endif()  
29  
30 ament_package()  
31  
32
```

Import the base packages that the messages will use

List the messages to be compiled

Messages Provided by ROS 2

The screenshot shows a code editor with a red annotation highlighting a specific section of the XML code. The code is as follows:

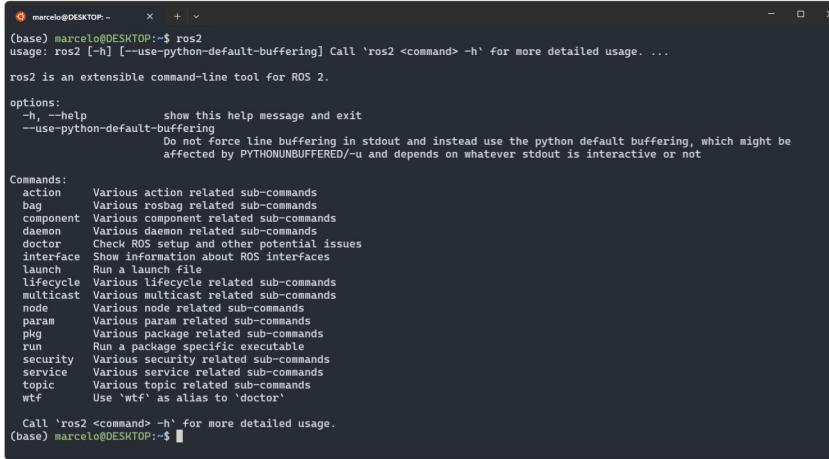
```
<?xml version="1.0"?>
<?xml-stylesheet href="http://download.ros.org/schema/package_format3.xsd" schematypens="http://www.w3.org/2001/XMLSchema"?>
<package format="3">
  <name>drone_msgs</name>
  <version>1.0.0</version>
  <description>ROS message definitions for the ROS 2 Workshop</description>
  <maintainer email="mjeiminho@tecnico.ulisboa.pt">Márcio Jeiminho</maintainer>
  <license>MIT</license>
<buildtool_depend>ament_cmake</buildtool_depend>
<depend>std_msgs</depend>
<buildtool_depend>rosidl_default_generators</buildtool_depend>
<exec_depend>rosidl_default_runtime</exec_depend>
<member_of_group>rosidl_interface_packages</member_of_group>
<test_depend>ament_lint_auto</test_depend>
<test_depend>ament_lint_common</test_depend>
<export>
  <build_type>ament_cmake</build_type>
</export>
</package>
```

A red arrow points from the top right towards the line containing the 'buildtool_depend' tag. Another red arrow points from the bottom right towards the line containing the 'exec_depend' tag.

Import the base packages that the messages will use

Terminal Tools

Terminal Tools



The terminal window shows the usage information for the ros2 command. It includes options for help (-h, --help), buffering (--use-python-default-buffering), and detailed command descriptions for various sub-commands like action, bug, component, daemon, doctor, interface, launch, lifecycle, multicast, node, param, pkg, run, security, service, topic, and wtf. A final note at the bottom suggests using 'ros2 <command> -h' for more detailed usage.

```
(base) marcelo@DESKTOP:~$ ros2
usage: ros2 [-h] [--use-python-default-buffering] Call `ros2 <command> -h` for more detailed usage. ...
ros2 is an extensible command-line tool for ROS 2.

options:
-h, --help            show this help message and exit
--use-python-default-buffering
                      Do not force line buffering in stdout and instead use the python default buffering, which might be
                      affected by PYTHONUNBUFFERED/-u and depends on whatever stdout is interactive or not

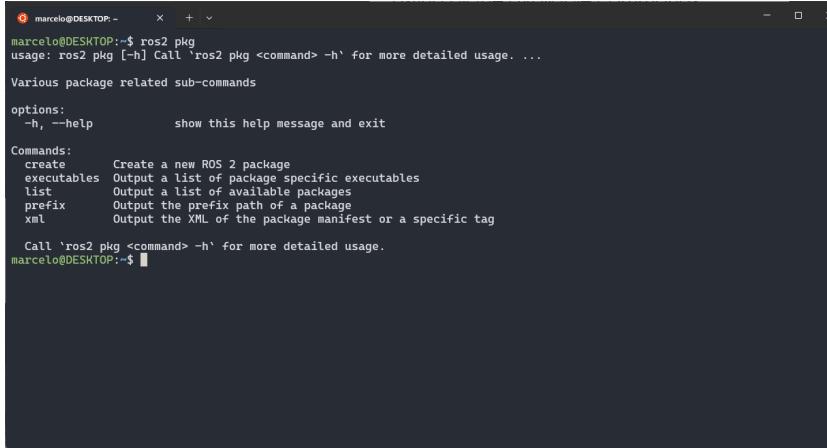
Commands:
action    Various action related sub-commands
bug      Various bug related sub-commands
component Various component related sub-commands
daemon   Various daemon related sub-commands
doctor   Check ROS setup and other potential issues
interface Show information about ROS interfaces
launch   Run a launch file
lifecycle Various lifecycle related sub-commands
multicast Various multicast related sub-commands
node     Various node related sub-commands
param    Various param related sub-commands
pkg      Various package related sub-commands
run      Run a package specific executable
security Various security related sub-commands
service  Various service related sub-commands
topic    Various topic related sub-commands
wtf      Use 'wtf' as alias to 'doctor'

Call `ros2 <command> -h` for more detailed usage.
(base) marcelo@DESKTOP:~$
```

Listing all the available commands:

`ros2`

Terminal Tools



A screenshot of a terminal window titled "marcelo@DESKTOP: ~". The user has run the command "ros2 pkg". The terminal displays the usage information and a detailed list of options and commands for the "pkg" tool.

```
marcelo@DESKTOP:~$ ros2 pkg
usage: ros2 pkg [-h] Call `ros2 pkg <command> -h` for more detailed usage. ...
Various package related sub-commands

options:
-h, --help      show this help message and exit

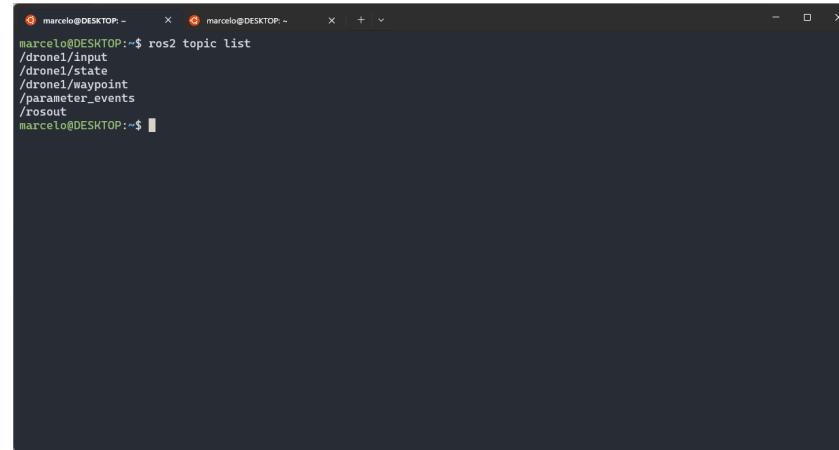
Commands:
create          Create a new ROS 2 package
executables     Output a list of package specific executables
list            Output a list of available packages
prefix          Output the prefix path of a package
xml             Output the XML of the package manifest or a specific tag

Call `ros2 pkg <command> -h` for more detailed usage.
marcelo@DESKTOP:~$
```

Each command can have multiple options. For example:

ros2 pkg

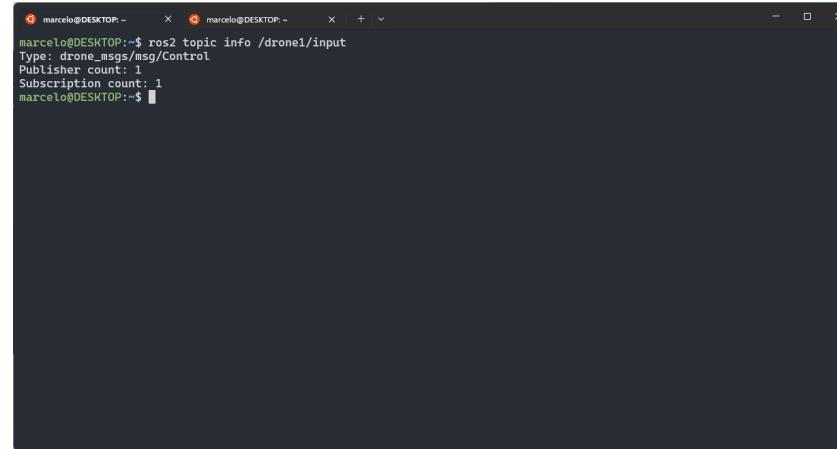
List all Topics Published/Subscribed

A screenshot of a terminal window titled "marcelo@DESKTOP: ~". It displays the command "ros2 topic list" followed by a list of topics: "/drone1/input", "/drone1/state", "/drone1/waypoint", "/parameter_events", and "/rosout".

```
marcelo@DESKTOP:~$ ros2 topic list
/drone1/input
/drone1/state
/drone1/waypoint
/parameter_events
/rosout
marcelo@DESKTOP:~$
```

ros2 topic list

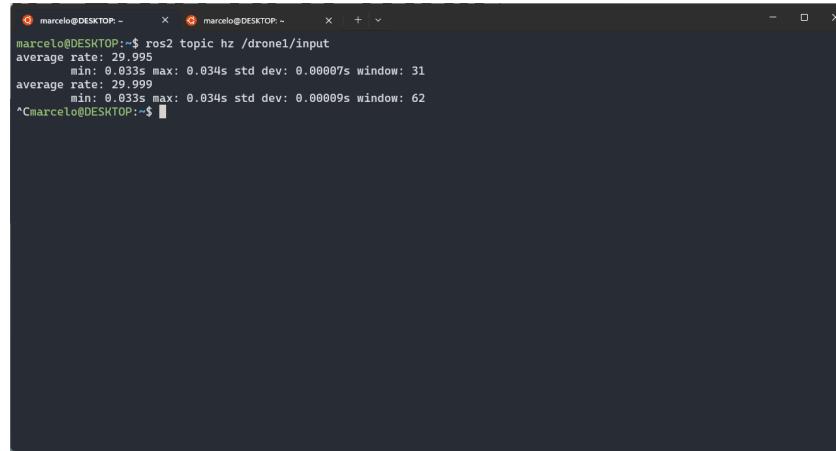
Get Information About a Topic

A screenshot of a Linux terminal window titled "marcelo@DESKTOP: ~". The window contains the following text:

```
marcelo@DESKTOP:~$ ros2 topic info /drone1/input
Type: drone_msgs/msg/Control
Publisher count: 1
Subscription count: 1
marcelo@DESKTOP:~$
```

`ros2 topic info /<name_of_topic>`

Get the Rate of a Topic

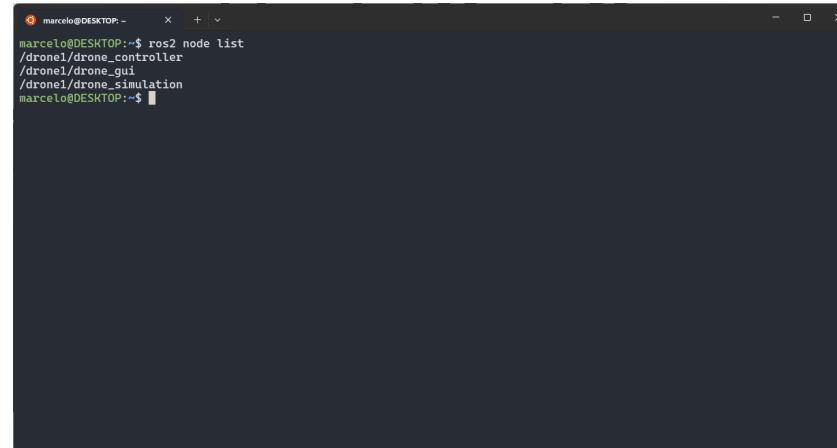


A screenshot of a terminal window titled "marcelo@DESKTOP: ~". It displays the command "ros2 topic hz /drone1/input" followed by its execution results. The results show two sets of statistics: one for a window of 31 messages and another for a window of 62 messages. Both windows show an average rate of 29.995 Hz, a minimum rate of 0.033s, and a maximum rate of 0.034s. The standard deviation for both windows is 0.00007s.

```
marcelo@DESKTOP:~$ ros2 topic hz /drone1/input
average rate: 29.995
    min: 0.033s max: 0.034s std dev: 0.00007s window: 31
average rate: 29.999
    min: 0.033s max: 0.034s std dev: 0.00009s window: 62
^Cmarcelo@DESKTOP:~$
```

`ros2 topic hz /<name_of_topic>`

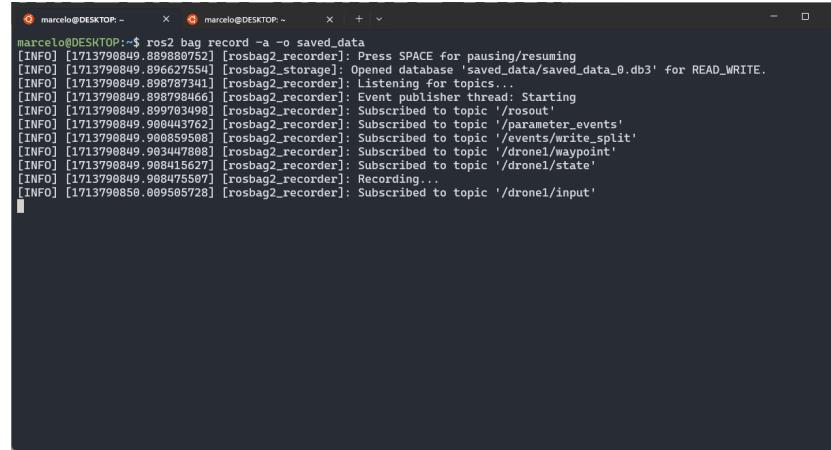
Get all the Nodes Running

A screenshot of a terminal window titled "marcelo@DESKTOP: ~". The window contains the command "ros2 node list" followed by its output: "/drone1/drone_controller", "/drone1/drone_gui", and "/drone1/drone_simulation". The terminal has a dark background and white text.

```
marcelo@DESKTOP: ~
marcelo@DESKTOP:~$ ros2 node list
/drone1/drone_controller
/drone1/drone_gui
/drone1/drone_simulation
marcelo@DESKTOP:~$
```

`ros2 node list`

Saving Data using Bags

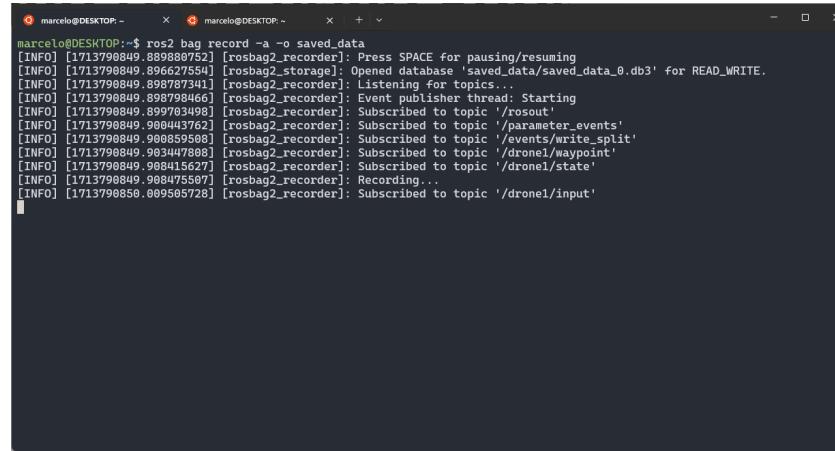


A screenshot of a terminal window titled 'marcelo@DESKTOP: ~'. The window contains the command 'ros2 bag record -a -o saved_data' followed by several lines of log output from the 'rosbag2_recorder' node. The log output shows the recorder opening a database, listening for topics, and subscribing to various ROS topics including '/rosout', '/parameter_events', '/events/write_split', '/drone1/waypoint', '/drone1/state', and '/drone1/input'. It also indicates that recording has started.

```
marcelo@DESKTOP:~$ ros2 bag record -a -o saved_data
[INFO] [1713790849.889880752] [rosbag2_recorder]: Press SPACE for pausing/resuming
[INFO] [1713790849.896627554] [rosbag2_storage]: Opened database 'saved_data/saved_data_0.db3' for READ_WRITE.
[INFO] [1713790849.898787341] [rosbag2_recorder]: Listening for topics...
[INFO] [1713790849.898798466] [rosbag2_recorder]: Event publisher thread: Starting
[INFO] [1713790849.899783498] [rosbag2_recorder]: Subscribed to topic '/rosout'
[INFO] [1713790849.900443762] [rosbag2_recorder]: Subscribed to topic '/parameter_events'
[INFO] [1713790849.900859508] [rosbag2_recorder]: Subscribed to topic '/events/write_split'
[INFO] [1713790849.903447888] [rosbag2_recorder]: Subscribed to topic '/drone1/waypoint'
[INFO] [1713790849.908415627] [rosbag2_recorder]: Subscribed to topic '/drone1/state'
[INFO] [1713790849.908475507] [rosbag2_recorder]: Recording...
[INFO] [1713790850.009505728] [rosbag2_recorder]: Subscribed to topic '/drone1/input'
```

`ros2 bag record -a -o <bag_name>`

Playing Back the Bag Data



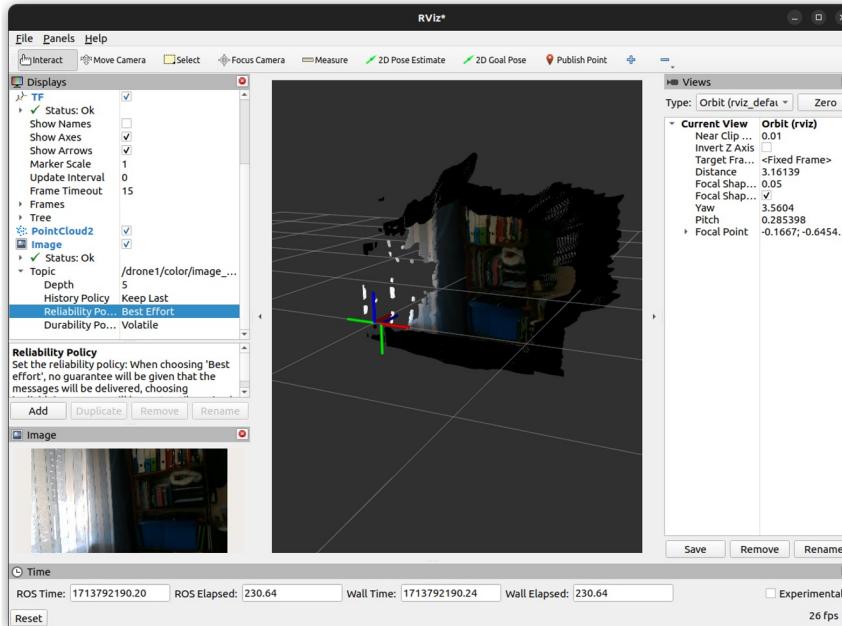
A terminal window titled 'marcelo@DESKTOP: ~' showing the output of the 'ros2 bag record' command. The command is used to record ROS messages from topics into a bag file. The output shows the recorder opening a database, listening for topics, and subscribing to various ROS topics like /rosout, /parameter_events, /events/write_split, /drone1/waypoint, /drone1/state, and /drone1/input.

```
[INFO] [17137998849.889880752] [rosbag2_recorder]: Press SPACE for pausing/resuming
[INFO] [17137998849.896627554] [rosbag2_storage]: Opened database 'saved_data/saved_data_0.db3' for READ_WRITE.
[INFO] [17137998849.898787341] [rosbag2_recorder]: Listening for topics...
[INFO] [17137998849.898798466] [rosbag2_recorder]: Event publisher thread: Starting
[INFO] [17137998849.899763498] [rosbag2_recorder]: Subscribed to topic '/rosout'
[INFO] [17137998849.900443762] [rosbag2_recorder]: Subscribed to topic '/parameter_events'
[INFO] [17137998849.900859588] [rosbag2_recorder]: Subscribed to topic '/events/write_split'
[INFO] [17137998849.903447888] [rosbag2_recorder]: Subscribed to topic '/drone1/waypoint'
[INFO] [17137998849.9084115627] [rosbag2_recorder]: Subscribed to topic '/drone1/state'
[INFO] [17137998849.908475587] [rosbag2_recorder]: Recording...
[INFO] [17137998850.009505728] [rosbag2_recorder]: Subscribed to topic '/drone1/input'
```

`ros2 bag play <bag_name>`

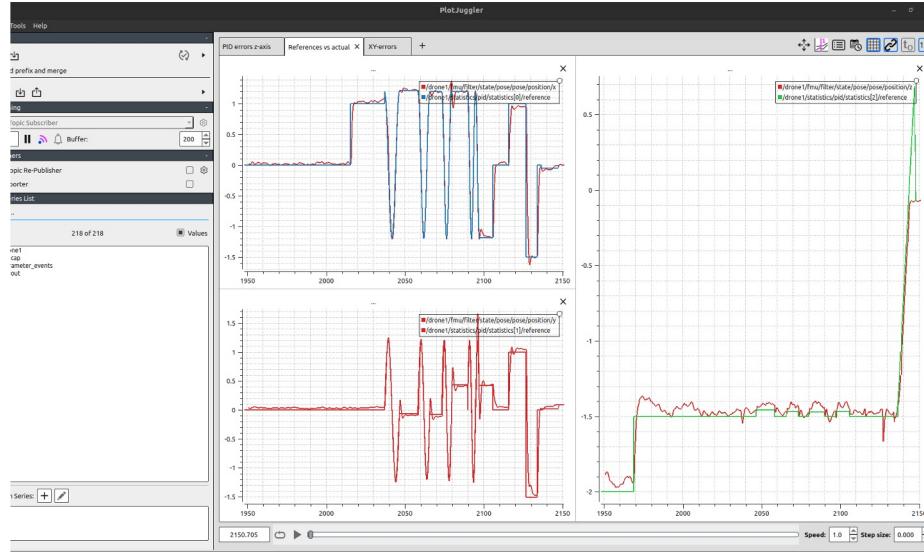
Visualization Tools

RVIZ 2



Executing: rviz2

Plotjuggler



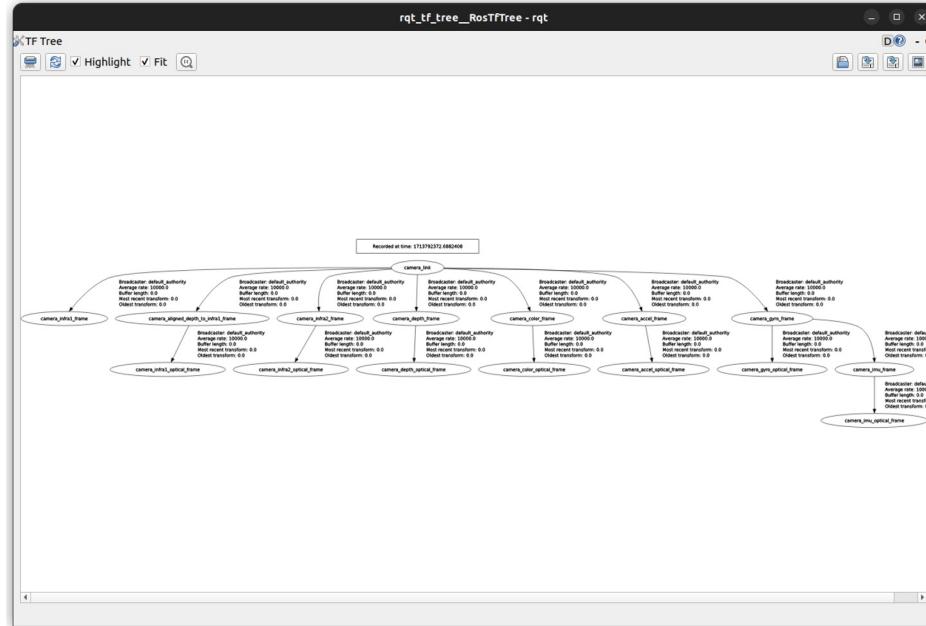
Installing:

```
sudo apt install ros-humble-plotjuggler
```

Executing:

```
ros2 run plotjuggler plotjuggler
```

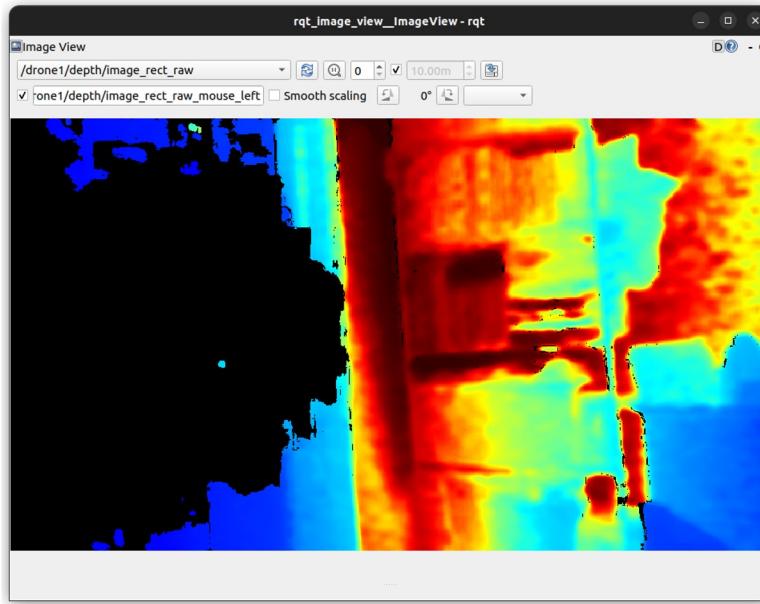
Transforms (TF2)



Executing:

```
ros2 run rqt_tf_tree rqt_tf_tree
```

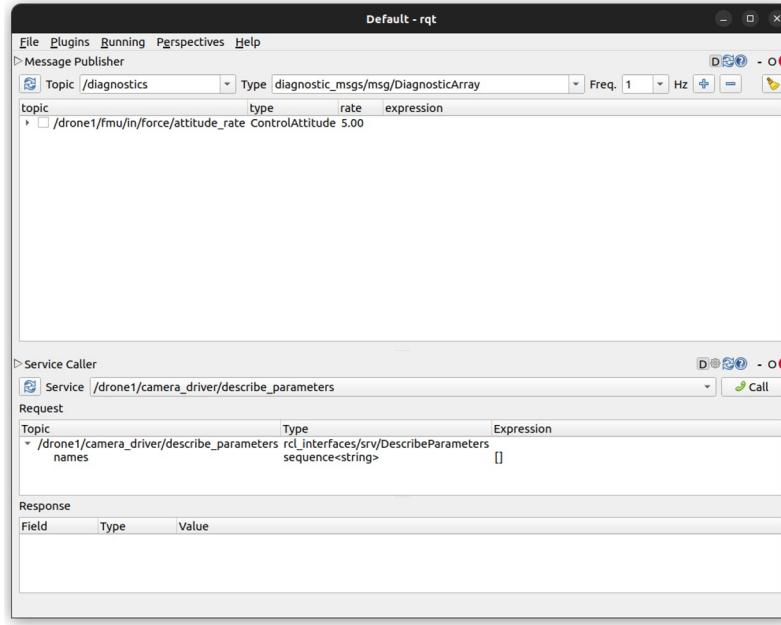
RQT Image View



Executing:

```
ros2 run rqt_image_view rqt_image_view
```

RQT GUI



Executing:

```
ros2 run rqt_gui rqt_gui
```

ROS ID

- By default, all the ROS 2 nodes run with domain ID 0
- To avoid interference between different groups of computers running ROS 2 on the same network, a different domain ID should be set for each group

```
export ROS_DOMAIN_ID=0
```

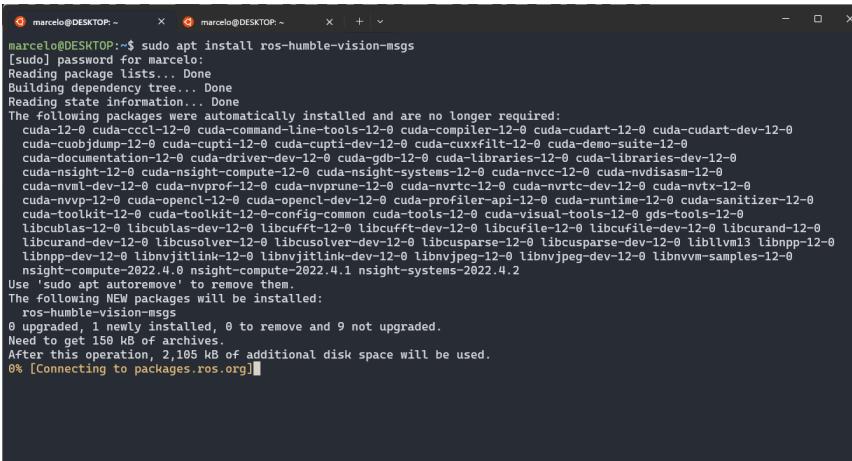
- By default, all the ROS 2 nodes broadcast to the local network

```
export ROS_LOCALHOST_ONLY=0
```

Installing External Libraries

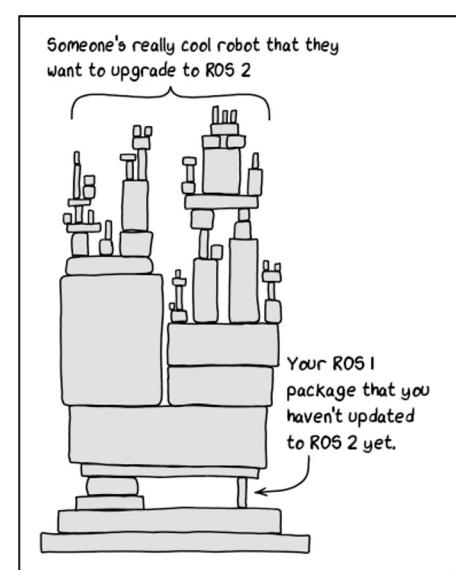
Installing packages follows this standard:

```
sudo apt install ros-<version>-<name-of-package>
```



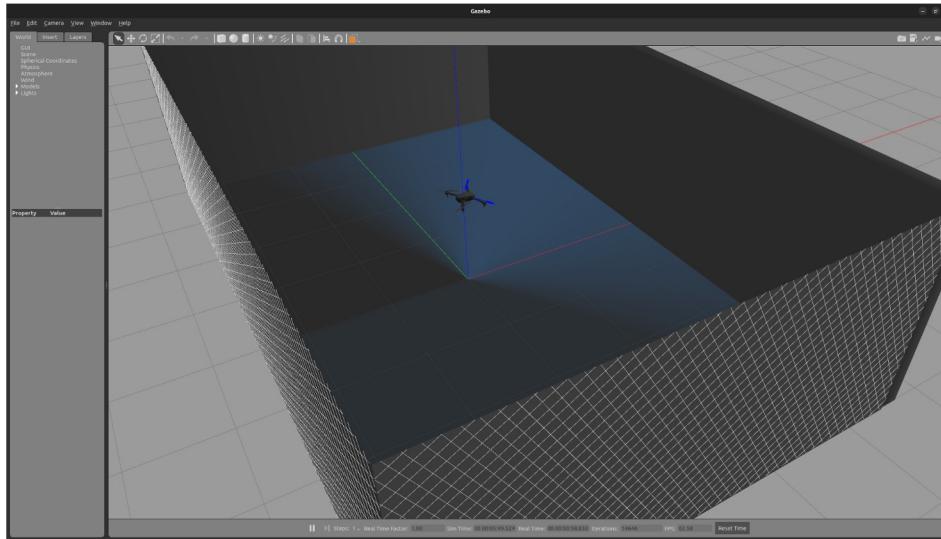
```
marcelo@DESKTOP:~ x marcelo@DESKTOP:~ x + ~
[sudo] password for marcelo:
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following packages were automatically installed and are no longer required:
  cuda-12-0 cuda-cccl-12-0 cuda-command-line-tools-12-0 cuda-compiler-12-0 cuda-cudart-12-0 cuda-cudart-dev-12-0
  cuda-cuboiddump-12-0 cuda-cupti-12-0 cuda-cupti-dev-12-0 cuda-cuxfilt-12-0 cuda-demo-suite-12-0
  cuda-documentation-12-0 cuda-driver-dev-12-0 cuda-gdb-12-0 cuda-libraries-12-0 cuda-libraries-dev-12-0
  cuda-nsight-12-0 cuda-nsight-compute-12-0 cuda-nsight-systems-12-0 cuda-nvc-12-0 cuda-nvdisasm-12-0
  cuda-nvml-dev-12-0 cuda-nvprof-12-0 cuda-nvprune-12-0 cuda-nvrtc-12-0 cuda-nvrtc-dev-12-0 cuda-nvtx-12-0
  cuda-nvvp-12-0 cuda-opencl-12-0 cuda-profile-api-12-0 cuda-runtime-12-0 cuda-sanitizer-12-0
  cuda-toolkit-12-0 cuda-toolkit-12-0-config-common cuda-tools-12-0 cuda-visual-tools-12-0 gds-tools-12-0
  libcublas-12-0 libcublas-dev-12-0 libcurlf-dev-12-0 libcurlf-12-0 libcurlf-dev-12-0 libcurlf-12-0 libcurlr-12-0
  libcurlr-dev-12-0 libcurlsolver-12-0 libcurlsolver-dev-12-0 libcurlsparse-12-0 libcurlsparse-dev-12-0 liblmlm13 libnpp-12-0
  libnpp-dev-12-0 libnvjittlink-12-0 libnvjittlink-dev-12-0 libnvjpeg-12-0 libnvjpeg-dev-12-0 libnvvm-samples-12-0
  nsight-compute-2022.4.0 nsight-compute-2022.4.1 nsight-systems-2022.4.2
Use 'sudo apt autoremove' to remove them.
The following NEW packages will be installed:
  ros-humble-vision-msgs
0 upgraded, 1 newly installed, 0 to remove and 9 not upgraded.
Need to get 150 kB of additional disk space will be used.
After this operation, 2,105 kB of additional disk space will be used.
0% [Connecting to packages.ros.org]
```

Example for the `vision_msgs` package



3D Simulators

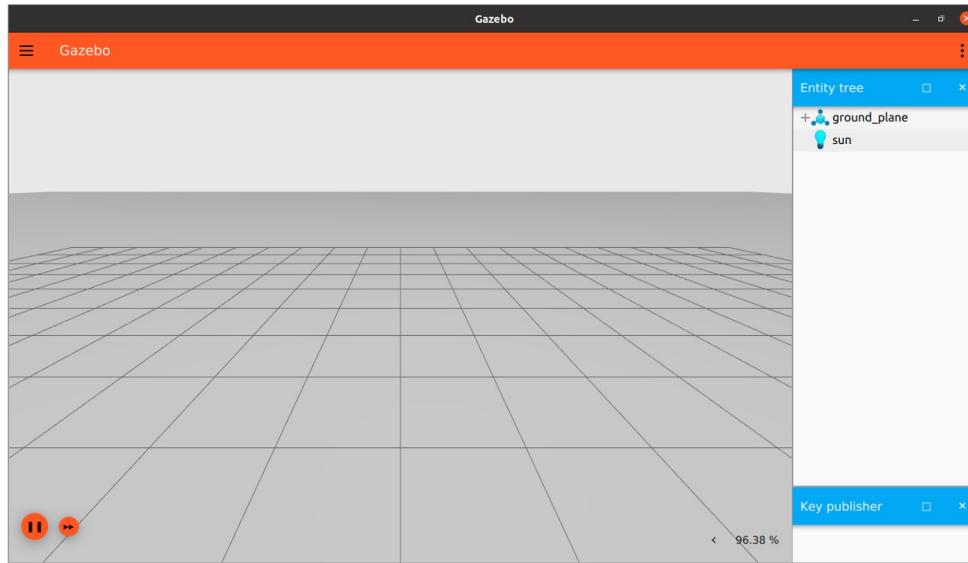
Simulators (Gazebo Classic)



Installing

```
sudo apt install ros-humble-gazebo-ros-pkgs
```

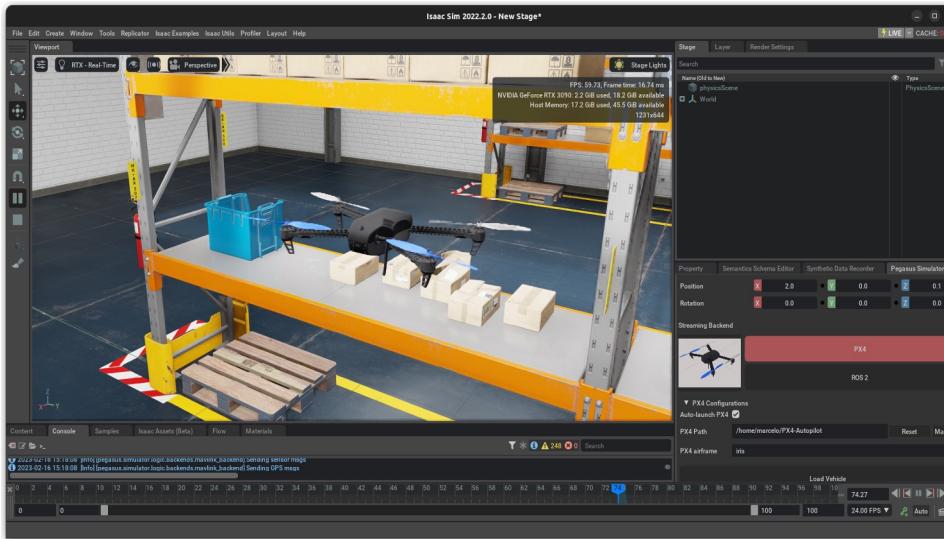
Simulators (Gazebo Garden)



Installing

```
sudo apt-get install ros-humble-ros-gz
```

Simulators (Pegasus Simulator)





Institute for Systems
and Robotics | LISBOA



TÉCNICO
LISBOA