ROS2

Ilia Nechaev

18 October 2024

What is ROS?

[Wikipedia] Robot Operating System (ROS or ros) is an open-source robotics middleware suite. Although ROS is not an operating system (OS) but a set of software frameworks for robot software development, it provides services designed for a heterogeneous computer cluster such as hardware abstraction, low-level device control, implementation of commonly used functionality, message-passing between processes, and package management.

[ros.org] The Robot Operating System (ROS) is a set of software libraries and tools that help you build robot applications.

What can ROS do?

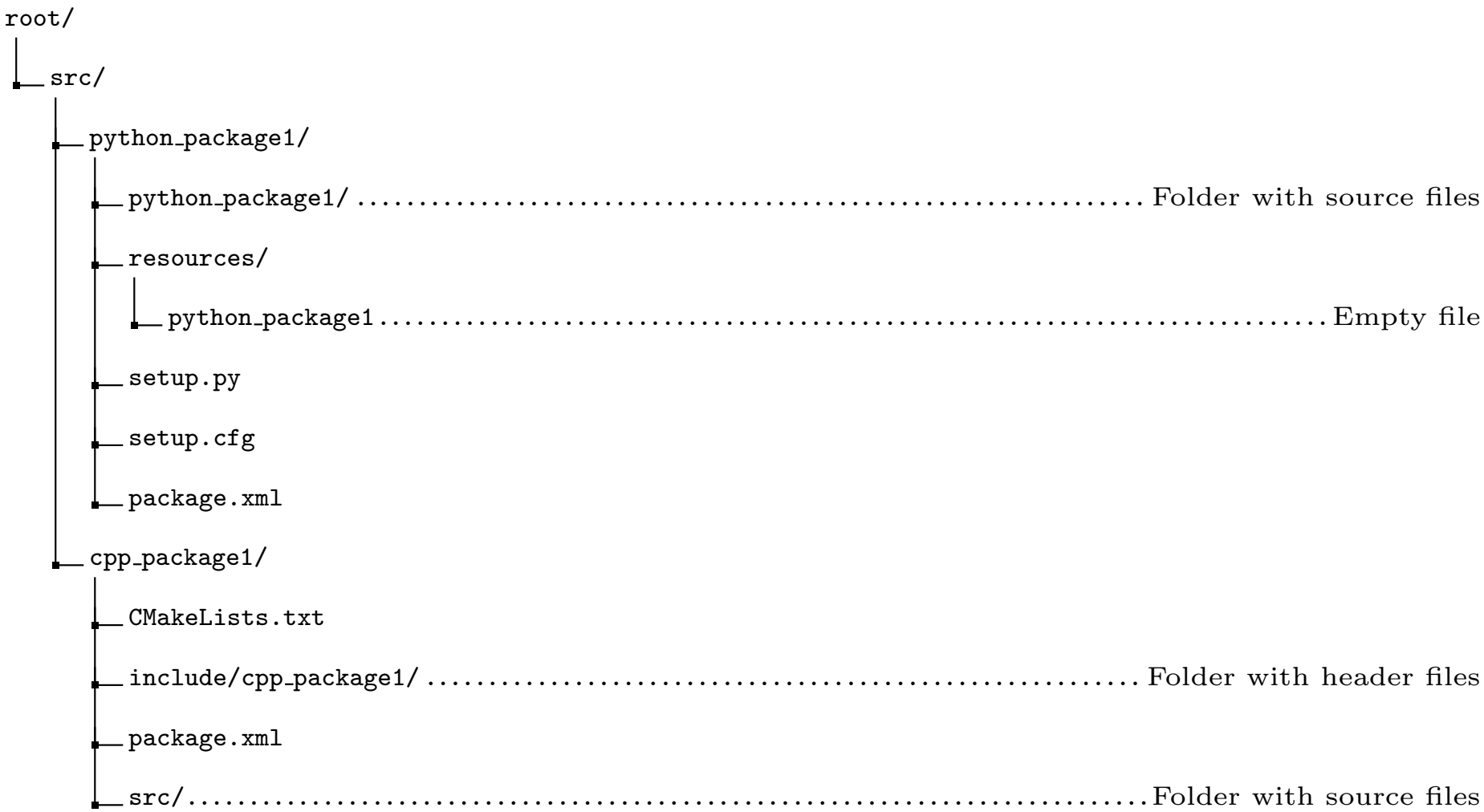
- Tools for creating and managing nodes (something like microservices)
  - Messages with callbacks
  - Services
- Utilities for working with time, parameters, logging, and more
- Utilities for debugging, visualization, and process management
- Nodes orchestration
- Package management
- Multiplatform support
- A lot of other things

ROS2

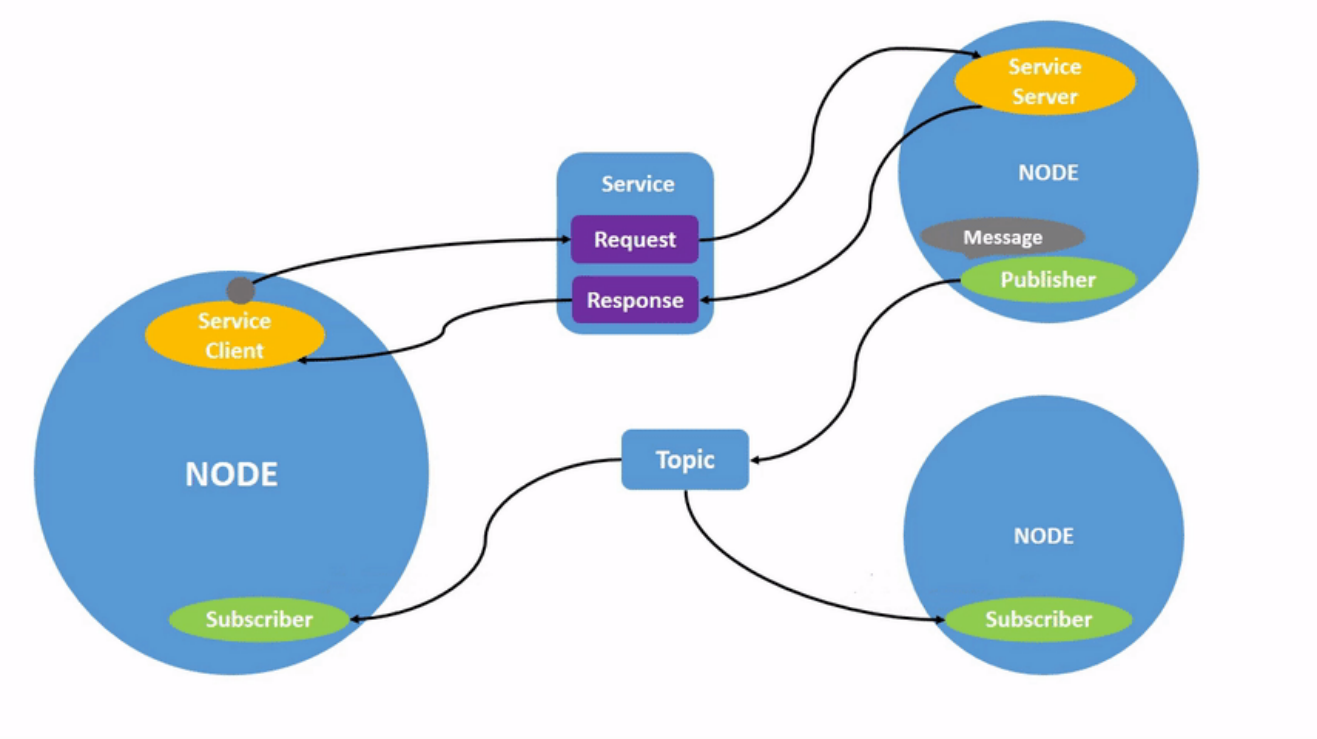
[docs.ros.org] Since ROS was started in 2007, a lot has changed in the robotics and ROS community. The goal of the ROS 2 project is to adapt to these changes, leveraging what is great about ROS 1 and improving what isn't.

ROS2 project structure

Each ROS2 project starts with creating workspace. The folder with special structure of subfolders. Minimal example:



Understanding nodes



Each node in ROS should be responsible for a single, modular purpose, e.g. controlling the wheel motors or publishing the images from camera. Each node can send and receive data from other nodes via topics, services, or actions

Simple node example

```python
import rclpy
from rclpy.node import Node

class SimpleNode(Node):
    def __init__(self):
        super(SimpleNode, self).__init__('simple_node')
        timer_period = 1
        self.timer = self.create_timer(timer_period, self.timer_callback)

    def timer_callback(self):
        self.get_logger().info("Hello world!")


def main():
    rclpy.init()
    node = SimpleNode()
    rclpy.spin(node)
    rclpy.shutdown()


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

Understanding topics

```python
import rclpy
from std_msgs.msg import String
from rclpy.node import Node

class TopicsNode(Node):
    def __init__(self):
        super(TopicsNode, self).__init__('any_name')
        self.publisher = self.create_publisher(String, 'sender', 10)
        timer_period = 0.5
        self.timer = self.create_timer(timer_period, self.timer_callback)
        self.subscriber = self.create_subscription(String, 'receiver',
            self.subscriber_callback, qos_profile=10)

    def subscriber_callback(self, msg):
        self.get_logger().info(f"Received: {msg.data}")

    def timer_callback(self):
        msg = String()
        msg.data = "Hello ROS2"
        self.publisher.publish(msg)


def main():
    rclpy.init()
    node = TopicsNode()
    rclpy.spin(node)
    rclpy.shutdown()

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

Understanding services

```python
import rclpy
from rclpy.node import Node
from std_srvs.srv import Trigger

class ServiceNode(Node):
    def __init__(self):
        super(ServiceNode, self).__init__('service_node')
        self.client = self.create_client(Trigger, "/service")
        self.string_to_store = ""
        self.service = self.create_service(Trigger, "/other_service",
             self.service_callback)
        while not self.client.wait_for_service(timeout_sec=1.0):
            self.get_logger().info(f'service "service" not available,
                 waiting again...')
        future = self.client.call_async(Trigger.Request())
        self.get_logger().info("Waiting for service...")
        rclpy.spin_until_future_complete(self, future)
        self.string_to_store = future.result().message

    def service_callback(self, request, response):
        response.success = True
        response.message = self.string_to_store
        return response

def main():
    rclpy.init()
    node = ServiceNode()
    rclpy.spin(node)
    rclpy.shutdown()

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

ROS2 messages and services

Messages and services are just lists of data types that they contain.
Available types for messages:

- bool
- byte
- char
- float[32|64]
- [u]int[8|16|32|64]
- string
- Other messages

You can also use arrays both static and fixed:

- int8[]
- int16[5]
- uint32[<=5]

ROS2. Parameters

You can pass parameters to nodes via yaml config file or as cli parameters in launch script.

```yaml
/**:
  ros__parameters:
    string: "Hello from parameters"
    number: 42
    floating_point: 2.39
```

Accessing parameters in node:

```python
import rclpy
from rclpy.node import Node

class ParameterNode(Node):
    def __init__(self):
        super(ParameterNode, self).__init__('parameter_node')
        self.declare_parameter("string", "")
        self.string_parameter = self.get_parameter("string").
            get_parameter_value().string_value
        self.get_logger().info(f"String: {self.string_parameter}")
        self.declare_parameter("number", 0)
        self.number_parameter = self.get_parameter("number").
            get_parameter_value().integer_value
        self.get_logger().info(f"Number: {self.number_parameter}")

def main():
    rclpy.init()
    node = ParameterNode()
    rclpy.spin_once(node)
    rclpy.shutdown()

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

ROS2. Building. package.xml

```
<?xml version="1.0"?>
<?xml-model
   href="http://download.ros.org/schema/package_format3.xsd"
   schematypens="http://www.w3.org/2001/XMLSchema"?>
<package format="3">
<name>example</name>
    <version>0.0.0</version>
    <description>TODO: Package description</description>
    <maintainer email="user@todo.todo">user</maintainer>
    <license>TODO: License declaration</license>

    <test_depend>ament_copyright</test_depend>
    <test_depend>ament_flake8</test_depend>
    <test_depend>ament_pep257</test_depend>
    <test_depend>python3-pytest</test_depend>

    <export>
        <build_type>ament_python</build_type>
    </export>

    <depend>your_package_in_rosdep</depend> <!--Optional-->

</package>
```

ROS2. Building. setup.py

```python
from setuptools import find_packages, setup

package_name = 'example'


setup(
    name=package_name,
    version='0.0.0',
    packages=find_packages(exclude=['test']),
    data_files=[
        ('share/ament_index/resource_index/packages',
            ['resource/' + package_name]),
        ('share/' + package_name, ['package.xml']),
        ('share/' + package_name + '/config', ['config/example.yaml']),
    ],
    install_requires=['setuptools'],
    zip_safe=True,
    maintainer='maintainer.name',
    maintainer_email='maintainer@email.com',
    description='TODO: Package description',
    license='TODO',
    tests_require=['pytest'],
    entry_points={
        'console_scripts': [
            'example = example.parameter_node:main',
        ],
    },
)
```

ROS2. Building

To build project:

1. `source /opt/ros/<ros_distro>/setup.bash`
2. `colcon build`
3. `rosdep init`
4. `rosdep update`
5. `rosdep install -y --from-path ./src --rosdistro=${ROS_DISTRO}`

But usually you build in docker.

```
1  FROM ros:<ros_distro>
2  ARG ROS_DISTRO=<ros_distro>
3
4  SHELL ["/bin/bash", "-c"]
5
6  RUN apt update
7  RUN apt install -y python3 python3-pip
8
9  WORKDIR /app
10
11 COPY requirements.txt /app
12 RUN pip3 install -r requirements.txt
13
14 COPY src /app/src
15
16 RUN rosdep init
17 RUN rosdep update
18 RUN rosdep install -y --from-path ./src --rosdistro=${ROS_DISTRO}
19
20 RUN source /opt/ros/${ROS_DISTRO}/setup.bash && colcon build
```

ROS2. Starting node

To start node:

1. Open new terminal (it mustn't be the same terminal where you built the project)
2. Source ROS2 environment with `source /opt/ros/<ros_distro>/setup.bash`
3. Source local environment with `source ./install/setup.bash`
4. Run node with `ros2 run example example`
5. If you want to pass parameters use

   ```
   ros2 run example example --ros-args -p string:=Hello -p number:=42
   ```

6. If you want to pass parameters from yaml file use

   ```
   ros2 run example example --ros-args --params-file path/to/parameters.yaml
   ```

7. To stop node press Ctrl+C

ROS2. Launching nodes with launch file

ROS2 provides a way to launch multiple nodes with a single command. But to do this you need to create launch file. You have three options:

- XML
- Python
- YAML

```
<launch>
    <arg name="string" default="hi" />

    <node
        pkg="example"
        exec="example"
        output="screen"
    >
    <param name="string" value="$(var string)" />
    <param from="$(find-pkg-share example)/config/example.yaml"/>
    </node>
</launch>
```

ROS2. Launching nodes with launch file

To launch nodes with launch file:

1. Open new terminal (it mustn't be the same terminal where you built the project)
2. Source ROS2 environment with `source /opt/ros/<ros_distro>/setup.bash`
3. Source local environment with `source ./install/setup.bash`
4. Run node with `ros2 launch example example.launch`
5. If you want to pass parameters use

   ```
   ros2 run example example.launch string:="Hi, ROS2!"
   ```

6. To stop nodes press Ctrl+C

ROS2. CLI tools

ROS2 provides multiple CLI tools for developing, debugging, running and monitoring ROS2 nodes

- ros2 run
- ros2 launch
- ros2 node [list|info]
- ros2 topic [echo|find|hz|info|list|pub|type]
- ros2 service [call|find|call|type]

To list all available commands: ros2 -h

ROS2. rqt_graph

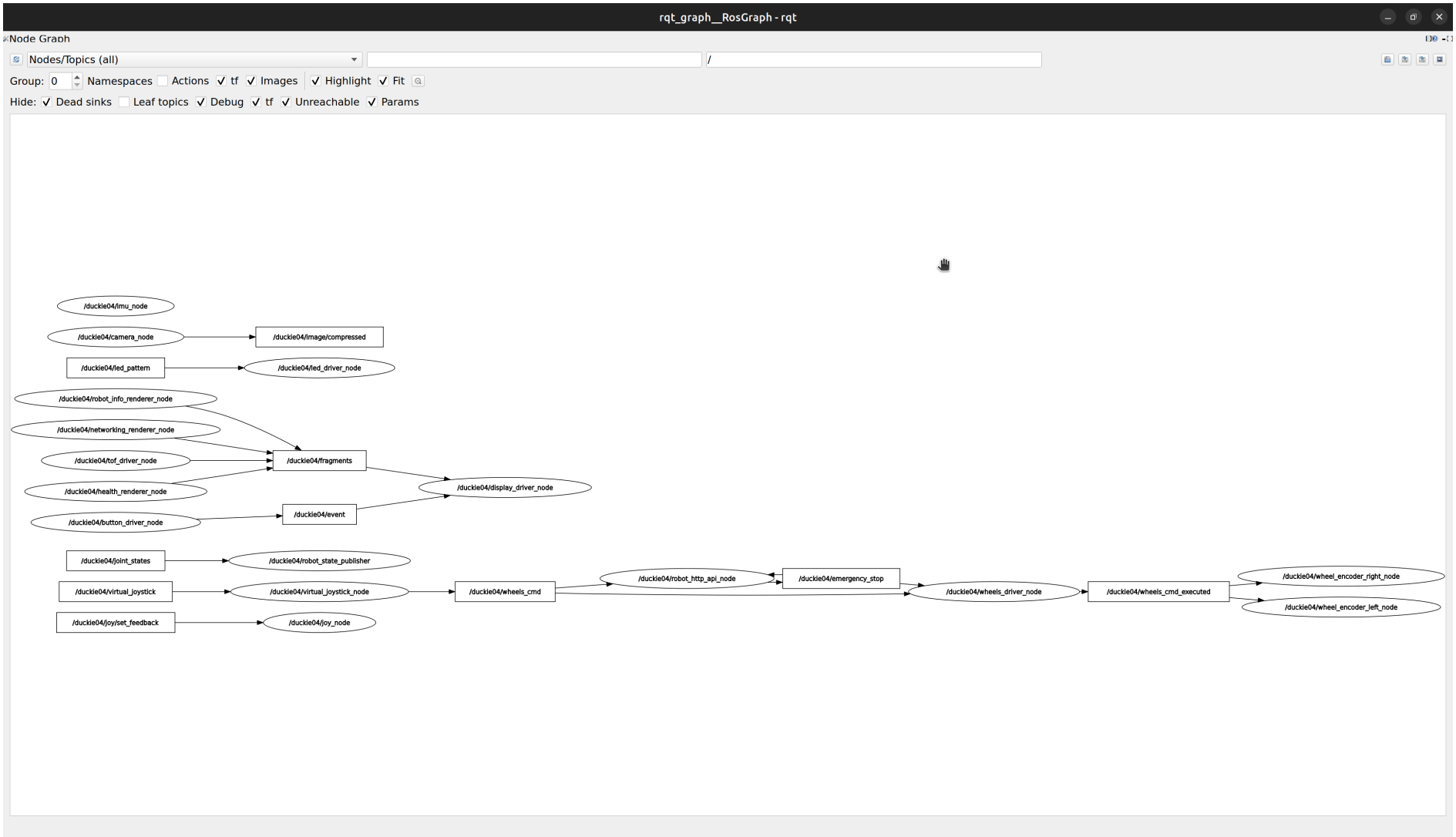


Figure: rqt_graph

ROS2. rqt_image_view

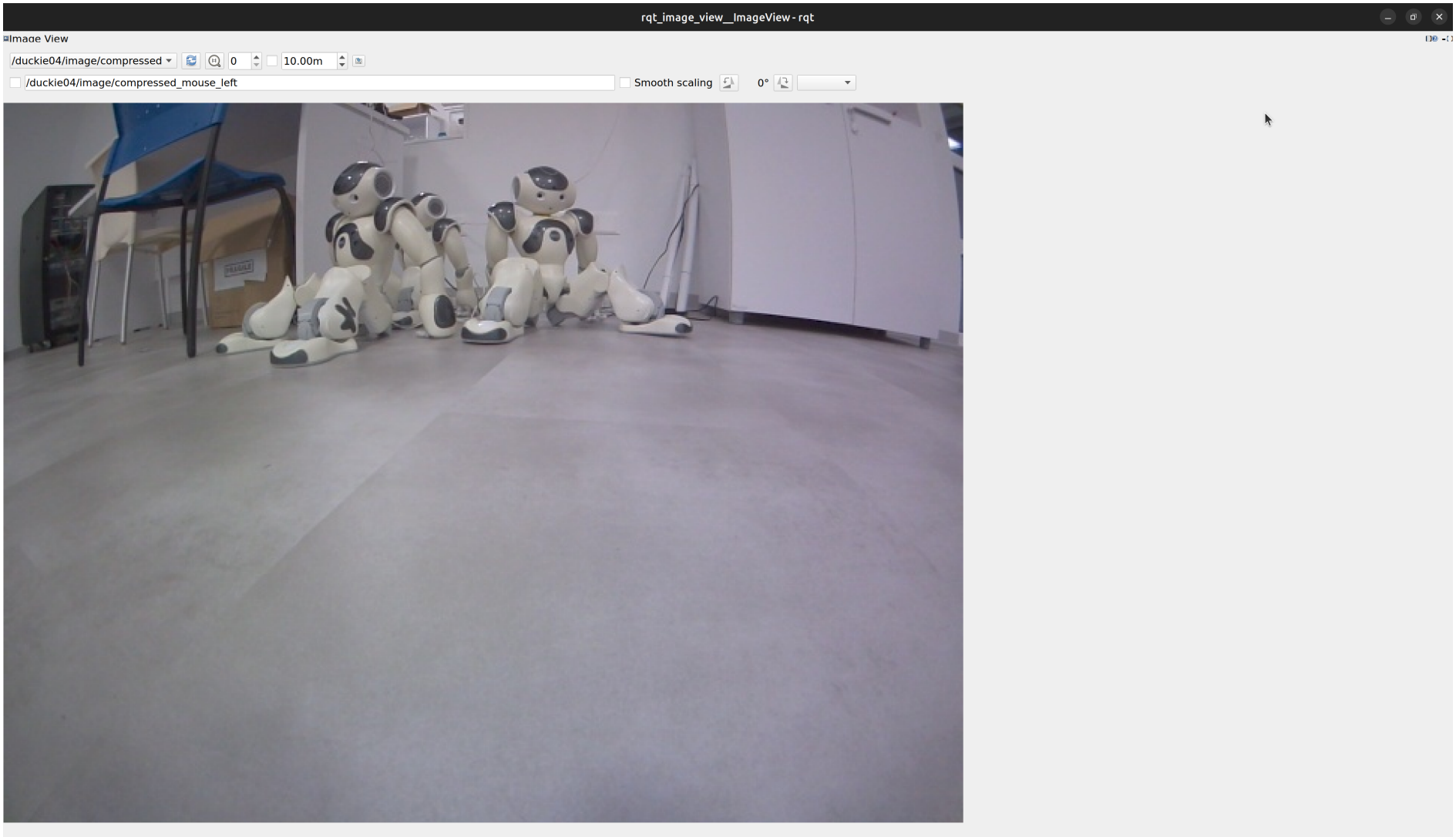


Figure: rqt_image_view

ROS2. Creating custom messages

srv/example_service.srv

```
int64 a
int64 b
std_msgs/msgs/String s
---
int64 sum
std_msgs/msgs/Bool result
```

package.xml

```

<!--...-->
  <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>
<!--...-->
```

CMakeLists.txt

```
cmake_minimum_required(VERSION 3.8)
project(tutorial_interfaces)

if(CMAKE_COMPILER_IS_GNUCXX OR
      CMAKE_CXX_COMPILER_ID MATCHES "
      Clang")
  add_compile_options(-Wall -Wextra -
        Wpedantic)
endif()

find_package(ament_cmake REQUIRED)
find_package(std_msgs REQUIRED)
find_package(rosidl_default_generators
       REQUIRED)

rosidl_generate_interfaces(${
      PROJECT_NAME}
  "srv/example_service.srv"
  DEPENDENCIES std_msgs
)

ament_package()
```