# (Murilo's) ROS2 Tutorial

***Release May 30, 2026***

**Murilo M. Marinho**

**May 30, 2026**

# PREAMBLE

**Note**

If you're looking for the official documentation, this is **NOT** it. For the official ROS documentation, refer to this link.

**Hint**

You can download this tutorial as a PDF .

**License**




**Attention**

AI training with this content must follow the license restrictions. In particular, please refer to the excerpt below.

*If AI training data includes the NonCommercial restriction, then following the NC restriction would require that all stages, from copying the data during training to sharing the trained model, must not be for commercial gain.*

*Following the NoDerivatives restriction would require that ND-licensed content not be used as training data.*

**About this tutorial**

ROS2 Jazzy tutorials by Murilo M. Marinho, focusing on Ubuntu 24.04 x64 LTS and the programming practices of successful state-of-the-art robotics implementations such as the SmartArmStack also used in the AISciencePlatform.

These tutorials have been the backbone of EEEN62021 Software for Robotics, one of the units of the MSc Robotics at the University of Manchester.

**Using this tutorial**

This is a tutorial that supposes that the user will follow it linearly. Some readers can skip the *Preamble* if they are somewhat already comfortable in Python and Ubuntu. Otherwise, all steps can be considered as dependent on the prior ones, starting from *ROS2 Setup*.

It is expected that the user will be working directly on an Ubuntu machine. **`Docker`** images are available but no compatibility with host systems other than Ubuntu have been attempted.

**Ways to show love**

If you enjoyed this tutorial, please

- star the repository or,
- contribute with feedback to the issue tracker.

**Quick overview**

1. ***Preamble: Ubuntu and Python***
   A few tips on Ubuntu terminal and checking your Python installation.

2. ***Preamble: Python Best Practices***
   A quick memory refresher for the Python stuff useful in ROS2.
3. ***ROS2 Setup*** **(start here)**
   Installing ROS2 and setting up its environment for use.
4. ***ROS2 Python Package/Build***
   Creating our first ROS2 package with **`ament_python`** and building it with **`colcon`**.
5. ***ROS2 Python Node***
   Creating `rclpy` Nodes and figuring out what all that means.
6. ***ROS2 Python Library***
   Create Python librarys and importing/using it in another **`ament_python`** package.
7. ***ROS2 Interfaces***
   ROS2 interfaces, i.e., messages `.msg`, services `.srv`, and actions `.action`. Creating custom interface packages with **`ament_cmake`**.
8. ***ROS2 Messages in Python***
   Writing and using ROS2 messages, publishers, and subscribers.
9. ***ROS2 Parameter/Launch***
   Making configurable ROS2 Nodes using parameters and launch files.
10. ***ROS2 Services in Python***
    Writing ROS2 services, service servers, and service clients.
11. ***ROS2 Actions in Python***
    Writing ROS2 actions, action servers, and action clients.
12. ***Frame Transformations***
    Frame transformations, the `TransformStamped` message, and `tf2`.
13. ***Gazebo and ROS2***
    **`Gazebo`** installation, basic usage, and integration examples with ROS2.
14. ***nav2***
    `nav2` installation, basic usage, and integration examples with custom ROS2 nodes.
15. ***Cybersecurity***
    Cybersecurity, networking, and public-key cryptography in a context of robotic systems.

CHAPTER

ONE

# UBUNTU TERMINAL BASICS

You already know how to turn on your computer and press some keys to make bits flip and colorful pixels shine on your monitor. Here, we'll go through a few tips on Ubuntu.

**Note**

In this section, we'll go through some basic tools available in Ubuntu's terminal that help with our quest to learn/use ROS2. This is not a complete tutorial on Ubuntu.

## 1.1 Who cares about the `terminal` anyways, are you like 100 years old or something?

Besides the unintended upside that if you're typing into a terminal fast enough with a black hoodie, you're cosplaying Mr. Robot at a very low cost, there wouldn't be another way to make a tutorial like this within the current age of the Universe without relying on Ubuntu's **`terminal`**.

GUIs (Graphical User Interfaces) change faster than long tutorials like this one can keep up with and **`terminal`** is our reliable partner in crime and unlikely to change much in the foreseeable future.

For the whole tutorial, you can copy and paste the commands in **`terminal`**. If it doesn't work, it's either your fault or mine, but surely not the **`terminal`**'s.

## 1.2 The `terminal`

**Note**

Check out Canonical's Tutorial on **`terminal`** for the complete story.

**Hint**

You can open a new terminal window by pressing `CTRL+ALT+T`.

**Warning**

This section is about the default terminal in Ubuntu. If you prefer to use some other terminal instead (there are many), then this might not be useful to you, and you might be happier referring to its documentation instead.

The **terminal** is one of those things with many names. Some call it **shell**, some **console**, some **command line**, some **terminal**. I'm sure there's someone furiously typing right now saying that I'm wrong and describing in detail what those differences might be. The truth is that, in the wild (a.k.a. the Internet), those terms are used pretty much as synonyms.

For all intents and purposes, Tom Hanks is not stuck in this terminal. Instead, we use it to send commands to Ubuntu and make stuff happen.

Table 1: (Murilo's) List of Useful Command Line Programs

| Program | Example usage | What it does |
|---|---|---|
| **pwd** | `pwd` | Outputs the absolute path to the current directory. |
| **mkdir** | `mkdir a_folder` | **Ma**kes a **dir**ectory called `a_folder` in the current directory. |
| **cd** | `cd a_folder` | **C**hanges **d**irectory to a specified target. |
| **touch** | `touch a_file.whatever` | Creates an empty file called `a_file.whatever`. |
| **cat** | `cat a_file.whatever` | Outputs into the console the contents of `a_file.whatever`. |
| **rm** | `rm a_file.whatever` | **Rem**oves a file or directory (with the `-r` option). |
| **ls** | `ls` | **Ls**ts the contents of the current directory. |
| **grep** | `cat a_file.whatever \| grep robocop` | Outputs the lines of `a_file.whatever` that contain the string `robocop`. |
| **nano** | `nano a_file.whatever` | Helps you edit a file using a (relatively?) user-friendly program so that you don't get stuck into vim. |
| **sudo** | `sudo touch a_sudo_made_file.whatever` | With the powers of a **s**uper **u**ser, **do** something. It allows a given user to modify sensitive files in Ubuntu. |
| **apt** | `sudo apt install git` | Installs Ubuntu packages, in this case, **git**. |
| **alias** | `alias say_hello="echo hello"` | Creates an alias for a command, i.e. another way to refer to it. |

## 1.3 Let's use it. (!?)

The thing is, we'll be using the terminal throughout the entire tutorial, so don't worry about going too deep right now.

To warm up, let's start by creating an empty file inside a new directory, as follows

**Hint**

The path ~ stands for the currently logged-in user's home folder. Not every command is able to expand it. In those cases, use `$USER` instead.

**Hint**

You can open a new terminal window by pressing `CTRL+ALT+T`.

> **Warning**
>
> For copying from the terminal use `CTRL+SHIFT+C`. For pasting to the terminal, use `CTRL+SHIFT+V`. Be careful with `CTRL+C`, in particular. It is used to, in simple terms, close running programs on the terminal.

```
cd ~
mkdir a_folder
cd a_folder
touch an_empty_file.txt
```

Then, we can use **nano** to create another file with some contents

```
nano file_with_stuff.txt
```

Then, **nano** will run. At this point we can start typing, so let's just type

```
stuff
```

then you can exit with the following keys

1. `CTRL+X`
2. `Y`
3. `ENTER`

you can also look at the bottom side of the window to know what keys to press. As an example, in **nano**, `^X` stands for `CTRL+X`.

Then, if you run

```
ls
```

the output will be

```
an_empty_file.txt  file_with_stuff.txt
```

we can, for example, get the contents of `file_with_stuff.txt` with

```
cat file_with_stuff.txt
```

whose output will be

```
stuff
```

So, enough of this example, let's get rid of everything with

> **Warning**
>
> **ALWAYS** be careful when using **rm**. The files removed this way do NOT go to the trash can, if you use it you pretty much said bye bye bye to those files/directories.

```
cd ~
rm -r a_folder
```

## 1.4 bash redirections

**Important**

More info is available at the Bash Reference Manual.

**Hint**

Before defaulting to writing a 300-lines-long Python script for the simplest and most common of tasks, it is always good to check if there is something already available in **bash** that can do the same thing in an easier and more stable way.

In a time long long ago, before ChatGPT became the new Deep Magic, **bash** was already tilting heads and leaving Ubuntu users in awe.

Among many powerful features, the *redirection operator*, >, stands out. It can be used to, unsurprisingly, *redirect* the output of a command to a file.

**Warning**

The operator > overwrites the target file with the output of the preceding command, it does not ask for permission, it just goes and does it.

The operator >> appends to the target file with the output of the preceding command.

Don't mix these up, there is no way to undo.

For example, if we want to store the result of the command `ls` to a file called `result_of_ls.txt`, the following will do

```
cd ~
ls > result_of_ls.txt
```

As a default in this version of Ubuntu, if the file does not exist it is created.

## 1.5 Tab completion

**Hint**

Use `TAB` completion extensively.

Whenever I have to look at a novice's shoulders while they interact with the terminal it gives me a certain level of anxiety. That is because they are trying to perfectly type even the longest and meanest paths for files, directories, and programs.

The terminal has `TAB` completion, so use it extensively. You can press `TAB` at any time to complete the name of a program, folder, file, or pretty much anything.

For example, we can move to a folder

```
cd ~
```

Then type a partial command or a part of its arguments. For example,

```
rm result_o
```

then, by pressing `TAB`, it should autocomplete to

```
rm result_of_ls.txt
```

## 1.6 Be careful with sudo

> **Warning**
>
> **DO NOT**, I repeat, **DO NOT** play around with **`sudo`**.

With great power, comes great opportunity to destroy your Ubuntu. It turns out that **`sudo`** is the master key of destruction, it will allow you to do basically anything in the system as far as the software is concerned.

So, don't.

For these tutorials, only use **`sudo`** when installing system-wide packages. Otherwise, do not use it.

## 1.7 Be careful even when not using sudo

With regular user privileges, the major system folders will be protected from tampering. However, our home folder, e.g. `/home/<YOU>` will not. In our home folder, we are the lords, so a mistake can be fatal for your files/directories.

## 1.8 File permissions

> **Warning**
>
> **DO NOT**, I repeat, **DO NOT** play around with **`sudo`**, **`chmod`**, or **`chown`**.

One of the reasons that using **`sudo`** indiscriminately will destroy your Ubuntu is file permissions. For example, if you *simply* open a file and save it as **`sudo`**, you'll change its permissions, and that might be enough to even block you from logging into Ubuntu via the GUI (Graphics User Interface).

I will not get into detail here about programs to change permissions because we won't need them extensively in these tutorials. However, it is important to be aware that this exists and might cause problems.

## 1.9 `nautilus`: browsing files with a GUI

To some extent similar to **`explorer`** in Windows and **`finder`** in macOS, **`nautilus`** is the default file manager in Ubuntu.

One tip is that it can be opened from the **`terminal`** as well, so that you don't have to find whatever folder you are again. For example,

**Hint**

The path `.` means the current folder.

```
cd ~
nautilus .
```

will open the currently logged-in user's home folder in **`nautilus`**.

CHAPTER

TWO

# INSTALLING PYTHON ON UBUNTU

**Warning**

If you change or try to tinker with the default Python version of Ubuntu, your system will most likely **BREAK COMPLETELY**. Do not play around with the default Python installation, because Ubuntu depends on it to work properly (or work at all).

In Ubuntu, Python is already installed! In fact, Ubuntu would not work without it. Let's check its version by running

```
python3 --version
```

which should output

```
Python 3.12.3
```

If the `3.12` part of your version is different, this tutorial might not work for you. Please make sure to use the default Python in your Ubuntu.

**Warning**

Note that the command is **`python3`** and not **`python`**. In fact, the result of

```
python
```

is

```
 Command 'python' not found, did you mean:
command 'python3' from deb python3
command 'python' from deb python-is-python3
```

## 2.1 A quick Python check

Run

```
python3
```

which should output something similar to

```
Python 3.12.3 (main, Jun 18 2025, 17:59:45) [GCC 13.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

in particular, if the `GCC 13` is different, then this tutorial might not work for you.

As you already know, to exit the interactive shell you can use `CTRL+D` or type `quit()` and press `ENTER`.

## 2.2 Some Python packages must be installed through `apt`

> **Warning**
>
> Aside from these packages that you **MUST** install from **`apt`**, it is best to use **`venv`** and **`pip`** to install packages only for your user without using `sudo`.

For some Python packages to work well with the default Python in Ubuntu, they must be installed through **`apt`**. If you deviate from this, you can cause issues that might not be easy to recover from.

For the purposes of this tutorial, let us install `pip` and `venv`

```
sudo apt update
sudo apt install -y python3-pip python3-venv
```

## 2.3 When you want to isolate your environment, use `venv`

> **Warning**
>
> At the time of this writing, there was no support for **`venv`** on ROS2 (More info). Until that is handled, we are not going to use **`venv`** for the ROS2 tutorials. However, we will use **`venv`** to protect our ROS2 environment from these Python preamble tutorials.

Using **`venv`** (More info) is quite straightforward.

### 2.3.1 Create a `venv`

```
cd ~
python3 -m venv ros2tutorial_venv
```

where the only argument, `ros2tutorial_venv`, is the name of the folder in which the `venv` will be created.

### 2.3.2 Activate a `venv`

Whenever we want to use a `venv`, it must be explicitly activated.

```
cd ~
source ros2tutorial_venv/bin/activate
```

The terminal will change to have the prefix `(ros2tutorial_venv)` to let us know that we are using a `venv`, as follows

```
(ros2tutorial_venv) murilo@murilos-toaster:~$
```

### 2.3.3 Deactivate a venv

To deactivate, run

```
deactivate
```

We'll know that we're no longer using the `ros2tutorial_venv` because the prefix will disappear back to

```
murilo@murilos-toaster:~$
```

## 2.4 Installing libraries

**Warning**

In these tutorials, we rely either on **`apt`** or **`pip`** to install packages. There are other package managers for Python and plenty of other ways to install and manage packages. They are, in general, not compatible with each other so, like cleaning products, **DO NOT** mix them.

**Hint**

Using `python3 -m pip` instead of calling just `pip` allows more control over which version of **`pip`** is being called. The need for this becomes more evident when several Python versions have to coexist in a system.

As an example, let us install the best robot modeling and control library ever conceived, DQ Robotics.

First, we activate the virtual environment

```
cd ~
source ros2tutorial_venv/bin/activate
```

then, we install

```
python3 -m pip install dqrobotics --break-system-packages
```

which will result in something similar to (might change depending on future versions)

```
Collecting dqrobotics
  Downloading dqrobotics-25.4.0a17-cp312-cp312-manylinux2014_aarch64.whl.metadata (2.9␣
↪kB)
Requirement already satisfied: numpy in /usr/lib/python3/dist-packages (from dqrobotics)␣
↪(1.26.4)
Downloading dqrobotics-25.4.0a17-cp312-cp312-manylinux2014_aarch64.whl (512 kB)
    512.5/512.5 kB 14.0 MB/s eta 0:00:00
[...]
Installing collected packages: dqrobotics
Successfully installed dqrobotics-25.4.0a17
```

## 2.5 Removing libraries (installed with `pip`)

We can remove the library we just installed with

```
python3 -m pip uninstall dqrobotics --break-system-packages
```

resulting in

```
Found existing installation: dqrobotics 25.4.0a7
Uninstalling dqrobotics-25.4.0a7:
  Would remove:
    /usr/local/lib/python3.12/dist-packages/dqrobotics-25.4.0a7.dist-info/*
    /usr/local/lib/python3.12/dist-packages/dqrobotics/*
Proceed (Y/n)?
```

**Hint**

If in the terminal a question is made, the option with an uppercase letter, in this case `Y`, will be the default. If you want the default, just press `ENTER`.

Then, press `ENTER`, which results in

```
Successfully uninstalled dqrobotics-25.4.0a7
```

## 2.6 When using `pip`, do NOT use `sudo`

Using `sudo` without knowing what one is doing is *the* easiest way to wreak havoc in a Ubuntu installation. Even seemingly innocuous operations such as copying files with `sudo` can cause irreparable damage to your Ubuntu environment.

When installing Python packages that are not available on **`apt`**, use **`pip`**.

CHAPTER

THREE

# (MURILO'S) PYTHON BEST PRACTICES

**Warning**

This tutorial expects prior knowledge in Python and object-oriented programming. As such, this section is not meant to be a comprehensive Python tutorial. You have better resources made by smarter people available online, e.g. The Python Tutorial.

## 3.1 Terminology

Let's go through the Python terminology used in this tutorial. This terminology is not necessarily uniform with other sources/tutorials you might find elsewhere. It is based on my interpretation of The Python Tutorial on Modules, the Python Glossary, and my own experience.

Table 1: (Murilo's) Python Glossary

| Term | Book Definition | Use in the wild |
|---|---|---|
| script | A Python file that can be executed. | Any Python file *meant to be* executed. |
| module | A file with content that is meant to be imported by other modules and scripts. | This term is used very loosely and can basically mean any Python file, but usually a Python file *meant to be* imported from. |
| package | A collection of modules. | A folder with an `__init__.py`, even if it doesn't have more than one module. When people say Python Packaging it refers instead to making your package installable (e.g. with a `setup.py` or `pyproject.toml`), so be ready for that ambiguity. |

## 3.2 Use a venv

We already know that it is a good practice to *When you want to isolate your environment, use venv*. So, let's turn that into a reflex and do so for this whole section.

```
cd ~
source ros2tutorial_venv/bin/activate
```

## 3.3 Minimalist package: something to start with

**In this step, we'll work on these.**

```
python/
`-- minimalist_package
    |-- minimalist_package
    |   |-- __init__.py
    |   |-- _minimalist_class.py
    |   |-- minimalist_async
    |   |   |-- __init__.py
    |   |   |-- _unlikely_to_return.py
    |   |   |-- async_await_example.py
    |   |   `-- async_callback_example.py
    |   `-- minimalist_script.py
    |-- setup.py
    `-- test
        `-- test_minimalist_class.py
```

First, let's make a folder for our project

**Hint**

The `-p` option for **`mkdir`** creates all parent folders as well, when they do not exist.

```
mkdir -p ~/ros2_tutorials_preamble/python/minimalist_package
```

Then, let's create a folder with the same name within it for our package. A Python package is a folder that has an `__init__.py`, so for now we add an empty `__init__.py` by doing so

```
cd ~/ros2_tutorials_preamble/python/minimalist_package
mkdir minimalist_package
cd minimalist_package
touch __init__.py
```

The (empty) package is done!

**Hint**

In `PyCharm`, open the `~/ros2_tutorials_preamble/python/minimalist_package` folder to correctly interact with this project.

**Warning**

It is confusing to have two nested folders with the same name. However, this is quite common and starts to make sense after getting used to it (it is also the norm in ROS2). The first folder is supposed to be how your file system sees your package, i.e. the *project* folder, and the other contains the actual Python package, with the `__init__.py` and other source code.

## 3.4 Minimalist script

**In this step, we'll work on these.**

```
python/
`-- minimalist_package
    |-- minimalist_package
    |   |-- __init__.py
    |   |-- _minimalist_class.py
    |   |-- minimalist_async
    |   |   |-- __init__.py
    |   |   |-- _unlikely_to_return.py
    |   |   |-- async_await_example.py
    |   |   `-- async_callback_example.py
    |   `-- minimalist_script.py
    |-- setup.py
    `-- test
        `-- test_minimalist_class.py
```

Let's start with a minimalist script that prints a string periodically, as follows.

In the directory `~/ros2_tutorials_preamble/python/minimalist_package/minimalist_package`, create the following file.

`minimalist_script.py`

```python
#!/bin/python3
import time


def main() -> None:
    """An example main() function that prints 'Howdy!' twice per second."""
    try:
        while True:
            print("Howdy!")
            time.sleep(0.5)
    except KeyboardInterrupt:
        pass
    except Exception as e:
        print(e)


if __name__ == "__main__":
    """When this module is run directly, it's __name__ property will be '__main__'."""
    main()
```

## 3.5 Running a Python script on the terminal

There are a few ways to run a script/module in the command line. Without worrying about file permissions, specifying that the file must be interpreted by Python (and which version of Python) is the most general way to run a script

```
cd ~/ros2_tutorials_preamble/python/minimalist_package/minimalist_package
python3 minimalist_script.py
```

which will output

> **Hint**
>
> You can end the **minimalist_script.py** by pressing CTRL+C in the terminal in which it is running.

```
Howdy!
Howdy!
Howdy!
```

Another way to run a Python script is to execute it directly in the terminal. This can be done with

```
cd ~/ros2_tutorials_preamble/python/minimalist_package/minimalist_package
./minimalist_script.py
```

which will result in

```
bash: ./minimalist_script.py: Permission denied
```

because our file does not have the permission to run as an executable. To give it that permission, we must run **ONCE**

```
cd ~/ros2_tutorials_preamble/python/minimalist_package/minimalist_package
chmod +x minimalist_script.py
```

and now we can run it properly with

```
cd ~/ros2_tutorials_preamble/python/minimalist_package/minimalist_package
./minimalist_script.py
```

resulting in

```
Howdy!
Howdy!
Howdy!
```

Note that for this second execution strategy to work, we **MUST** have the *#!*, called shebang, at the beginning of the first line. The path after the shebang specifies what program will be used to interpret that file. In general, differently from Windows, Ubuntu does not guess the file type by the extension when running it.

```
#!/bin/python3
```

If we remove the shebang line and try to execute the script, it will return the following errors, because Ubuntu doesn't know what to do with that file.

```
./minimalist_script.py: line 2: import: command not found
./minimalist_script.py: line 5: syntax error near unexpected token `('
./minimalist_script.py: line 5: `def main() -> None:'
```

## 3.6 When using `if __name__=="__main__":`, just call the real `main()`

There are multiple ways of running a Python script. In the one we just saw, the name of the module becomes `__main__`, but in others that does not happen, meaning that the `if` can be completely skipped. So, write the `main()` function of a script as something standalone and, in the condition, just call it and do nothing else, as shown below

```
if __name__ == "__main__":
    """When this module is run directly, it's __name__ property will be '__main__'."""
    main()
```

## 3.7 It's dangerous to go alone: Always wrap the contents of `main` function on a *try–except* block

It is good practice to wrap the contents of `main()` call in a `try--except` block with at least the `KeyboardInterrupt` clause. This allows the user to shutdown the module cleanly. We have done so in the example as follows

```
def main() -> None:
    """An example main() function that prints 'Howdy!' twice per second."""
    try:
        while True:
            print("Howdy!")
            time.sleep(0.5)
    except KeyboardInterrupt:
        pass
    except Exception as e:
        print(e)
```

This is of particular importance when hardware is used, otherwise, the connection with it might be left in an undefined state causing difficult-to-understand problems at best and physical harm at worst.

The `Exception` clause in our example is very broad, but a **MUST** in code that is still under development. Exceptions of all sorts can be generated when there is a communication error with the hardware, software (internet, etc), or other issues.

This broad `Exception` clause could be replaced for a less broad exception handling if that makes sense in a given application, but that is usually not necessary nor safe. When handling hardware, it is, in general, **IMPOSSIBLE** to test the code of all combinations of inputs and states. As they say,

> *Be wary, for overconfidence is a slow and insidious [source for terrible bugs and failed demos]*

**Hint**

Catching all `Exceptions` might make debugging more difficult in some cases. At your own risk, you can remove this clause temporarily when trying to debug a stubborn bug, at the risk of forgetting to put it back and ruining your hardware.

## 3.8 Minimalist class: Use classes profusely

**In this step, we'll work on these.**

```
python/
`-- minimalist_package
```

```
      |-- minimalist_package
      |   |-- __init__.py
      |   |-- _minimalist_class.py
      |   |-- minimalist_async
      |   |   |-- __init__.py
      |   |   |-- _unlikely_to_return.py
      |   |   |-- async_await_example.py
      |   |   `-- async_callback_example.py
      |   `-- minimalist_script.py
      |-- setup.py
      `-- test
          `-- test_minimalist_class.py
```

As you are familiar with object-oriented programming, you know that classes are central to this paradigm. As a memory refresher, let's make a class that honestly does nothing really useful but illustrates all the basics of a Python class.

Create the file below in the directory `~/ros2_tutorials_preamble/python/minimalist_package/minimalist_package`.

`_minimalist_class.py`

```python
class MinimalistClass:
    """
    A minimalist class example with the most used elements.
    https://docs.python.org/3/tutorial/classes.html
    """
    # Attribute reference, accessed with MinimalistClass.attribute_reference
    attribute_reference: str = "Hello "

    def __init__(self,
                 attribute_arg: float = 10.0,
                 private_attribute_arg: float = 20.0):  # With a default value of 20.0
        """The __init__ works together with __new__ (not shown here) to
        construct a class. Loosely it is called the Python 'constructor' in
        some references, although it is officially an 'initializer' hence
        the name.
        https://docs.python.org/3/reference/datamodel.html#object.__init__
        It customizes an instance with input arguments.
        """
        # Attribute that can be accessed externally
        self.attribute: float = attribute_arg

        # Attribute that should not be accessed externally
        # a name prefixed with an underscore (e.g. _spam) should be treated
        # as a non-public part of the API (whether it is a function, a method or a data member).
        # It should be considered an implementation detail and subject to change without notice.
        self._private_attribute: float = private_attribute_arg

    def method(self) -> float:
        """Methods with 'self' should use at least one statement in which 'self' is required."""
```

(continues on next page)

(continued from previous page)

```
        return self.attribute + self._private_attribute

    def set_private_attribute(self, private_attribute_arg: float) -> None:
        """If a private attribute should be writeable, define a setter."""
        self._private_attribute = private_attribute_arg

    def get_private_attribute(self) -> float:
        """If a private attribute should be readable, define a getter."""
        return self._private_attribute

    @staticmethod
    def static_method():
        """
        Methods that do not use the 'self' should be decorated with the @staticmethod.
        It will only have access to attribute references.
        https://docs.python.org/3.10/library/functions.html#staticmethod
        """
        return MinimalistClass.attribute_reference + "World!"
```

then, let's modify the `__init__.py` with the following contents

`__init__.py`

```
"""
Having an __init__.py file within a directory turns it into a Python Package.
A package within a package is called a subpackage.
https://docs.python.org/3/tutorial/modules.html#packages
"""
from minimalist_package._minimalist_class import MinimalistClass
```

**Note**

When adding imports to the `__init__.py`, the folder that we use to open in Pycharm and that we call to execute the scripts is *extremely* relevant. When packages are deployed (e.g. in PyPI or ROS2), the "correct" way to import in `__init__.py` is to use `from <PACKAGE_NAME>.<MODULE> import <THING_TO_IMPORT>`, which is why we're doing it this way.

**Note**

Relative imports such as `from . import <THING_TO_IMPORT>` might work in some cases, and that is fine. It is a supported and valid way to import. However, don't be surprised when it doesn't work in ROS2, PyPI packages, etc., and generates a lot of frustration.

## 3.9 Not a matter of taste: Code style

It might be parsing through jibber-jabber code in l__tcode lessons with weird C-pointer logic and nested dereference operators that gets you through the door into one of those fancy companies with no dress code and free snacks, perks that I'm totally not envious of one bit. In the ideal world, at least, writing easy-to-understand code with the proper style is what should keep you in that job.

So, always pay attention to the naming of classes (PascalCase), files and functions (snake_case), etc.

Thankfully, Python has a bunch of style rules builtin the language and PEP (Python Enhancement Proposal), such as PEP8. Take this time to read it and get inspired by The Zen of Python.

## 3.10 Take the (type) hint: Always use type hints

> **Note**
>
> For more info, check out the documentation on Python typing and the type hints cheat sheet

Before you flood my inbox with complaints, let me vent for you. A *preemptive* vent.

> *But, you know, one of the cool things in Python is that we don't have to explicitly type variables. Do you want to turn Python into C?? Why do you love C++ so much you unpythonic Python hater????*

The dynamic typing nature of Python is, no doubt, a strong point of the language. Note that adding type hints does not impede your code to be used with other types as arguments. Type hints are, to no one's surprise, hints to let users (and some automated tools) know what types your functions were made for, e.g. to allow your favorite IDE (Integrated Development Environment) to help you with code suggestions.

In these tutorials, we are not going to use any complex form of type hints. We're basically going to attain ourselves to the simplest two forms, the (attribute, argument, etc) type, and the return types.

For attributes we use `<attribute>: type`, as shown below

```
        self.attribute: float = attribute_arg
```

For method arguments we use `<argument>: <type>` and for return types we use `def <method>(<params>) -> <type>`, as shown below in our example

```
    def set_private_attribute(self, private_attribute_arg: float) -> None:
        """If a private attribute should be writeable, define a setter."""
        self._private_attribute = private_attribute_arg
```

## 3.11 Document your code with Docstrings

You do not need to document every single line you code, that would in fact be quite obnoxious

```
# c stores the sum of a and b
c = a + b

# d stores the square of c
d = c**2

# check if d is zero
if d == 0:
   # Print warning
   print("Warning")
```

But, on the other side of the coin, it doesn't take too long for us to forget what the parameters of a function mean. *Take the (type) hint: Always use type hints* helps a lot, but additional information is always welcome. If you get used to using docstrings for every new method, your programming will be better in general because documenting your code makes you think about it.

The example below shows a quick explanation of what the class does using a docstring

```
class MinimalistClass:
    """
    A minimalist class example with the most used elements.
    https://docs.python.org/3/tutorial/classes.html
    """
```

The PEP 257 talks about docstrings but does not define too much beyond saying that we should use it. My recommendation as of now would be the Sphinx markup, because of the many Python libraries using it for Sphinx documentation/tutorials like this one.

The sample code shown in this section has docstrings everywhere, but they are being used to explain the general usage of some Python syntax. When documenting your code, obviously, the documentation should be about what the method/class/attribute does.

**Hint**

Ideally, all documentation is perfect from the start. In reality, however, that rarely ever happens so some documentation is always better than none. My advice would be to write something as it goes and possibly adjust it to more stable or cleaner documentation when the need arises.

## 3.12 Unit tests: always test your code

**Note**

For a comprehensive tutorial on unit testing go through the unittest docs.

**In this step, we'll work on these.**

```
python/
`-- minimalist_package
    |-- minimalist_package
    |   |-- __init__.py
    |   |-- _minimalist_class.py
    |   |-- minimalist_async
    |   |   |-- __init__.py
    |   |   |-- _unlikely_to_return.py
    |   |   |-- async_await_example.py
    |   |   `-- async_callback_example.py
    |   `-- minimalist_script.py
    |-- setup.py
    `-- test
        `-- test_minimalist_class.py
```

Unit testing is a flag that has been waved by programming enthusiasts and is often a good measurement of code maturity.

The elephant in the room is that writing unit tests is **boring**. Yes, we know, *very* boring.

Unit tests are boring because they are an *investment*. Unit testing won't necessarily make your code [...] better, faster, [...] *right now*. However, without tests, don't be surprised after some point if your implementations make you drown

in tech debt. Dedicating a couple of minutes now to make a couple of tests when your codebase is still in its infancy makes it more manageable and less boresome.

Back to the example, a good practice is to create a folder name `test` at the same level as the packages to be tested, like so

```
cd ~/ros2_tutorials_preamble/python/minimalist_package
mkdir test
```

Then, we create a file named `test_minimalist_class.py` with the contents below in the `test` folder.

> **Note**
>
> The prefix `test_` is important as it is used by some frameworks to automatically discover tests. So it is better not to use that prefix if that file does not contain a unit test.

`test_minimalist_class.py`

```
import unittest
from minimalist_package import MinimalistClass


class TestMinimalistClass(unittest.TestCase):
    """For each `TestCase`, we create a subclass of `unittest.TestCase`."""

    def setUp(self):
        self.minimalist_instance = MinimalistClass(attribute_arg=15.0,
                                                   private_attribute_arg=35.0)

    def test_attribute(self):
        self.assertEqual(self.minimalist_instance.attribute, 15.0)

    def test_private_attribute(self):
        self.assertEqual(self.minimalist_instance._private_attribute, 35.0)

    def test_method(self):
        self.assertEqual(self.minimalist_instance.method(), 15.0 + 35.0)

    def test_get_set_private_attribute(self):
        self.minimalist_instance.set_private_attribute(20.0)
        self.assertEqual(self.minimalist_instance.get_private_attribute(), 20.0)

    def test_static_method(self):
        self.assertEqual(MinimalistClass.static_method(), "Hello World!")


def main():
    unittest.main()
```

### 3.12.1 Running the tests

For a quick jolt of instant gratification, let's run the tests before we proceed with the explanation.

There are many ways to run tests written with `unittest`. The following will run all tests found in the folder `test`

```
cd ~/ros2_tutorials_preamble/python/minimalist_package
python -m unittest discover -v test
```

which will output

```
test_attribute (test_minimalist_class.TestMinimalistClass) ... ok
test_get_set_private_attribute (test_minimalist_class.TestMinimalistClass) ... ok
test_method (test_minimalist_class.TestMinimalistClass) ... ok
test_private_attribute (test_minimalist_class.TestMinimalistClass) ... ok
test_static_method (test_minimalist_class.TestMinimalistClass) ... ok

----------------------------------------------------------------------
Ran 5 tests in 0.000s

OK
```

Yay! We've done it!

### 3.12.2 Start with use `unittest`

> **Note**
>
> ROS2 uses `pytest` as default, but that doesn't mean you also have to use it in every Python code you ever write.

There are many test frameworks for Python. Nonetheless, the unittest module is built into Python so, unless you have a very good reason not to use it, just [use] it.

We import the `unittest` module along with the class that we want to test, namely `MinimalistClass`.

```
import unittest
from minimalist_package import MinimalistClass
```

### 3.12.3 Test them all

> **Note**
>
> Good unit tests will not only let you know when something broke but also *where* it broke. A failed test of a high-level function might not give you too much information, whereas a failed test of a lower-level (more fundamental) function will allow you to pinpoint the issue.

Unit tests are somewhat like insurance. The more coverage you have, the better. In this example, we test all the elements in the class. Each test will be based on one or more asserts. For more info check the unittest docs.

In a few words, we make a subclass of `unittest.TestCase` and create methods within it that test one part of the code, hence the name unit tests.

```
    def test_attribute(self):
        self.assertEqual(self.minimalist_instance.attribute, 15.0)

    def test_private_attribute(self):
        self.assertEqual(self.minimalist_instance._private_attribute, 35.0)

    def test_method(self):
        self.assertEqual(self.minimalist_instance.method(), 15.0 + 35.0)

    def test_get_set_private_attribute(self):
        self.minimalist_instance.set_private_attribute(20.0)
        self.assertEqual(self.minimalist_instance.get_private_attribute(), 20.0)

    def test_static_method(self):
        self.assertEqual(MinimalistClass.static_method(), "Hello World!")
```

If one of the `asserts` fails, then the related test will fail, and the test framework will let us know which one.

### 3.12.4 The test's main function

Generally, a test script based on *unittest* will have the following main function. It will run all available tests in our test class. For more info and alternatives check the unittest docs.

```
def main():
    unittest.main()
```

CHAPTER

# FOUR

# PYTHON'S `asyncio`

**Note**

Asynchronous code is not the same as code that runs in parallel, even more so in Python because of the GIL (Global Interpreter Lock) (More info). Basically, the `async` framework allows us to not waste time waiting for results that we don't know when will arrive. It either allows us to attach a `callback` for when the result is ready, or to run many service calls and `await` for them all, instead of running one at a time.

There are two main ways to interact with `async` code, the first being by `await`ing the results or by handling those results through `callbacks`. Let's go through both of them with examples.

## 4.1 Use a venv

We already know that it is a good practice to *When you want to isolate your environment, use venv*. So, let's turn that into a reflex and do so for this whole section.

```
cd ~
source ros2tutorial_venv/bin/activate
```

## 4.2 Create the `minimalist_async` package

**In this step, we'll work on these.**

```
python/
`-- minimalist_package
    |-- minimalist_package
    |   |-- __init__.py
    |   |-- _minimalist_class.py
    |   |-- minimalist_async
    |   |   |-- __init__.py
    |   |   |-- _unlikely_to_return.py
    |   |   |-- async_await_example.py
    |   |   `-- async_callback_example.py
    |   `-- minimalist_script.py
    |-- setup.py
    `-- test
        `-- test_minimalist_class.py
```

As we learned in *Minimalist package: something to start with*, let's make a package called `minimalist_async`.

```
cd ~/ros2_tutorials_preamble/python/minimalist_package/minimalist_package
mkdir minimalist_async
cd minimalist_async
```

we then create an `__init__.py` file with the following contents

`__init__.py`

```
from minimalist_package.minimalist_async._unlikely_to_return import unlikely_to_return
```

## 4.3 Create the async function

**In this step, we'll work on these.**

```
python/
`-- minimalist_package
    |-- minimalist_package
    |   |-- __init__.py
    |   |-- _minimalist_class.py
    |   |-- minimalist_async
    |   |   |-- __init__.py
    |   |   |-- _unlikely_to_return.py
    |   |   |-- async_await_example.py
    |   |   `-- async_callback_example.py
    |   `-- minimalist_script.py
    |-- setup.py
    `-- test
        `-- test_minimalist_class.py
```

Let's create a module called `_unlikely_to_return.py` to hold a function used for this example at the `~/ros2_tutorials_preamble/python/minimalist_package/minimalist_package/minimalist_async` folder with the following contents

`_unlikely_to_return.py`

```
import asyncio
import random
from textwrap import dedent


async def unlikely_to_return(tag: str, likelihood: float = 0.1) -> float:
    """
    A function that is unlikely to return.
    :return: When it returns, the successful random roll as a float.
    """
    while True:
        a = random.uniform(0.0, 1.0)
        if a < likelihood:
            print(f"{tag} Done.")
            return a
```

(continues on next page)

(continued from previous page)

```
        else:
            print(f"{tag} retry needed (roll = {a} > {likelihood})")
            await asyncio.sleep(0.1)
```

Because we're using `await` in the function, we start by defining an `async` function.

**Hint**

If the function/method uses `await` anywhere, it should be `async` (More info).

This function was thought this way to emulate, for example, us waiting for something external without actually having to. To do so, we add a `while True:` and return only with 10% chance. Instead of using a `time.sleep()` we use `await asyncio.sleep(0.1)` to unleash the power of `async`. The main difference is that `time.sleep()` is synchronous (blocking), meaning that the interpreter will be locked here until it finishes. With `await`, the interpreter is free to do other things and come back to this one later after the desired amount of time has elapsed.

The function by itself doesn't do much, so let's use it in another module.

## 4.4 Using `await`

**TL;DR Using `await`**

1. Run multiple `Tasks`.
2. Use `await` for them, **after they were executed**.

**In this step, we'll work on these.**

```
python/
`-- minimalist_package
    |-- minimalist_package
    |   |-- __init__.py
    |   |-- _minimalist_class.py
    |   |-- minimalist_async
    |   |   |-- __init__.py
    |   |   |-- _unlikely_to_return.py
    |   |   |-- async_await_example.py
    |   |   `-- async_callback_example.py
    |   `-- minimalist_script.py
    |-- setup.py
    `-- test
        `-- test_minimalist_class.py
```

Differently from synchronous programming, using `async` needs us to reflect on several tasks being executed at the same time(-ish). The main use case is for programs with multiple tasks that can run concurrently and, at some point, we need the result of those tasks to either end the program or further continue with other tasks.

The `await` strategy we're seeing now is suitable when either we need the results from all tasks before proceeding or when the order of results matters.

To illustrate this, let's make a file called `async_await_example.py` in `minimalist_async` with the following contents.

`async_await_example.py`

```python
import asyncio
from minimalist_package.minimalist_async import unlikely_to_return


async def async_main() -> None:
    tags: list[str] = ["task1", "task2"]
    tasks: list[asyncio.Task] = []

    # Start all tasks before awaiting them, otherwise the code
    # will not be concurrent.
    for task_tag in tags:
        task = asyncio.create_task(
            unlikely_to_return(tag=task_tag)
        )
        tasks.append(task)

    # Alternatively, use asyncio.gather()
    # At this point, the functions are already running concurrently. We are now␣
↪(a)waiting for the
    # results, IN THE ORDER OF THE AWAIT, even if the other task ends first.
    print("Awaiting results...")
    for (tag, task) in zip(tags, tasks):
        result = await task
        print(f"The result of task={tag} was {result}.")


def main() -> None:
    try:
        asyncio.run(async_main())
    except KeyboardInterrupt:
        pass
    except Exception as e:
        print(e)


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

We start by importing the `async` method we defined in the other module

```python
from minimalist_package.minimalist_async import unlikely_to_return
```

The function will be run by an instance of `asyncio.Task`. When the task is created, it is equivalent to calling the function and it starts running concurrently to the script that created the task. The example is a bit on the fancy side to make it easier to read and maintain, but the concept is simple. When using the `await` paradigm, focus on the following

1. Make the function it should run, like our `unlikely_to_return()`.
2. Run all concurrent tasks and keep a reference to them as `asyncio.Task`.
3. `await` on each `asyncio.Task`, in the order in which you want those results.

```python
async def async_main() -> None:
    tags: list[str] = ["task1", "task2"]
    tasks: list[asyncio.Task] = []

    # Start all tasks before awaiting them, otherwise the code
    # will not be concurrent.
    for task_tag in tags:
        task = asyncio.create_task(
            unlikely_to_return(tag=task_tag)
        )
        tasks.append(task)

    # Alternatively, use asyncio.gather()
    # At this point, the functions are already running concurrently. We are now␣
↪(a)waiting for the
    # results, IN THE ORDER OF THE AWAIT, even if the other task ends first.
    print("Awaiting results...")
    for (tag, task) in zip(tags, tasks):
        result = await task
        print(f"The result of task={tag} was {result}.")
```

Ok, enough with the explanation, let's go to the endorphin rush of actually running the program with

```
cd ~/ros2_tutorials_preamble/python/minimalist_package/
python3 -m minimalist_package.minimalist_async.async_await_example
```

Which will result in something like shown below. The function is stochastic, so it might take more or less time to return and the order of the tasks ending might also be different.

However, in the `await` framework, the results will **ALWAYS** be processed in the order that was specified by the `await`, **EVEN WHEN THE OTHER TASK ENDS FIRST**, as in the example below. This is neither good nor bad, it will be proper for some cases and not proper for others.

We can also see that both tasks are running concurrently until `task2` finishes, then only `task1` is executed.

```
Awaiting results...
task1 retry needed (roll = 0.36896762068176037 > 0.1).
task2 retry needed (roll = 0.8429002838770375 > 0.1).
task1 retry needed (roll = 0.841018521652675 > 0.1).
task2 retry needed (roll = 0.1351152094825686 > 0.1).
task1 retry needed (roll = 0.9484654265361889 > 0.1).
task2 retry needed (roll = 0.3167046796566366 > 0.1).
task1 retry needed (roll = 0.7519672365071198 > 0.1).
task2 retry needed (roll = 0.38440407016827005 > 0.1).
task1 retry needed (roll = 0.23155484384953284 > 0.1).
task2 retry needed (roll = 0.6418306170261009 > 0.1).
task1 retry needed (roll = 0.532161975008607 > 0.1).
task2 Done.
task1 retry needed (roll = 0.448132225703992 > 0.1).
task1 retry needed (roll = 0.13504700640433664 > 0.1).
task1 retry needed (roll = 0.7404815278498079 > 0.1).
task1 retry needed (roll = 0.9830081693068259 > 0.1).
task1 retry needed (roll = 0.4070546146764875 > 0.1).
```

(continues on next page)

(continued from previous page)

```
task1 retry needed (roll = 0.7474267487174882 > 0.1).
task1 Done.
The result of task=task1 was 0.038934769861482144.
The result of task=task2 was 0.06380247590535493.

Process finished with exit code 0
```

Hooray! May there be concurrency!

## 4.5 Using `callback`

**TL;DR Using `callbacks`**

1. Run multiple `Tasks`.
2. Add a `callback` to handle the result **as soon as it is ready**.
3. Use `await` for each `Task` just so that the main loop does not return prematurely.

**In this step, we'll work on these.**

```
python/
`-- minimalist_package
    |-- minimalist_package
    |   |-- __init__.py
    |   |-- _minimalist_class.py
    |   |-- minimalist_async
    |   |   |-- __init__.py
    |   |   |-- _unlikely_to_return.py
    |   |   |-- async_await_example.py
    |   |   `-- async_callback_example.py
    |   `-- minimalist_script.py
    |-- setup.py
    `-- test
        `-- test_minimalist_class.py
```

Differently from `awaiting` for each task and then processing their result, we can define `callbacks` in such a way that each result will be processed as they come. In that way, the results can be processed in an arbitrary order. Once again, this is inherently neither a good strategy nor a bad one. Some frameworks will work with callbacks, for example ROS1, ROS2, and Qt, but some others will prefer to use `await`.

Enough diplomacy, let's make a file called `async_callback_example.py` in `minimalist_async` with the following contents.

`async_callback_example.py`

```
1 from functools import partial
2 import asyncio
3 from minimalist_package.minimalist_async import unlikely_to_return
4
```

(continues on next page)

(continued from previous page)

```python

def handle_return_callback(tag: str, future: asyncio.Future) -> None:
    """
    Callback example for asyncio.Future
    :param tag: An example parameter, in this case, a tag
    :param future: A asyncio.Future is expected to be the last parameter
    of the callback.
    :return: Nothing.
    """
    if future is not None and future.done():
        print(f"The result of task={tag} was {future.result()}.")
    else:
        print(f"Problem with task={tag}.")


async def async_main() -> None:
    tags: list[str] = ["task1", "task2"]
    tasks: list[asyncio.Task] = []

    # Start all tasks before adding the callback
    for task_tag in tags:
        task = asyncio.create_task(
            unlikely_to_return(tag=task_tag)
        )
        task.add_done_callback(
            partial(handle_return_callback, task_tag)
        )
        tasks.append(task)

    # Alternatively, use asyncio.gather()
    # At this point, the functions are already running concurrently. And the result will␣
↪be processed
    # by the callback AS "SOON" AS THEY ARE AVAILABLE.
    print("Awaiting results...")
    for task in tasks:
        await task


def main() -> None:
    try:
        asyncio.run(async_main())
    except KeyboardInterrupt:
        pass
    except Exception as e:
        print(e)


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

In the `callback` paradigm, besides the function that does the actual task, as in the prior example, we have to make a, to no one's surprise, callback function to process the results as they come.

We do so with

```
def handle_return_callback(tag: str, future: asyncio.Future) -> None:
    """
    Callback example for asyncio.Future
    :param tag: An example parameter, in this case, a tag
    :param future: A asyncio.Future is expected to be the last parameter
    of the callback.
    :return: Nothing.
    """
    if future is not None and future.done():
        print(f"The result of task={tag} was {future.result()}.")
    else:
        print(f"Problem with task={tag}.")
```

In this case, the `callback` must receive a `asyncio.Future` and process it. Test the future for `None` in case the task fails for any reason.

Aside from that, there are only two key differences with the `await` logic example we showed before,

1. The callback must be added with `task.add_done_callback()`, remember to use `partial()` if the callback has other parameters besides the `Future`
2. `await` for the tasks at the end, not because this script will process it (it will be processed as they come by its `callback`), but because otherwise the main script will return and (most likely) nothing will be done.

```
async def async_main() -> None:
    tags: list[str] = ["task1", "task2"]
    tasks: list[asyncio.Task] = []

    # Start all tasks before adding the callback
    for task_tag in tags:
        task = asyncio.create_task(
            unlikely_to_return(tag=task_tag)
        )
        task.add_done_callback(
            partial(handle_return_callback, task_tag)
        )
        tasks.append(task)

    # Alternatively, use asyncio.gather()
    # At this point, the functions are already running concurrently. And the result will␣
↪be processed
    # by the callback AS "SOON" AS THEY ARE AVAILABLE.
    print("Awaiting results...")
    for task in tasks:
        await task
```

But enough talk… Have at you! Let's run the code with

```
cd ~/ros2_tutorials_preamble/python/minimalist_package/
python3 -m minimalist_package.minimalist_async.async_callback_example
```

Depending on our luck, we will have a very illustrative result like the one below. This example shows that, with the `callback` logic, when the second task ends before the first one, it will be automatically processed by its `callback`.

```
Awaiting results...
task1 retry needed (roll = 0.6248308966234916 > 0.1).
task2 retry needed (roll = 0.24259714032999036 > 0.1).
task1 retry needed (roll = 0.1996764883575476 > 0.1).
task2 Done.
The result of task=task2 was 0.09069407383542283.
task1 retry needed (roll = 0.6700777523785147 > 0.1).
task1 retry needed (roll = 0.7344216907108979 > 0.1).
task1 retry needed (roll = 0.4907223062034761 > 0.1).
task1 retry needed (roll = 0.20026037098687932 > 0.1).
task1 Done.
The result of task=task1 was 0.09676678954317675.
```

Can you feel the new synaptic connections?

CHAPTER

FIVE

# MAKING YOUR PYTHON PACKAGE INSTALLABLE

**Caution**

Projects with a `pyproject.toml` are currently the default for pure Python projects. In **ROS2**, currently we fully rely on `setup.py` approach using setuptools therefore that's what I discuss herein.

## 5.1 Use a venv

We already know that it is a good practice to *When you want to isolate your environment, use venv*. So, let's turn that into a reflex and do so for this whole section.

```
cd ~
source ros2tutorial_venv/bin/activate
```

## 5.2 The `setup.py`

**In this step, we'll work on this.**

```
python/
`-- minimalist_package
    |-- minimalist_package
    |   |-- __init__.py
    |   |-- _minimalist_class.py
    |   |-- minimalist_async
    |   |   |-- __init__.py
    |   |   |-- _unlikely_to_return.py
    |   |   |-- async_await_example.py
    |   |   `-- async_callback_example.py
    |   `-- minimalist_script.py
    |-- setup.py
    `-- test
        `-- test_minimalist_class.py
```

Has Python Packaging ever looked daunting to you? Of course not, but let's go through a quick overview of how we can get this done.

First, in the following directory, `~/ros2_tutorials_preamble/python/minimalist_package` we create the file below.

setup.py

```
from setuptools import setup, find_packages

package_name = 'minimalist_package'

setup(
    name=package_name,
    version='23.6.0',
    packages=find_packages(exclude=['test']),
    install_requires=['setuptools'],
    zip_safe=True,
    maintainer='Murilo M. Marinho',
    maintainer_email='murilomarinho@ieee.org',
    description='A minimalist package',
    license='MIT',
    entry_points={
        'console_scripts': [
            'minimalist_script = minimalist_package.minimalist_script:main',
            'async_await_example = minimalist_package.minimalist_async.async_await_
↪example:main',
            'async_callback_example = minimalist_package.minimalist_async.async_callback_
↪example:main'
        ],
    },
)
```

**Note**

By no coincidence, the `setup.py` is a Python file. We use Python to interpret it, meaning that we can process information using Python to define the arguments for the `setup()` function.

All arguments defined above are quite self-explanatory and are passed to the `setup()` function available at the `setuptools` module built into Python.

The probably most unusual part of it is the `entry_points` dictionary. In the key `console_scripts`, we can list up scripts in the package that can be used as console programs after the package is installed. Indeed, `setuptools` is rich, has a castle, and can do magic.

## 5.3 Installing `wheel`

**Warning**

In the current version of Python, if you do not install `wheel` as described herein, the following warning will be output.

```
DEPRECATION: minimalist-package is being installed using the legacy 'setup.py install
↪' method because it does not have a 'pyproject.toml'
and the 'wheel' package is not installed. pip 23.1 will enforce this behaviour change.
↪ A possible replacement is to enable the '--use-pep517'
option. Discussion can be found at https://github.com/pypa/pip/issues/8559
```

To install the package in the recommended way in this tutorial, we need `wheel`. While using the `venv`, we install it

```
python3 -m pip install wheel
```

## 5.4 Installing the Python package

We first go to the folder containing our *project* folder and we build and install the *project* folder within it using **`pip`** as follows

```
cd ~/ros2_tutorials_preamble/python
python3 -m pip install ./minimalist_package
```

which results in

```
Processing ./minimalist_package
  Preparing metadata (setup.py) ... done
Requirement already satisfied: setuptools in ~ros2tutorial_venv/lib/python3.10/site-
↪packages (from minimalist-package==23.6.0) (65.6.3)
Building wheels for collected packages: minimalist-package
  Building wheel for minimalist-package (setup.py) ... done
  Created wheel for minimalist-package: filename=minimalist_package-23.6.0-py3-none-any.
↪whl size=8608 sha256=929446a2fa81fc99fc5dec239a9f3e4439bc8fa8fe49cc4deb987d6f31b3d8b9
  Stored in directory: /private/var/folders/4k/20khytt17blf21lptscczbl00000gn/T/pip-
↪ephem-wheel-cache-j3a0f5xy/wheels/00/16/ef/
↪863b898c6ea4d32d47a24fda31f80cbc9cb1063742032b7d49
Successfully built minimalist-package
Installing collected packages: minimalist-package
Successfully installed minimalist-package-23.6.0
```

Done!

## 5.5 Running the newly available scripts

After installing, we have access to the scripts (and packages). For instance, we can do

```
minimalist_script
```

which will return the friendly

```
Howdy!
Howdy!
Howdy!
```

The other two scripts are also available, for instance, we can do

```
async_await_example
```

which will return something similar to

```
Awaiting results...
task1 retry needed (roll = 0.1534174185325745 > 0.1).
task2 retry needed (roll = 0.35338687437350913 > 0.1).
task1 Done.
```

(continues on next page)

(continued from previous page)

```
task2 retry needed (roll = 0.3877920607121429 > 0.1).
The result of task=task1 was 0.07646509818952207.
task2 retry needed (roll = 0.7010015915930288 > 0.1).
task2 retry needed (roll = 0.8907576123834621 > 0.1).
task2 retry needed (roll = 0.4233577578392548 > 0.1).
task2 retry needed (roll = 0.7512028176843422 > 0.1).
task2 retry needed (roll = 0.33501957024540663 > 0.1).
task2 Done.
The result of task=task2 was 0.09239734738421612.
```

## 5.6 Importing things from the installed package

We first run an interactive session with

```
python3
```

we can then interact with it as any other installed package

```
>>> from minimalist_package import MinimalistClass
>>> mc = MinimalistClass()
>>> print(mc.get_private_attribute())
20.0
```

Hooray!

## 5.7 Uninstalling packages

Given that we installed it using **`pip`**, removing it is also a breeze. We do

```
python3 -m pip uninstall minimalist_package
```

which will return something similar to

```
Found existing installation: minimalist-package 23.6.0
Uninstalling minimalist-package-23.6.0:
  Would remove:
    /home/murilo/ros2tutorial_venv/bin/async_await_example
    /home/murilo/ros2tutorial_venv/bin/async_callback_example
    /home/murilo/ros2tutorial_venv/bin/minimalist_script
    /home/murilo/ros2tutorial_venv/lib/python3.10/site-packages/minimalist_package-23.6.
↪0.dist-info/*
    /home/murilo/ros2tutorial_venv/lib/python3.10/site-packages/minimalist_package/*
Proceed (Y/n)?
```

and just press `ENTER`, resulting in the package being uninstalled

```
Successfully uninstalled minimalist-package-23.6.0
```

CHAPTER

SIX

# ROS2 INSTALLATION

**Note**

This tutorial is an abridged version of the original ROS 2 Documentation. This tutorial considers a fresh Ubuntu Desktop (not Server) 24.04 LTS installation, that you have super user access and common sense. It might work in other cases, but those have not been tested in this tutorial.

**Warning**

All commands must be followed to the letter, in the precise order described herein. Any deviation from what is described might cause unspecified problems and not all of them are easily solvable.

## 6.1 Update `apt` packages

**Hint**

You can quickly open a new terminal window by pressing CTRL+ATL+T.

After a fresh install, update and upgrade all **apt** packages.

```
sudo apt update && sudo apt upgrade -y
```

## 6.2 Install a few pre-requisites

```
sudo apt install -y software-properties-common curl terminator git
```

Namely:

| | |
|---|---|
| software-properties-common | Allows us to access the ROS2 packages using **apt**. |
| curl | Helps download installation/configuration files from the terminal. |
| terminator | ROS uses plenty of terminals, so this helps keep one's sanity intact by enabling the management of several terminals in a single window. Despite what some might say, this particular terminator has no interest whatsoever in Sarah Connor. |
| git | The trendy source control program everyone mentions in their CV. You might be interested in knowing why it's called `git`. |

## 6.3 Add ROS2 sources

Your **apt** needs to know where the ROS2 packages can be found and to be able to verify their authenticity. After setting up the **apt** sources, the local package list must be updated. The following commands will do all that magic.

```
sudo add-apt-repository universe
export ROS_APT_SOURCE_VERSION=$(curl -s https://api.github.com/repos/ros-infrastructure/
↪ros-apt-source/releases/latest | grep -F "tag_name" | awk -F\" '{print $4}')
curl -L -o /tmp/ros2-apt-source.deb "https://github.com/ros-infrastructure/ros-apt-
↪source/releases/download/${ROS_APT_SOURCE_VERSION}/ros2-apt-source_${ROS_APT_SOURCE_
↪VERSION}.$(. /etc/os-release && echo $VERSION_CODENAME)_all.deb" # If using Ubuntu␣
↪derivates use $UBUNTU_CODENAME
sudo dpkg -i /tmp/ros2-apt-source.deb
sudo apt update && sudo apt upgrade -y
```

## 6.4 Install ROS2 packages

There are plenty of ways to install ROS2, the following will suffice for now.

```
sudo apt install -y ros-jazzy-desktop ros-dev-tools
```

## 6.5 Set up system environment to find ROS2

ROS2 packages are implemented in such a way that they live peacefully in the `/opt/ros/{ROS_DISTRO}` folder in your Ubuntu. A given terminal window or program will only know that ROS2 exists, and which version you want to use, if you run a setup file *for each terminal, every time you open a new one*.

The `~/.bashrc` file can be used for that exact purpose as, in Ubuntu, that is the file that configures each terminal window for a given user.

**TL;DR** just run this **ONCE AND ONLY ONCE**

```
echo "# Source ROS2 Jazzy, as instructed in https://ros2-tutorial.readthedocs.io" >> ~/.
↪bashrc
echo "source /opt/ros/jazzy/setup.bash" >> ~/.bashrc
source ~/.bashrc
```

## 6.6 Check if it works

If the following command

```
ros2
```

outputs something similar to what is shown below, then it worked! Otherwise, it didn't!

```
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
  bag        Various rosbag 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`
```

(continues on next page)

(continued from previous page)

```
Call `ros2 <command> -h` for more detailed usage.
```

CHAPTER

SEVEN

# TERMINATOR IS LIFE

> **Note**
>
> You can refer to the project's documentation for more info.

After installing **`terminator`** as instructed in the last section, the default terminal window will be automatically updated to use it.

## 7.1 Shortcuts

To prevent repetition, let's go through the most relevant terminator shortcuts only once, here, now.

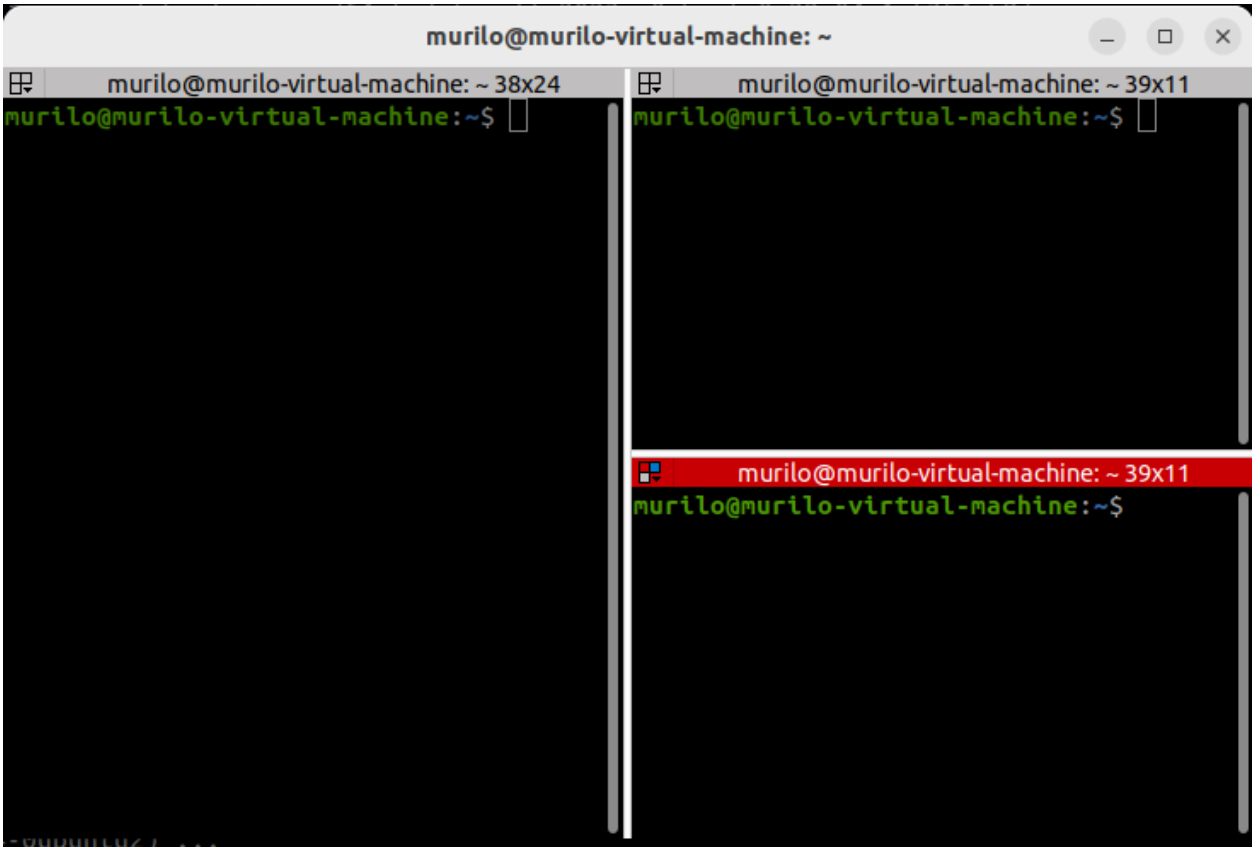
Table 1: Terminator Shortcuts

| Shortcut | Description |
|---|---|
| `CTRL+ALT+T` | Open a new terminal window using your default viewer. |
| `SHIFT+CTRL+E` | **Horizontally** split the currently focused window by adding a new terminal. |
| `SHIFT+CTRL+O` | **Vertically** split the currently focused window by adding a new terminal. |

For example, pressing the following combination:

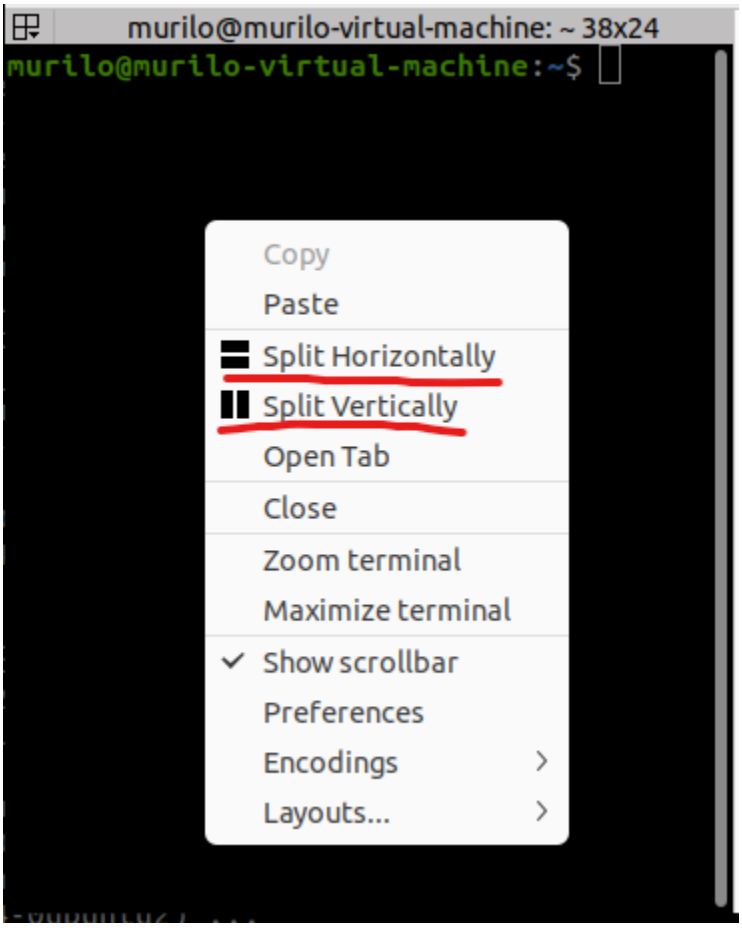
1. `CTRL+ALT+T`
2. `SHIFT+CTRL+E`
3. `SHIFT+CTRL+O`

Will result in three terminal windows that look like so.




## 7.2 OK, but what if shortcuts scare me

Instead of using shortcuts, a context menu can be opened by right-clicking a terminal window. Then, you can choose to *Split Horizontally* or *Split Vertically* to achieve the same results.


murilo@murilo-virtual-machine: ~ 38x24
murilo@murilo-virtual-machine:~$
Copy
Paste
Split Horizontally
Split Vertically
Open Tab
Close
Zoom terminal
Maximize terminal
Show scrollbar
Preferences
Encodings
Layouts...

CHAPTER

EIGHT

# WORKSPACE SETUP

Similar to how ROS2 files are installed in `/opt/ros/{ROS_DISTRO}` so that you can have several distributions installed simultaneously, you can also have many separate workspaces in your system.

In addition, because files in the `/opt` folder require superuser privileges (for good reasons), having a user-wide workspace is the accepted practice. They call this an **overlay**.

## 8.1 Setting up

In ROS2, a workspace is nothing more than a folder in which all your packages are contained.

No, really, you just need to make a folder, e.g. the one we will use throughout these tutorials.

```
cd ~
mkdir -p ros2_tutorial_workspace/src
```

It is common practice to have all source files inside the `src` folder, so we will also do so for these tutorials. Nonetheless, it is not a strict requirement.

## 8.2 First build

Regardless of it being a currently empty project, we run **`colcon`** once to set up the environment and illustrate a few things. The program **`colcon`** is the build system of ROS2 and will be described in more detail later.

For now, run

```
cd ~/ros2_tutorial_workspace
colcon build
```

for which the output will be something similar to

```
Summary: 0 packages finished [0.08s]
```

given that we have an empty workspace, no surprise here.

The folders `build`, `install`, and `log` have been generated automatically by **`colcon`**. The project structure becomes as follows.

```
ros2_tutorial_workspace/
|-- build
|-- install
|-- log
`-- src
```

Inside the `install` folder lie everything in the project that can be accessed by the users.

> **Note**
>
> An easy way to understand if something is not accessible via **ROS2** commands is if they cannot be found inside your *install* folder.

Do the following just once, so that all terminal windows automatically source this new workspace for you.

```
echo "# Source the ROS2 overlay, as instructed in https://ros2-tutorial.readthedocs.io" >
↪> ~/.bashrc
echo "source ~/ros2_tutorial_workspace/install/setup.bash" >> ~/.bashrc
source ~/.bashrc
```

However, since our workspace is currently empty, there's not much we can do with it. Let's add some content.

CHAPTER

NINE

# CREATE PACKAGES (ROS2 PKG CREATE)

**ROS2** packages are the most basic element in the **ROS2** infrastructure. You must be able to create them correctly and consistently. Everything else that **ROS2** can offer rotates about the effective use of packages.

ROS2 has a tool to help create package templates. We can get all available options by running

```
ros2 pkg create -h
```

which outputs a list of handy options to populate the package template with useful files. Namely, the four emphasized ones.

```
usage: ros2 pkg create [-h] [--package-format {2,3}] [--description DESCRIPTION] [--
↪license LICENSE] [--destination-directory DESTINATION_DIRECTORY] [--build-type {cmake,
↪ament_cmake,ament_python}]
                 [--dependencies DEPENDENCIES [DEPENDENCIES ...]] [--maintainer-email␣
↪MAINTAINER_EMAIL] [--maintainer-name MAINTAINER_NAME] [--node-name NODE_NAME] [--
↪library-name LIBRARY_NAME]
                 package_name

Create a new ROS 2 package

positional arguments:
  package_name          The package name

options:
  -h, --help            show this help message and exit
  --package-format {2,3}, --package_format {2,3}
                        The package.xml format.
  --description DESCRIPTION
                        The description given in the package.xml
  --license LICENSE     The license attached to this package; this can be an arbitrary␣
↪string, but a LICENSE file will only be generated if it is one of the supported␣
↪licenses (pass '?' to get a list)
  --destination-directory DESTINATION_DIRECTORY
                        Directory where to create the package directory
  --build-type {cmake,ament_cmake,ament_python}
                        The build type to process the package with
  --dependencies DEPENDENCIES [DEPENDENCIES ...]
                        list of dependencies
  --maintainer-email MAINTAINER_EMAIL
                        email address of the maintainer of this package
  --maintainer-name MAINTAINER_NAME
```

(continues on next page)

(continued from previous page)

```
                        name of the maintainer of this package
  --node-name NODE_NAME
                        name of the empty executable
  --library-name LIBRARY_NAME
                        name of the empty library
```

Although it is recommended to use some of the other options as well, such as `--license`, `--maintainer-email`, and `--maintainer-name`, they would make the commands too long for the purposes of this tutorial. Remember to choose the most suitable values for your application.

CHAPTER

TEN

# CREATING A PYTHON PACKAGE (FOR `AMENT_PYTHON`)

**Note**

This is **NOT** the only way to build Python packages in ROS2.

Packages in ROS2 can either rely on **`CMake`** or directly use setup tools available in Python. For pure Python projects, it might be easier to use **`ament_python`**, so we start this tutorial with it.

Let us build the simplest of Python packages and start from there.

```
cd ~/ros2_tutorial_workspace/src
ros2 pkg create the_simplest_python_package \
--build-type ament_python
```

**Hint**

If you don't explicitly define the maintainer name and email, **`ros2 pkg create`** will try to:

1. Define the maintainer's name as the currently logged-in user's name (see source and source).
2. Define the maintainer's email by getting it from **`git`** (see source). It will get whatever is defined with **`git config --global user.email`**.

which will result in the output below, meaning the package has been generated successfully.

```
going to create a new package
package name: the_simplest_python_package
destination directory: /root/ros2_tutorial_workspace/src
package format: 3
version: 0.0.0
description: TODO: Package description
maintainer: ['root <murilo.marinho@manchester.ac.uk>']
licenses: ['TODO: License declaration']
build type: ament_python
dependencies: []
creating folder ./the_simplest_python_package
creating ./the_simplest_python_package/package.xml
creating source folder
creating folder ./the_simplest_python_package/the_simplest_python_package
creating ./the_simplest_python_package/setup.py
```

(continues on next page)

(continued from previous page)

```
creating ./the_simplest_python_package/setup.cfg
creating folder ./the_simplest_python_package/resource
creating ./the_simplest_python_package/resource/the_simplest_python_package
creating ./the_simplest_python_package/the_simplest_python_package/__init__.py
creating folder ./the_simplest_python_package/test
creating ./the_simplest_python_package/test/test_copyright.py
creating ./the_simplest_python_package/test/test_flake8.py
creating ./the_simplest_python_package/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
BSL-1.0
BSD-2.0
BSD-2-Clause
BSD-3-Clause
GPL-3.0-only
LGPL-3.0-only
MIT
MIT-0
```

We can build the workspace that now has this empty package using **colcon**

```
cd ~/ros2_tutorial_workspace
colcon build
```

which will now output

```
Starting >>> the_simplest_python_package
Finished <<< the_simplest_python_package [0.49s]

Summary: 1 package finished [0.55s]
```

meaning that **colcon** successfully built the example package. It is important to note that **ROS2** does not know this package exists yet.

CHAPTER

ELEVEN

# CREATING A PYTHON NODE WITH A TEMPLATE (FOR AMENT_PYTHON)

It is always good to rely on the templates available in **ros2 pkg create**, mostly because the best practices for packaging might change between ROS2 versions.

Let us use the template for creating a package with a Node, as follows.

```
cd ~/ros2_tutorial_workspace/src
ros2 pkg create python_package_with_a_node \
--build-type ament_python \
--node-name sample_python_node
```

Which will output many things in common with the prior example, but with two major differences.

1. It generates a template Node
2. The `setup.py` has information about the Node.

```
going to create a new package
package name: python_package_with_a_node
destination directory: /root/ros2_tutorial_workspace/src
package format: 3
version: 0.0.0
description: TODO: Package description
maintainer: ['root <murilo.marinho@manchester.ac.uk>']
licenses: ['TODO: License declaration']
build type: ament_python
dependencies: []
node_name: sample_python_node
creating folder ./python_package_with_a_node
creating ./python_package_with_a_node/package.xml
creating source folder
creating folder ./python_package_with_a_node/python_package_with_a_node
creating ./python_package_with_a_node/setup.py
creating ./python_package_with_a_node/setup.cfg
creating folder ./python_package_with_a_node/resource
creating ./python_package_with_a_node/resource/python_package_with_a_node
creating ./python_package_with_a_node/python_package_with_a_node/__init__.py
creating folder ./python_package_with_a_node/test
creating ./python_package_with_a_node/test/test_copyright.py
creating ./python_package_with_a_node/test/test_flake8.py
creating ./python_package_with_a_node/test/test_pep257.py
creating ./python_package_with_a_node/python_package_with_a_node/sample_python_node.py
```

(continues on next page)

(continued from previous page)

```
[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
BSL-1.0
BSD-2.0
BSD-2-Clause
BSD-3-Clause
GPL-3.0-only
LGPL-3.0-only
MIT
MIT-0
```

Then, we can build the workspace as usual to consider the new package as well.

```
cd ~/ros2_tutorial_workspace
colcon build
```

which will result in going through the package we created in the prior example and the current one.

```
Starting >>> python_package_with_a_node
Starting >>> the_simplest_python_package
Finished <<< the_simplest_python_package [0.56s]
Finished <<< python_package_with_a_node [0.56s]

Summary: 2 packages finished [0.62s]
```

CHAPTER

# TWELVE

# ALWAYS SOURCE AFTER YOU BUILD

When creating new packages or modifying existing ones, many changes will not be visible by the system unless our workspace is re-built and re-sourced.

For example, if we try the following in the terminal window we used to first build this example package

```
ros2 run python_package_with_a_node sample_python_node
```

it will not work and will output

```
Package 'python_package_with_a_node' not found
```

As the workspace grows bigger and the packages more complex, figuring out such errors becomes a considerable hassle. My suggestion is to always source after a build, so that sourcing errors can always be ruled out.

```
cd ~/ros2_tutorial_workspace
colcon build
source install/setup.bash
```

**Note**

Remember to source this newly built workspace in *every* terminal in which you want to use these packages. In these other terminals, you can simply source the workspace.

```
cd ~/ros2_tutorial_workspace
source install/setup.bash
```

## 12.1 Go to an installed package

After sourcing an installed package, we can see the installed contents. This can be particularly useful if you want to inspect the contents of packages that are installed, but have not been built by you.

```
cd $(ros2 pkg prefix python_package_with_a_node)
```

This should move the terminal to the folder `~/ros2_tutorial_workspace/install/python_package_with_a_node`.

The simplified file structure is shown below. The `lib` folder will hold, for instance, nodes. The `share` folder will hold, for instance, launch files and other installable files that might be relevant for the package.

```
python_package_with_a_node/
|-- lib
|   `-- python_package_with_a_node
|       `-- sample_python_node
`-- share
```

Whenever the nodes you run do not seem to match the source code you are working on, you should look at this installation folder. Programs such as **ros2 run**, with the **colcon** instructions shown herein, will exclusively look at the content in your install folder.

**Note**

You can use the flag `--share` to go straight to the `share` folder.

```
cd $(ros2 pkg prefix python_package_with_a_node --share)
```

## 12.2 Troubleshooting tips

In this section I show some troubleshooting tips for common issues related to **colcon build**. You might not understand all of them at this stage of the tutorial. Check back to this section in case problems arise.

### 12.2.1 Package not found

One important tool to assist in case your package is not found is **ros2 pkg list**. It can be called as follows.

```
ros2 pkg list
```

It will output a large number of packages even for the most basic installations of **ROS2**. If you are looking for a particular package, you can use **grep** which is more actively used (and explained) in other parts of this tutorial. For instance, if you are looking for `python_package_with_a_node` you can do as follows.

```
ros2 pkg list | grep python_package_with_a_node
```

This will either output nothing if the package is not found or it will output the name of the package, as follows.

```
python_package_with_a_node
```

### 12.2.2 Fixing a dirty state in your `colcon build`

Sometimes, a problematic build might not go away even with repeated calls to **colcon build**.

The most common cause of this is when, by mistake, a terminal with an active **venv** was used when calling **colcon build**. The usual error message will look like so.

```
Traceback (most recent call last):
  File "/opt/ros/jazzy/share/ament_cmake_core/cmake/core/package_xml_2_cmake.py", line␣
↪22, in <module>
    from catkin_pkg.package import parse_package_string
ModuleNotFoundError: No module named 'catkin_pkg'
```

To fix this, you must

1. Deactivate the **venv**.

2. Remove the `build`, `install`, and `log` folders.
3. Rebuild and resource in a clean terminal, without a **`venv`**.

In this tutorial, this would be equivalent to doing

> **Caution**
>
> Remember that **`rm`** can cause *permanent* loss of data. Please understand the following command and its implications *before* executing it.

```
deactivate
cd ~/ros2_tutorial_workspace
rm -rf build/ install/ log/
colcon build
source install/setup.bash
```

In rare cases, even without using a **`venv`**, the workspace can be left in an unclean state in which older build artifacts cause build and runtime issues, such as failed builds and programs that do not seem to match their intended source code. These artifacts might include old files that should have been removed, issues with dependencies, and so on. In this case, removing the `build`, `install`, and `log` folders can be useful.

### 12.2.3 Dependency issues in the first `colcon build`

It might also be the case that certain packages fail to build after `build`, `install`, and `log` are removed, or that the build only works after **`colcon`** is called twice in a row.

This is usually because the dependencies of the packages in your workspace are poorly configured and, in consequence, ROS2 is not building them in the correct order. If your workspace does not build properly after being cleaned as mentioned above, you must correct its dependencies until it builds properly.

This can usually done by verifying if your `package.xml` has the correct dependencies.

CHAPTER

# THIRTEEN

# RUNNING A NODE (`ROS2 RUN`)

The most basic way of running a Node is using the ROS2 tool **`ros2 run`**.

More information on it can be obtained through

```
ros2 run -h
```

which returns the most relevant arguments `package_name` and `executable_name`.

```
usage: ros2 run [-h] [--prefix PREFIX] package_name executable_name ...

Run a package specific executable

positional arguments:
  package_name     Name of the ROS package
  executable_name  Name of the executable
  argv             Pass arbitrary arguments to the executable

options:
  -h, --help       show this help message and exit
  --prefix PREFIX  Prefix command, which should go before the executable. Command must␣
↪be wrapped in quotes if it contains spaces (e.g. --prefix 'gdb -ex run --args').
```

Back to our example, with a properly sourced terminal, the example node can be executed with

```
ros2 run python_package_with_a_node sample_python_node
```

which will now correctly output

```
Hi from python_package_with_a_node.
```

## 13.1 Troubleshooting tips

If **`ROS2`** is unable to find the node, but it is able to find the package, then you can rely on **`ros2 pkg executables`**. For instance, you can run as follows.

```
ros2 pkg executables python_package_with_a_node
```

The command, at this stage, should output the following.

```
python_package_with_a_node sample_python_node
```

This shows that **ROS2** has been correctly able to find `sample_python_node` within `python_package_with_a_node`. If the command outputs nothing, this means that no nodes were found.

CHAPTER

FOURTEEN

# CREATING A PYTHON NODE FROM SCRATCH (FOR `AMENT_PYTHON`)

**See also**

The official API documentation: https://docs.ros.org/en/jazzy/p/rclpy/rclpy.html

**TL;DR Making an `ament_python` Node**

1. Modify `package.xml` with any additional dependencies.
2. Create the Node.
3. Modify the `setup.py` file.

Let us add an additional Node to our **`ament_python`** package that actually uses ROS2 functionality. These are the steps that must be taken, in general, to add a new Node.

## 14.1 File structure

In this section, we will be modifying or creating the following files.

```
python_package_with_a_node
|-- package.xml
|-- python_package_with_a_node
|   |-- __init__.py
|   |-- print_forever_node.py
|   `-- sample_python_node.py
|-- resource
|   `-- python_package_with_a_node
|-- setup.cfg
|-- setup.py
`-- test
    |-- test_copyright.py
    |-- test_flake8.py
    `-- test_pep257.py
```

## 14.2 Handling dependencies (`package.xml`)

The `package.xml` was automatically generated by **`ros2 pkg create`** and holds basic information about the package.

One important role of `package.xml` is to declare dependencies with other ROS2 packages. It is common for new Nodes to have additional dependencies, so we will cover that here. For any ROS2 package, we must modify the `package.xml` to add new dependencies.

In this toy example, let us add `rclpy` as a dependency because it is the Python implementation of the RCL (ROS Client Library). All Nodes that use anything related to ROS2 will directly or indirectly depend on that library.

By no coincidence, the `package.xml` has the `.xml` extension, meaning that it is written in XML (Extensible Markup Language).

Let us add the dependency between the `<license>` and `<test_depend>` tags. This is not a strict requirement but is where it commonly is for standard packages.

`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>python_package_with_a_node</name>
  <version>0.0.0</version>
  <description>TODO: Package description</description>
  <maintainer email="murilomarinho@ieee.org">murilo</maintainer>
  <license>TODO: License declaration</license>

  <depend>rclpy</depend>

  <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>
</package>
```

## 14.3 Creating the Node

In the directory `src/python_package_with_a_node/python_package_with_a_node`, create a new file called `print_forever_node.py`. Copy and paste the following contents into the file.

`print_forever_node.py`

```
import rclpy
from rclpy.node import Node


class PrintForever(Node):
    """A ROS2 Node that prints to the console periodically."""

```

(continues on next page)

(continued from previous page)

```
    def __init__(self):
        super().__init__('print_forever')
        timer_period: float = 0.5
        self.timer = self.create_timer(timer_period, self.timer_callback)
        self.print_count: int = 0

    def timer_callback(self):
        """Method that is periodically called by the timer."""
        self.get_logger().info(f'Printed {self.print_count} times.')
        self.print_count = self.print_count + 1


def main(args=None):
    """
    The main function.
    :param args: Not used directly by the user, but used by ROS2 to configure
    certain aspects of the Node.
    """
    try:
        rclpy.init(args=args)

        print_forever_node = PrintForever()

        rclpy.spin(print_forever_node)
    except KeyboardInterrupt:
        pass
    except Exception as e:
        print(e)


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

## 14.4 Making `ros2 run` work

We need an additional step to make it deployable in a place where **`ros2 run`** can find it.

To do so, we modify the `console_scripts` key in the `entry_points` dictionary defined in `setup.py`, to have our new node, as follows

> **Hint**
>
> `console_scripts` expects a `list` of `str` in a specific format. Hence, follow the format properly and don't forget the commas to separate elements in the `list`.

`~/ros2_tutorial_workspace/src/python_package_with_a_node/setup.py`

```
from setuptools import find_packages, setup

package_name = 'python_package_with_a_node'
```

(continues on next page)

(continued from previous page)

```
4
5  setup(
6      name=package_name,
7      version='0.0.0',
8      packages=find_packages(exclude=['test']),
9      data_files=[
10         ('share/ament_index/resource_index/packages',
11          ['resource/' + package_name]),
12         ('share/' + package_name, ['package.xml']),
13     ],
14     install_requires=['setuptools'],
15     zip_safe=True,
16     maintainer='murilo',
17     maintainer_email='murilomarinho@ieee.org',
18     description='TODO: Package description',
19     license='TODO: License declaration',
20     tests_require=['pytest'],
21     entry_points={
22         'console_scripts': [
23             'sample_python_node = python_package_with_a_node.sample_python_node:main',
24             'print_forever_node = python_package_with_a_node.print_forever_node:main'
25         ],
26     },
27 )
```

The format is straightforward, as follows

| | |
|---|---|
| `print_forever_node` | The name of the node when calling it through **`ros2 run`**. |
| `python_package_with_a_node` | The name of the package. |
| `print_forever_node` | The name of the script, without the `.py` extension. |
| `main` | The function, within the script, that will be called. In general, `main`. |

## 14.5 Build and source

We can build and source the workspace with the following command.

```
cd ~/ros2_tutorial_workspace
colcon build
source install/setup.bash
```

**Note**

For additional explanation and troubleshooting tips, see *Always source after you build*.

**Warning**

**`colcon`** will *not* work properly if your terminal has an active **`venv`**.

## 14.6 Test

And, with that, we can run

```
ros2 run python_package_with_a_node print_forever_node
```

which will output, as expected

```
[INFO] [1753518652.646459087] [print_forever]: Printed 0 times.
[INFO] [1753518653.131078795] [print_forever]: Printed 1 times.
[INFO] [1753518653.632436004] [print_forever]: Printed 2 times.
[INFO] [1753518654.132090212] [print_forever]: Printed 3 times.
[INFO] [1753518654.630924338] [print_forever]: Printed 4 times.
```

To stop, press CTRL+C on the terminal and the Node will return gracefully.

CHAPTER

# FIFTEEN

# THE PYTHON NODE, EXPLAINED

> **Note**
>
> The way that a Python Node in ROS2 works, i.e. the explanation in this section, does not depend on the building with **`ament_python`** or **`ament_cmake`**.

In a strict sense, the `print_forever_node.py` is not a minimal Node, but it does showcase most good practices in a Node that actually does something.

## 15.1 The imports

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

As in any Python code, we have to import the libraries that we will use and specific modules/classes within those libraries. With `rclpy`, there is no difference.

## 15.2 Making a subclass of `Node`

The current version of ROS2 behaves better when your custom Node is a subclass of `rclpy.node.Node`. That is achieved with

```
class PrintForever(Node):
    """A ROS2 Node that prints to the console periodically."""

    def __init__(self):
        super().__init__('print_forever')
        timer_period: float = 0.5
```

About inheritance in Python, you can check the official documentation on inheritance and on super().

In more advanced nodes, inheritance does not cut it, but that is an advanced topic to be covered some other time.

## 15.3 Use a `Timer` for periodic work (when using `rclpy.spin()`)

**Tips for the future you**

If the code relies on `rclpy.spin()`, a Timer must be used for periodic work.

In its most basic usage, periodic tasks in ROS2 must be handled by a Timer.

To do so, have the node create it with the `create_timer()` method, as follows.

```
    def __init__(self):
        super().__init__('print_forever')
        timer_period: float = 0.5
        self.timer = self.create_timer(timer_period, self.timer_callback)
        self.print_count: int = 0
```

The method that is periodically called by the Timer is, in this case, as follows. We use `self.get_logger().info()` to print to the terminal periodically.

```
    def timer_callback(self):
        """Method that is periodically called by the timer."""
        self.get_logger().info(f'Printed {self.print_count} times.')
```

In ROS2, the logging methods, i.e. `self.get_logger().info()`, are methods of the Node itself. So, the capability to log (print to the terminal) using ROS2 Nodes is dependent on the scope in which that Node exists.

## 15.4 Where the ROS2 magic happens: `rclpy.init()` and `rclpy.spin()`

All the ROS2 magic happens in some sort of `spin()` method. It is called this way because the `spin()` method will constantly loop (or spin) through **items of work**, e.g. scheduled Timer callbacks. All the **items of work** will only be effectively executed when an **executor** runs through it. For simple Nodes, such as the one in this example, the **global** executor is implicitly used. You can read a bit more about that here.

Anyhow, the point is that nothing related to ROS2 will happen unless the two following methods are called. First, `rclpy.init()` is going to initialize a bunch of ROS2 elements behind the curtains, whereas `rclpy.spin()` will block the program and, well, **spin** through Timer callbacks forever. There are alternative ways to `spin()`, but we will not discuss them right now.

```
def main(args=None):
    """
    The main function.
    :param args: Not used directly by the user, but used by ROS2 to configure
    certain aspects of the Node.
    """
    try:
        rclpy.init(args=args)

        print_forever_node = PrintForever()

        rclpy.spin(print_forever_node)
    except KeyboardInterrupt:
        pass
    except Exception as e:
        print(e)
```

## 15.5 Have a `try-catch` block for `KeyboardInterrupt`

**Important**

In the current version of the official ROS2 examples, for reasons beyond my comprehension, this step is not followed.

However, when running Nodes either in the terminal or in **PyCharm**, catching a `KeyboardInterrupt` is the only reliable way to finish the Nodes cleanly. A `KeyboardInterrupt` is emitted at a terminal by pressing `CTRL+C`, whereas it is emitted by **PyCharm** when pressing *Stop*.

That is particularly important when real robots need to be gracefully shut down (otherwise they might inadvertently start the evil robot uprising), but it also looks unprofessional when all your Nodes return with an ugly stack trace.

```
def main(args=None):
    """
    The main function.
    :param args: Not used directly by the user, but used by ROS2 to configure
    certain aspects of the Node.
    """
    try:
        rclpy.init(args=args)

        print_forever_node = PrintForever()

        rclpy.spin(print_forever_node)
    except KeyboardInterrupt:
        pass
    except Exception as e:
        print(e)
```

**Important**

Despite what is shown in the official ROS2 examples, do not add these to your code. This will cause the node to crash when shutting down and not return 0 as expected by anything automatic running your nodes. This can cause many confusing behaviours.

```
# Destroy the node explicitly
# (optional - otherwise it will be done automatically
# when the garbage collector destroys the node object)
minimal_publisher.destroy_node()
rclpy.shutdown()
```

## 15.6 Document your code with Docstrings

As simple as a code might look for you right now, it needs to be documented for anyone you work with, including the future you. In a few weeks/months/years time, the `BeStNoDeYouEvErWrote (TM)` might be indistinguishable from Yautja Language.

Add as much description as possible to classes and methods, using the Docstring Convention.

Example of a class:

```
class PrintForever(Node):
    """A ROS2 Node that prints to the console periodically."""
```

Example of a method:

```
    def timer_callback(self):
        """Method that is periodically called by the timer."""
```

CHAPTER

SIXTEEN

# CREATING A PYTHON LIBRARY (FOR AMENT_PYTHON)

Let us start, as already recommended in this tutorial, with a template by **ros2 pkg create**.

```
cd ~/ros2_tutorial_workspace/src
ros2 pkg create python_package_with_a_library \
--build-type ament_python \
--library-name sample_python_library
```

which outputs the forever beautiful wall of text we're now used to, with a minor difference regarding the additional library template, as highlighted below.

```
going to create a new package
package name: python_package_with_a_library
destination directory: /root/ros2_tutorial_workspace/src
package format: 3
version: 0.0.0
description: TODO: Package description
maintainer: ['root <murilo.marinho@manchester.ac.uk>']
licenses: ['TODO: License declaration']
build type: ament_python
dependencies: []
library_name: sample_python_library
creating folder ./python_package_with_a_library
creating ./python_package_with_a_library/package.xml
creating source folder
creating folder ./python_package_with_a_library/python_package_with_a_library
creating ./python_package_with_a_library/setup.py
creating ./python_package_with_a_library/setup.cfg
creating folder ./python_package_with_a_library/resource
creating ./python_package_with_a_library/resource/python_package_with_a_library
creating ./python_package_with_a_library/python_package_with_a_library/__init__.py
creating folder ./python_package_with_a_library/test
creating ./python_package_with_a_library/test/test_copyright.py
creating ./python_package_with_a_library/test/test_flake8.py
creating ./python_package_with_a_library/test/test_pep257.py
creating folder ./python_package_with_a_library/python_package_with_a_library/sample_
↪python_library
creating ./python_package_with_a_library/python_package_with_a_library/sample_python_
↪library/__init__.py

[WARNING]: Unknown license 'TODO: License declaration'.  This has been set in the␣
↪package.xml, but no LICENSE file has been created.
```

(continues on next page)

(continued from previous page)

```
It is recommended to use one of the ament license identifiers:
Apache-2.0
BSL-1.0
BSD-2.0
BSD-2-Clause
BSD-3-Clause
GPL-3.0-only
LGPL-3.0-only
MIT
MIT-0
```

## 16.1 The folders/files, Mason, what do they mean?

The ROS2 package created from the template has a structure like so. In particular, we can see that `python_package_with_a_library` is repeated twice in a row. This is a common source of error, so don't forget! The first is the name of the **ROS2** package, and the second is the name of Python package that will be installed by **ROS2**.

```
python_package_with_a_library/
|-- package.xml
|-- python_package_with_a_library
|   |-- __init__.py
|   `-- sample_python_library
|       `-- __init__.py
|-- resource
|   `-- python_package_with_a_library
|-- setup.cfg
|-- setup.py
`-- test
    |-- test_copyright.py
    |-- test_flake8.py
    `-- test_pep257.py
```

We learned the meaning of most of those in the preamble, namely *(Murilo's) Python Best Practices*. To quickly clarify a few things, see the table below.

Table 1: ROS2 Python package folders/files explained

| File/Directory | Meaning |
|---|---|
| `python_package_with_a_library` | The ROS2 package folder. |
| `python_package_with_a_library/ python_package_with_a_library` | The Python package, as we saw in the preamble. |
| `sample_python_library` | The module corresponding to our sample library. |
| `resource/python_package_with_a_library` | A file for ROS2 to index this package correctly. See Resource file. |
| `test` | The folder containing the tests, as we already saw in the preamble. |
| `setup.cfg` | Used by setup.py, see setup.cfg docs. |
| `setup.py` | The instructions to make the package installable, as we saw in the preamble. |

## 16.2 Overview of the library

> **Hint**
>
> If you have created the bad habit of declaring all/too many things in your `__init__.py` file, take the hint and start breaking the definitions into different files and use the `__init__.py` just to export the relevant parts of your library.

For the sake of the example, let us create a library with a Python `function` and another one with a `class`. To guide our next steps, we first draw a quick overview of what our `python_package_with_a_library` will look like.

```
python_package_with_a_library/
|-- package.xml
|-- python_package_with_a_library
|   |-- __init__.py
|   `-- sample_python_library
|       |-- __init__.py
|       |-- _sample_class.py
|       `-- _sample_function.py
|-- resource
|   `-- python_package_with_a_library
|-- setup.cfg
|-- setup.py
`-- test
    |-- test_copyright.py
    |-- test_flake8.py
    `-- test_pep257.py
```

With respect to the highlighted files, we will

1. Create the `_sample_function.py`.
2. Create the `_sample_class.py`.
3. Modify `__init__.py` to use the new function and class.

All other files and directories will remain as-is, in the way they were generated by **`ros2 pkg create`**.

## 16.3 Create the sample function

Create a new file with the following contents and name.

`~/ros2_tutorial_workspace/src/python_package_with_a_library/python_package_with_a_library/sample_python_library/_sample_function.py`

```
def sample_function_for_square_of_sum(a: float, b: float) -> float:
    """Returns the square of a sum (a + b)^2 = a^2 + 2ab + b^2"""
    return a**2 + 2*a*b + b**2
```

The function has two parameters, `a` and `b`. For simplicity, we're expecting arguments of type `float` and returning a `float`, but it could be any Python function.

## 16.4 Create the sample class

Create a new file with the following contents and name.

`~/ros2_tutorial_workspace/src/python_package_with_a_library/python_package_with_a_library/sample_python_library/_sample_class.py`

```python
class SampleClass:
    """A sample class to check how they can be imported by other ROS2 packages."""

    def __init__(self, name: str):
        self._name = name

    def get_name(self) -> str:
        """
        Gets the name of this instance.
        :return: This name.
        """
        return self._name
```

The class is quite simple with a private data member and a method to retrieve it.

## 16.5 Modify the `__init__.py`

`~/ros2_tutorial_workspace/src/python_package_with_a_library/python_package_with_a_library/sample_python_library/__init__.py`

```python
from python_package_with_a_library.sample_python_library._sample_class import SampleClass
from python_package_with_a_library.sample_python_library._sample_function import sample_function_for_square_of_sum
```

The `__init__.py` file should import from the internal modules to expose their contents to other packages.

The absolute path in terms of **ROS2** is needed to guarantee this will work well. It is clever to stay away from package relative imports in this case.

## 16.6 Build and source

No surprise here, right?

```
cd ~/ros2_tutorial_workspace
colcon build
source install/setup.bash
```

> **Note**
>
> For additional explanation and troubleshooting tips, see *Always source after you build*.

> **Warning**
>
> **colcon** will *not* work properly if your terminal has an active **venv**.

If it builds without any unexpected issues, we're good to go!

CHAPTER

SEVENTEEN

# USING A PYTHON LIBRARY FROM ANOTHER PACKAGE (FOR AMENT_PYTHON)

Let us create a package with a Node that uses the library we created in the prior example.

Note that we must add the `python_package_with_a_library` as a dependency to our new package. The easiest way to do so is through **`ros2 pkg create`**. We also add `rclpy` as a dependency so that our Node can do something useful.

```
cd ~/ros2_tutorial_workspace/src
ros2 pkg create python_package_that_uses_the_library \
--dependencies rclpy python_package_with_a_library \
--build-type ament_python \
--node-name node_that_uses_the_library
```

resulting in yet another version of our favorite wall of text

```
going to create a new package
package name: python_package_that_uses_the_library
destination directory: /root/ros2_tutorial_workspace/src
package format: 3
version: 0.0.0
description: TODO: Package description
maintainer: ['root <murilo.marinho@manchester.ac.uk>']
licenses: ['TODO: License declaration']
build type: ament_python
dependencies: ['rclpy', 'python_package_with_a_library']
node_name: node_that_uses_the_library
creating folder ./python_package_that_uses_the_library
creating ./python_package_that_uses_the_library/package.xml
creating source folder
creating folder ./python_package_that_uses_the_library/python_package_that_uses_the_
↪library
creating ./python_package_that_uses_the_library/setup.py
creating ./python_package_that_uses_the_library/setup.cfg
creating folder ./python_package_that_uses_the_library/resource
creating ./python_package_that_uses_the_library/resource/python_package_that_uses_the_
↪library
creating ./python_package_that_uses_the_library/python_package_that_uses_the_library/__
↪init__.py
creating folder ./python_package_that_uses_the_library/test
creating ./python_package_that_uses_the_library/test/test_copyright.py
creating ./python_package_that_uses_the_library/test/test_flake8.py
creating ./python_package_that_uses_the_library/test/test_pep257.py
```

(continues on next page)

(continued from previous page)

```
creating ./python_package_that_uses_the_library/python_package_that_uses_the_library/
↪node_that_uses_the_library.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
BSL-1.0
BSD-2.0
BSD-2-Clause
BSD-3-Clause
GPL-3.0-only
LGPL-3.0-only
MIT
MIT-0
```

## 17.1 Package structure

We'll be working on the following file structure.

```
python_package_that_uses_the_library/
|-- package.xml
|-- python_package_that_uses_the_library
|   |-- __init__.py
|   `-- node_that_uses_the_library.py
|-- resource
|   `-- python_package_that_uses_the_library
|-- setup.cfg
|-- setup.py
`-- test
    |-- test_copyright.py
    |-- test_flake8.py
    `-- test_pep257.py
```

## 17.2 The sample Node

Given that it was created from a template, the file `node_that_uses_the_library.py` is currently *mostly* empty. Let us replace its contents with

`node_that_uses_the_library.py`

```
1 import random
2 import string
3
4 import rclpy
5 from rclpy.node import Node
6 from python_package_with_a_library.sample_python_library import SampleClass, sample_
  ↪function_for_square_of_sum
7
8
9 class NodeThatUsesTheLibrary(Node):
```

(continues on next page)

(continued from previous page)

```
    """A ROS2 Node that prints to the console periodically."""

    def __init__(self):
        super().__init__('node_that_uses_the_library')
        timer_period: float = 0.5
        self.timer = self.create_timer(timer_period, self.timer_callback)

    def timer_callback(self):
        """
        Method that is periodically called by the timer.
        Prints out the result of sample_function_for_square_of_sum of two random numbers,
        followed by the result of SampleClass.get_name() for an instance created with
        a ten-character-long ascii string of random characters.
        """
        a: float = random.uniform(0, 1)
        b: float = random.uniform(1, 2)
        c: float = sample_function_for_square_of_sum(a, b)
        self.get_logger().info(f'sample_function_for_square_of_sum({a},{b}) returned {c}.
↪')

        random_name_ascii: str = ''.join(random.choice(string.ascii_letters) for _ in␣
↪range(10))
        sample_class_with_random_name = SampleClass(name=random_name_ascii)
        self.get_logger().info(f'sample_class_with_random_name.get_name() '
                               f'returned {sample_class_with_random_name.get_name()}.')


def main(args=None):
    """
    The main function.
    :param args: Not used directly by the user, but used by ROS2 to configure
    certain aspects of the Node.
    """
    try:
        rclpy.init(args=args)

        node_that_uses_the_library = NodeThatUsesTheLibrary()

        rclpy.spin(node_that_uses_the_library)
    except KeyboardInterrupt:
        pass
    except Exception as e:
        print(e)


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

Indeed, the most difficult part is to make and configure the library itself. After that, to use it in another package, it is straightforward. We `import` the library.

```
import random
```

(continues on next page)

(continued from previous page)

```
import string

import rclpy
from rclpy.node import Node
from python_package_with_a_library.sample_python_library import SampleClass, sample_
↪function_for_square_of_sum
```

And then use the symbols we imported as we would with any other Python library.

```
    def timer_callback(self):
        """
        Method that is periodically called by the timer.
        Prints out the result of sample_function_for_square_of_sum of two random numbers,
        followed by the result of SampleClass.get_name() for an instance created with
        a ten-character-long ascii string of random characters.
        """
        a: float = random.uniform(0, 1)
        b: float = random.uniform(1, 2)
        c: float = sample_function_for_square_of_sum(a, b)
        self.get_logger().info(f'sample_function_for_square_of_sum({a},{b}) returned {c}.
↪')

        random_name_ascii: str = ''.join(random.choice(string.ascii_letters) for _ in␣
↪range(10))
        sample_class_with_random_name = SampleClass(name=random_name_ascii)
        self.get_logger().info(f'sample_class_with_random_name.get_name() '
                               f'returned {sample_class_with_random_name.get_name()}.')
```

## 17.3 Build and source

As always, this is needed so that our new package and node can be recognized by **`ros2 run`**.

```
cd ~/ros2_tutorial_workspace
colcon build
source install/setup.bash
```

> **Note**
>
> For additional explanation and troubleshooting tips, see *Always source after you build*.

**Warning**

**colcon** will *not* work properly if your terminal has an active **venv**.

## 17.4 Run

**Hint**

Remember that you can stop the node at any time with CTRL+C.

```
ros2 run python_package_that_uses_the_library node_that_uses_the_library
```

Which outputs something similar to the shown below, but with different numbers and strings as they are randomized.

```
[INFO] [1753585839.509922172] [node_that_uses_the_library]: sample_function_for_square_
↪of_sum(0.9787232004970391,1.7320908702316369) returned 7.348512926060575.
[INFO] [1753585839.510400755] [node_that_uses_the_library]: sample_class_with_random_
↪name.get_name() returned GQkUZgSkje.
[INFO] [1753585839.993999505] [node_that_uses_the_library]: sample_function_for_square_
↪of_sum(0.7637556876478347,1.0377535114838756) returned 3.2454353945561767.
[INFO] [1753585839.994351922] [node_that_uses_the_library]: sample_class_with_random_
↪name.get_name() returned EztAWYuFMB.
[INFO] [1753585840.494895006] [node_that_uses_the_library]: sample_function_for_square_
↪of_sum(0.26638758438777976,1.1809445792770386) returned 2.0947703919786846.
[INFO] [1753585840.495950422] [node_that_uses_the_library]: sample_class_with_random_
↪name.get_name() returned ITLIHPOMgv.
[INFO] [1753585840.994468589] [node_that_uses_the_library]: sample_function_for_square_
↪of_sum(0.5244531764161572,1.7524376840394509) returned 5.184231990426279.
[INFO] [1753585840.994695797] [node_that_uses_the_library]: sample_class_with_random_
↪name.get_name() returned LGtybBngKv.
```

CHAPTER

EIGHTEEN

# ROS2 INTERFACES (`ROS2 INTERFACE`)

Changed in version Jazzy: Added actions.

If by now you haven't particularly fallen in love with **ROS2**, fear not. Indeed, we haven't done much so far that couldn't be achieved more easily by other means.

**ROS2** begins to shine most in its interprocess communication, through what are called ROS2 interfaces. In particular, the fact that we can easily interface Nodes written in Python and C++ is a strong selling point.

`Messages` are one of the three types of ROS2 interfaces. This will most likely be the standard of communication between Nodes in your packages. We will also see the bidirectional `Services` and `Actions`.

## 18.1 Description

In **ROS2**, interfaces are files written in the ROS2 IDL (Interface Description Language). Each type of interface is described in a `.msg`, `.srv`, or `.action` file, which is then built by **`colcon`** into libraries that can be imported into your **ROS** programs.

When dealing with common robotics concepts such as geometric and sensor messages, it is good practice to use interfaces that already exist in ROS2, instead of creating new ones that serve the exact same purpose. In addition, for complicated interfaces, we can reuse `.msg` files to simplify the architecture.

## 18.2 Getting info on interfaces

We can get information about ROS2 interfaces available in our system with **`ros2 interface`**. Let us first get more information about the program usage with

```
ros2 interface -h
```

which results in

```
usage: ros2 interface [-h] Call `ros2 interface <command> -h` for more detailed usage. ..
↪.

Show information about ROS interfaces

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

Commands:
  list      List all interface types available
  package   Output a list of available interface types within one package
```

(continues on next page)

(continued from previous page)

```
  packages  Output a list of packages that provide interfaces
  proto     Output an interface prototype
  show      Output the interface definition

  Call `ros2 interface <command> -h` for more detailed usage.
```

This shows that with **`ros2 interface list`** we can get a list of all interfaces available in our workspace. That returns a huge list of interfaces, so it will not be replicated entirely here. Instead, we can run

```
ros2 interface packages
```

to get the list of packages with interfaces available, which returns something similar to

```
action_msgs
action_tutorials_interfaces
actionlib_msgs
builtin_interfaces
composition_interfaces
diagnostic_msgs
example_interfaces
geometry_msgs
lifecycle_msgs
logging_demo
map_msgs
nav_msgs
package_with_interfaces
pcl_msgs
pendulum_msgs
rcl_interfaces
rmw_dds_common
rosbag2_interfaces
rosgraph_msgs
sas_msgs
sensor_msgs
service_msgs
shape_msgs
statistics_msgs
std_msgs
std_srvs
stereo_msgs
tf2_msgs
trajectory_msgs
turtlesim
type_description_interfaces
unique_identifier_msgs
visualization_msgs
```

From those, `sensor_msgs` and `geometry_msgs` are packages to always keep in mind when looking for a suitable interface. It will help to keep your Nodes compatible with the community.

> **Warning**
>
> The `std_msgs` package, widely used in ROS1, is deprecated in ROS2 since Foxy. The `example_interfaces` somewhat takes its place, but the recommended practice is to create "semantically meaningful message types". They might remove both or either of these in future versions, so do not use them.

As an example, let us take a look into the `example_interfaces` package, containing, as the name implies, example interface types. We can do so with

```
ros2 interface package example_interfaces
```

which returns

```
example_interfaces/msg/UInt16
example_interfaces/msg/Empty
example_interfaces/action/Fibonacci
example_interfaces/msg/String
example_interfaces/msg/Int32
example_interfaces/msg/UInt32MultiArray
example_interfaces/msg/Float64MultiArray
example_interfaces/msg/Float32MultiArray
example_interfaces/srv/AddTwoInts
example_interfaces/msg/UInt8MultiArray
example_interfaces/msg/Int8
example_interfaces/msg/Int16MultiArray
example_interfaces/msg/UInt32
example_interfaces/srv/SetBool
example_interfaces/msg/Int64
example_interfaces/msg/MultiArrayDimension
example_interfaces/msg/Int8MultiArray
example_interfaces/msg/ByteMultiArray
example_interfaces/msg/Int32MultiArray
example_interfaces/srv/Trigger
example_interfaces/msg/Int64MultiArray
example_interfaces/msg/Float64
example_interfaces/msg/Byte
example_interfaces/msg/Int16
example_interfaces/msg/UInt16MultiArray
example_interfaces/msg/UInt64MultiArray
example_interfaces/msg/Char
example_interfaces/msg/UInt8
example_interfaces/msg/Bool
example_interfaces/msg/UInt64
example_interfaces/msg/WString
example_interfaces/msg/MultiArrayLayout
example_interfaces/msg/Float32
```

## 18.3 Messages

For example, let's say that we are interested in looking up the contents of `example_interfaces/msg/String`. We can do so with **`ros2 interface show`**, like so

```
ros2 interface show example_interfaces/msg/String
```

which returns the contents of the source file used to create this message

```
# This is an example message of using a primitive datatype, string.
# If you want to test with this that's fine, but if you are deploying
# it into a system you should create a semantically meaningful message type.
# If you want to embed it in another message, use the primitive data type instead.
string data
```

Basically, the comments help to emphasize that interface types with too broad meaning are unloved in ROS2. Given that these example interfaces are either unsupported or only loosely supported, do not rely on them.

The real content of the message file is `string data`, showing that it contains a single string called `data`. Using `ros2 interface show` on other example interfaces, it is easy to see how to build interesting message types to fit our needs.

## 18.4 Services

In the case of a service, let's look up the contents of `example_interfaces/srv/AddTwoInts`.

We run

```
ros2 interface show example_interfaces/srv/AddTwoInts
```

that results in

```
int64 a
int64 b
---
int64 sum
```

Notice that the `---` is what separates the `Request`, above, from the `Response` below. Anyone using this service would expect that the result would be $sum = a + b$, but this logic needs to be implemented on the Node. The service itself is just a way of bidirectional communication.

## 18.5 Actions

In the case of an action, let's look up the contents of `example_interfaces/action/Fibonacci`.

We run

```
ros2 interface show example_interfaces/action/Fibonacci
```

that results in

```
# Goal
int32 order
---
# Result
int32[] sequence
---
# Feedback
int32[] sequence
```

Notice that the two `---` are separators to show us the `Goal`, `Result`, and `Feedback` components of the action. Despite the descriptive name of `Fibonacci`, the `.action` file by itself does nothing. The logic must be implemented on the Node.

CHAPTER

NINETEEN

# CREATING A DEDICATED PACKAGE FOR CUSTOM INTERFACES

Changed in version Jazzy: Added `AmazingQuoteStamped.msg`, `MoveStraightIn2D.action` and simplified service discussion to use `AddPoints.srv`.

**Warning**

Despite this push in ROS2 towards having the users define even the simplest of message types, to define new interfaces in ROS2 we must use an **`ament_cmake`** package. It **cannot** be done with an **`ament_python`** package.

**See also**

The contents of this session were simplified in this version. A more complex example is shown in https://ros2-tutorial.readthedocs.io/en/humble/service_servers_and_clients.html.

All interfaces in ROS2 must be made in an **`ament_cmake`** package. We have so far not needed it, but for this scenario we cannot escape. Thankfully, for this we don't need to dig too deep into **`CMake`**, so fear not.

## 19.1 Overview

We will create a package with the following structure. Besides our good and old `package.xml`, everything else might be unfamiliar. We will go through those in detail.

```
package_with_interfaces/
|-- CMakeLists.txt
|-- action
|   `-- MoveStraightIn2D.action
|-- msg
|   |-- AmazingQuote.msg
|   `-- AmazingQuoteStamped.msg
|-- package.xml
`-- srv
    `-- AddPoints.srv
```

## 19.2 Creating the package

There isn't a template for message-only packages using **`ros2 pkg create`**. We'll need to build on top of a mostly empty **`ament_cmake`** package.

To take this chance to also learn how to nest messages on other interfaces, we also add the dependency on `geometry_msgs`.

```
cd ~/ros2_tutorial_workspace/src
ros2 pkg create package_with_interfaces \
--build-type ament_cmake \
--dependencies geometry_msgs
```

which again shows our beloved wall of text, with a few highlighted differences because of it being a **`ament_cmake`** package.

```
going to create a new package
package name: package_with_interfaces
destination directory: /root/ros2_tutorial_workspace/src
package format: 3
version: 0.0.0
description: TODO: Package description
maintainer: ['root <murilo.marinho@manchester.ac.uk>']
licenses: ['TODO: License declaration']
build type: ament_cmake
dependencies: ['geometry_msgs']
creating folder ./package_with_interfaces
creating ./package_with_interfaces/package.xml
creating source and include folder
creating folder ./package_with_interfaces/src
creating folder ./package_with_interfaces/include/package_with_interfaces
creating ./package_with_interfaces/CMakeLists.txt

[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
BSL-1.0
BSD-2.0
BSD-2-Clause
BSD-3-Clause
GPL-3.0-only
LGPL-3.0-only
MIT
MIT-0
```

The `package.xml` works the same way as when using `ament_python`. However, we no longer have a `setup.py` or `setup.cfg`, everything is handled by the `CMakeLists.txt`.

## 19.3 The `package.xml` dependencies

Whenever the package has any type of interface, the `package.xml` **must** include three specific dependencies. Namely, the ones highlighted below. Edit the `package_with_interfaces/package.xml` like so

`~/ros2_tutorial_workspace/src/package_with_interfaces/package.xml`

```
1 <?xml version="1.0"?>
2 <?xml-model href="http://download.ros.org/schema/package_format3.xsd" schematypens=
↪"http://www.w3.org/2001/XMLSchema"?>
```

(continues on next page)

(continued from previous page)

```
<package format="3">
  <name>package_with_interfaces</name>
  <version>0.0.0</version>
  <description>TODO: Package description</description>
  <maintainer email="murilomarinho@ieee.org">murilo</maintainer>
  <license>TODO: License declaration</license>

  <buildtool_depend>ament_cmake</buildtool_depend>

  <depend>geometry_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>
```

## 19.4 The message folder

The convention is to add all messages to a folder called `msg`. Let's follow that convention

```
cd ~/ros2_tutorial_workspace/src/package_with_interfaces
mkdir msg
```

## 19.5 The message file

> **Note**
>
> Here is a list of available built-in types for ROS2 interfaces.

Let us create a message file to transfer inspirational quotes between Nodes. For example, the one below.

> Use the force, Pikachu!
>
> —Uncle Ben

There are many ways to represent this, but for the sake of the example let us give each message an `id` and two rather obvious fields. Create a file called `AmazingQuote.msg` in the folder `msg` that we just created with the following contents.

`~/ros2_tutorial_workspace/src/package_with_interfaces/msg/AmazingQuote.msg`

```
# AmazingQuote.msg from https://ros2-tutorial.readthedocs.io
# An inspirational quote a day keeps the therapist away
int32 id
```

(continues on next page)

(continued from previous page)

```
string quote
string philosopher_name
```

### 19.5.1 Re-using a message from the same package

> **Note**
>
> Only messages can be used to define other interfaces. For instance, they can be used to define other messages, services, and actions.

With the `AmazingQuote.msg`, we have seen how to use built-in types. Let's use another message, `AmazingQuoteStamped.msg`, to learn two more possibilities, namely using messages from the same package and messages defined elsewhere.

`~/ros2_tutorial_workspace/src/package_with_interfaces/msg/AmazingQuoteStamped.msg`

```
# AmazingQuoteStamped.msg from https://ros2-tutorial.readthedocs.io
# An AmazingQuote.msg with a header
std_msgs/Header header
AmazingQuote quote
```

Note that if the message is defined in the same package, the package name does not appear in the message (or service) definition. If the message is defined elsewhere, we have to fully specify the package.

In many **ROS2** packages, messages with the suffix `Stamped` exist. As a rule, those are the same messages but with a additional `std_msgs/Header` so that they can be timestamped.

## 19.6 The service folder

The convention is to add all services to a folder called `srv`. Let's follow that convention

```
cd ~/ros2_tutorial_workspace/src/package_with_interfaces
mkdir srv
```

## 19.7 The service file

Add the file `AddPoints.srv` in the `srv` folder with the following contents

`~/ros2_tutorial_workspace/src/package_with_interfaces/srv/AddPoints.srv`

```
# AddPoints.srv from https://ros2-tutorial.readthedocs.io
# Adds the values of points `a` and `b` to give the output `result`
geometry_msgs/Point a
geometry_msgs/Point b
---
geometry_msgs/Point result
```

## 19.8 The action folder

The convention is to add all actions to a folder called `action`. Let's follow that convention.

```
cd ~/ros2_tutorial_workspace/src/package_with_interfaces
mkdir action
```

## 19.9 The action file

Add the file `MoveStraightIn2D.action` in the `action` folder with the following contents

`~/ros2_tutorial_workspace/src/package_with_interfaces/action/MoveStraightIn2D.action`

```
# MoveStraightIn2D.action from https://ros2-tutorial.readthedocs.io
# Attempts to move from initial position to `desired_position`.
# Returns `final_position` achieved.
# Feedback is the Euclidean distance between `initial_position` and the current position.
geometry_msgs/Point desired_position
---
geometry_msgs/Point final_position
---
float32 distance
```

## 19.10 The `CMakeLists.txt` directives

> **Note**
>
> The order of the **`CMake`** directives is very important and getting the order wrong can result in bugs with cryptic error messages.

If a package is dedicated to interfaces, there is no need to worry too much about the **`CMake`** details. We can follow the boilerplate as shown below. Edit the `package_with_interfaces/CMakeLists.txt` like so

`~/ros2_tutorial_workspace/src/package_with_interfaces/CMakeLists.txt`

```
cmake_minimum_required(VERSION 3.8)
project(package_with_interfaces)

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

# find dependencies
find_package(ament_cmake REQUIRED)
find_package(geometry_msgs REQUIRED)
# uncomment the following section in order to fill in
# further dependencies manually.
# find_package(<dependency> REQUIRED)
find_package(rosidl_default_generators REQUIRED)

#### ROS2 Interface Directives ####
```

(continues on next page)

(continued from previous page)

```
set(interface_files
  # Messages
  "msg/AmazingQuote.msg"
  "msg/AmazingQuoteStamped.msg"

  # Services
  "srv/AddPoints.srv"

  # Actions
  "action/MoveStraightIn2D.action"

)

rosidl_generate_interfaces(${PROJECT_NAME}
  ${interface_files}
  DEPENDENCIES
  geometry_msgs

)

ament_export_dependencies(
  rosidl_default_runtime
)
#### ROS2 Interface Directives [END] ####

if(BUILD_TESTING)
  find_package(ament_lint_auto REQUIRED)
  # the following line skips the linter which checks for copyrights
  # comment the line when a copyright and license is added to all source files
  set(ament_cmake_copyright_FOUND TRUE)
  # the following line skips cpplint (only works in a git repo)
  # comment the line when this package is in a git repo and when
  # a copyright and license is added to all source files
  set(ament_cmake_cpplint_FOUND TRUE)
  ament_lint_auto_find_test_dependencies()
endif()

ament_package()
```

## 19.11 What to do when adding new interfaces?

**TL;DR Adding new interfaces**

1. Add new dependencies to `package.xml`
2. Add each new interface file to `set(interface_files ...)`
3. Add new dependencies to `rosidl_generate_interfaces(... DEPENDENCIES ...)`

Yes, you **MUST** add the same dependency in two places!

If additional interfaces are required

1. Modify the `package.xml` to have any additional dependencies. See *Handling dependencies (package.xml)* for more details.
2. Add each new interface file to `set(interface_files ...)`

```
set(interface_files
  # Messages
  "msg/AmazingQuote.msg"
  "msg/AmazingQuoteStamped.msg"

  # Services
  "srv/AddPoints.srv"

  # Actions
  "action/MoveStraightIn2D.action"

)
```

**Note**

There are ways to use **CMake** directives to automatically add all files in a given folder and provide other conveniences. In hindsight, that might seem to reduce our burden. However, the method described herein is the one used in the official ROS2 packages (e.g. geometry_msgs), so let us trust that they have good reasons for it.

## 19.12 Build and source

Before we proceed, let us build and source once.

```
cd ~/ros2_tutorial_workspace
colcon build
source install/setup.bash
```

**Note**

For additional explanation and troubleshooting tips, see *Always source after you build*.

**Warning**

**colcon** will *not* work properly if your terminal has an active **venv**.

## 19.13 Getting info on custom interfaces

As long as the package has been correctly built and sourced, we can easily get information on its interfaces using **ros2 interface**.

For instance, running

```
ros2 interface package package_with_interfaces
```

returns

```
package_with_interfaces/msg/AmazingQuote
package_with_interfaces/msg/AmazingQuoteStamped
package_with_interfaces/action/MoveStraightIn2D
package_with_interfaces/srv/AddPoints
```

and we can further get more specific info on `AmazingQuote` (or `AmazingQuoteStamped`)

```
ros2 interface show package_with_interfaces/msg/AmazingQuote
```

which returns

```
# AmazingQuote.msg from https://ros2-tutorial.readthedocs.io
# An inspirational quote a day keeps the therapist away
int32 id
string quote
string philosopher_name
```

alternatively, we can do the same for `AddPoints`

```
ros2 interface show package_with_interfaces/srv/AddPoints
```

which returns expanded information on each field of the service

```
# AddPoints.srv from https://ros2-tutorial.readthedocs.io
# Adds the values of points `a` and `b` to give the output `result`
geometry_msgs/Point a
        float64 x
        float64 y
        float64 z
geometry_msgs/Point b
        float64 x
        float64 y
        float64 z
---
geometry_msgs/Point result
        float64 x
        float64 y
        float64 z
```

Lastly, we can do the same for `MoveStraightIn2D`

```
ros2 interface show package_with_interfaces/action/MoveStraightIn2D
```

which returns expanded information about all fields of the action

```
# MoveStraightIn2D.action from https://ros2-tutorial.readthedocs.io
# Attempts to move from initial position to `desired_position`.
# Returns `final_position` achieved.
# Feedback is the norm of the error between `initial_position` and the current position.
geometry_msgs/Point desired_position
        float64 x
        float64 y
        float64 z
---
```

(continues on next page)

(continued from previous page)

```
geometry_msgs/Point final_position
        float64 x
        float64 y
        float64 z
---
float32 error_norm
```

CHAPTER

TWENTY

# PUBLISHERS AND SUBSCRIBERS: USING MESSAGES

Finally, we reached the point where **ROS2** becomes appealing. As you saw in the last section, we can easily create complex interface types using an easy and generic description. We can use those to provide interprocess communication, i.e. two different programs talking to each other, which otherwise can be error-prone and very difficult to implement.

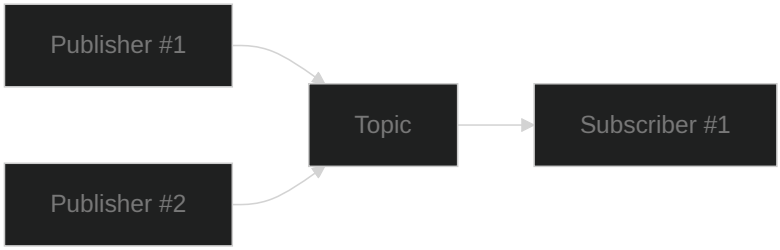
**ROS2** messages work on a model in which any number of processes can communicate over a `Topic` that only accepts one message type. Each topic is uniquely identified by a string.

Then

- A program that sends (publishes) information to the topic has one or more `Publisher` (s).
- A program that reads (subscribes) information from a topic has one or more `Subscriber` (s).

Each Node can have any number of `Publishers` and `Subscribers` and a combination thereof, connecting to an arbitrary number of Nodes. This forms part of the connections in the so-called ROS graph. An example is shown below.

## 20.1 Diagram



> **Note**
>
> This is an abstraction. As long as the information flows in this manner, it does not mean that an entity called `topic` must exist. In **ROS2**, this type of communication happens, in fact, peer-to-peer.

## 20.2 Package structure

This will be the structure of the package. The main elements are highlighted.

```
python_package_that_uses_the_messages/
|-- package.xml
|-- python_package_that_uses_the_messages
|   |-- __init__.py
|   |-- amazing_quote_publisher_node.py
|   `-- amazing_quote_subscriber_node.py
|-- resource
|   `-- python_package_that_uses_the_messages
|-- setup.cfg
|-- setup.py
`-- test
    |-- test_copyright.py
    |-- test_flake8.py
    `-- test_pep257.py
```

## 20.3 Create the package

First, let us create an **`ament_python`** package that depends on our newly developed `packages_with_interfaces` and build from there.

```
cd ~/ros2_tutorial_workspace/src
ros2 pkg create python_package_that_uses_the_messages \
--build-type ament_python \
--dependencies rclpy package_with_interfaces
```

### ros2 pkg create output

```
going to create a new package
package name: python_package_that_uses_the_messages
destination directory: /root/ros2_tutorial_workspace/src
package format: 3
version: 0.0.0
description: TODO: Package description
maintainer: ['root <murilo.marinho@manchester.ac.uk>']
licenses: ['TODO: License declaration']
build type: ament_python
dependencies: ['rclpy', 'package_with_interfaces']
creating folder ./python_package_that_uses_the_messages
creating ./python_package_that_uses_the_messages/package.xml
creating source folder
creating folder ./python_package_that_uses_the_messages/python_package_that_uses_the_
```

(continues on next page)

(continued from previous page)

```
↪messages
creating ./python_package_that_uses_the_messages/setup.py
creating ./python_package_that_uses_the_messages/setup.cfg
creating folder ./python_package_that_uses_the_messages/resource
creating ./python_package_that_uses_the_messages/resource/python_package_that_uses_the_
↪messages
creating ./python_package_that_uses_the_messages/python_package_that_uses_the_messages/__
↪init__.py
creating folder ./python_package_that_uses_the_messages/test
creating ./python_package_that_uses_the_messages/test/test_copyright.py
creating ./python_package_that_uses_the_messages/test/test_flake8.py
creating ./python_package_that_uses_the_messages/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
BSL-1.0
BSD-2.0
BSD-2-Clause
BSD-3-Clause
GPL-3.0-only
LGPL-3.0-only
MIT
MIT-0
```

## 20.4 Overview

**Note**

By no coincidence, I am using the terminology Node *with* a publisher, and Node *with* a subscriber. In general, each Node will have a combination of publishers, subscribers, and other interfaces.

Before we start exploring the elements of the package, let us

1. Create the Node with a publisher.
2. Create the Node with a subscriber.
3. Update the `setup.py` so that **`ros2 run`** finds these programs.

## 20.5 Create the Node with a publisher

**TL;DR Creating a publisher**

1. Add new dependencies to `package.xml`
2. Import new messages `from <package_name>.msg import <msg_name>`
3. In a subclass of `Node`

1. Create a publisher with `self.publisher = self.create_publisher(...)`
2. Send messages with `self.publisher.publish(....)`

4. Add the new Node to `setup.py`

For the publisher, create a file called `amazing_quote_publisher_node.py`, with the following contents

`~/ros2_tutorial_workspace/src/python_package_that_uses_the_messages/python_package_that_uses_the_messages/amazing_quote_publisher_node.py`

```python
import rclpy
from rclpy.node import Node
from package_with_interfaces.msg import AmazingQuote


class AmazingQuotePublisherNode(Node):
    """A ROS2 Node that publishes an amazing quote."""

    def __init__(self):
        super().__init__('amazing_quote_publisher_node')

        self.amazing_quote_publisher = self.create_publisher(
            msg_type=AmazingQuote,
            topic='/amazing_quote',
            qos_profile=1)

        timer_period: float = 0.5
        self.timer = self.create_timer(timer_period, self.timer_callback)

        self.incremental_id: int = 0

    def timer_callback(self):
        """Method that is periodically called by the timer."""

        amazing_quote = AmazingQuote()
        amazing_quote.id = self.incremental_id
        amazing_quote.quote = 'Use the force, Pikachu!'
        amazing_quote.philosopher_name = 'Uncle Ben'

        self.amazing_quote_publisher.publish(amazing_quote)

        self.incremental_id = self.incremental_id + 1


def main(args=None):
    """
    The main function.
    :param args: Not used directly by the user, but used by ROS2 to configure
    certain aspects of the Node.
    """
    try:
        rclpy.init(args=args)

```

(continues on next page)

(continued from previous page)

```
        amazing_quote_publisher_node = AmazingQuotePublisherNode()

        rclpy.spin(amazing_quote_publisher_node)
    except KeyboardInterrupt:
        pass
    except Exception as e:
        print(e)


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

When we built our `package_with_interfaces` in the last section, what ROS2 did for us, among other things, was create a Python library called `package_with_interfaces.msg` containing the Python implementation of the `AmazingQuote.msg`. Because of that, we can use it by importing it like so

```
import rclpy
from rclpy.node import Node
from package_with_interfaces.msg import AmazingQuote
```

The publisher must be created with the `Node.create_publisher(...)` method, which has the three parameters defined in the publisher and subscriber parameter table.

```
        self.amazing_quote_publisher = self.create_publisher(
            msg_type=AmazingQuote,
            topic='/amazing_quote',
            qos_profile=1)
```

| | |
|---|---|
| `msg_ty` | A class, namely the message that will be used in the topic. In this case, `AmazingQuote`. |
| `topic` | The topic through which the communication will occur. Can be arbitrarily chosen, but to make sense `/amazing_quote`. |
| `qos_pr` | The simplest interpretation for this parameter is the maximum number of messages that will be stored in a buffer if your node (including `spin(...)`) takes too long to process them. (See more on docs for QoSProfile.) |

**Warning**

All the arguments in publisher and subscriber parameter table should be *EXACTLY* the same in the `Publishers` and `Subscribers` of the same topic.

Then, each message is handled much like any other class in Python. We instantiate and initialize the message as follows

```
        amazing_quote = AmazingQuote()
        amazing_quote.id = self.incremental_id
        amazing_quote.quote = 'Use the force, Pikachu!'
        amazing_quote.philosopher_name = 'Uncle Ben'
```

Lastly, the message needs to be published using `Node.publish(msg)`.

```
        self.amazing_quote_publisher.publish(amazing_quote)
```

**Note**

In general, the message will **NOT** be published instantaneously after `Node.publish()` is called. It is usually fast, but entirely dependent on `rclpy.spin()` and how much work it is doing.

## 20.6 Create the Node with a subscriber

**TL;DR Creating a subscriber**

1. Add new dependencies to `package.xml`
2. Import new messages `from <package_name>.msg import <msg_name>`
3. In a subclass of `Node`
    1. Create a callback `def callback(self, msg):`
    2. Create a subscriber `self.subscriber = self.create_subscription(...)`
4. Add the new Node to `setup.py`

For the subscriber Node, create a file in `python_package_that_uses_the_messages/python_package_that_uses_the_messages` called `amazing_quote_subscriber_node.py`, with the following contents

`~/ros2_tutorial_workspace/src/python_package_that_uses_the_messages/python_package_that_uses_the_messages/amazing_quote_subscriber_node.py`

```
import rclpy
from rclpy.node import Node
from package_with_interfaces.msg import AmazingQuote


class AmazingQuoteSubscriberNode(Node):
    """A ROS2 Node that receives and AmazingQuote and prints out its info."""

    def __init__(self):
        super().__init__('amazing_quote_subscriber_node')
        self.amazing_quote_subscriber = self.create_subscription(
            msg_type=AmazingQuote,
            topic='/amazing_quote',
            callback=self.amazing_quote_subscriber_callback,
            qos_profile=1)

    def amazing_quote_subscriber_callback(self, msg: AmazingQuote):
        """Method that is called when a new msg is received by the node."""

        self.get_logger().info(f"""
        I have received the most amazing of quotes.
        It says
```

(continues on next page)

(continued from previous page)

```
                '{msg.quote}'

        And was thought by the following genius

            -- {msg.philosopher_name}

        This latest quote had the id={msg.id}.
        """)


def main(args=None):
    """
    The main function.
    :param args: Not used directly by the user, but used by ROS2 to configure
    certain aspects of the Node.
    """
    try:
        rclpy.init(args=args)

        amazing_quote_subscriber_node = AmazingQuoteSubscriberNode()

        rclpy.spin(amazing_quote_subscriber_node)
    except KeyboardInterrupt:
        pass
    except Exception as e:
        print(e)


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

Similarly to the publisher, in the subscriber, we start by importing the message in question

```
import rclpy
from rclpy.node import Node
from package_with_interfaces.msg import AmazingQuote
```

Then, in our subclass of `Node`, we call `Node.create_subscription(...)` as follows

```
        self.amazing_quote_subscriber = self.create_subscription(
            msg_type=AmazingQuote,
            topic='/amazing_quote',
            callback=self.amazing_quote_subscriber_callback,
            qos_profile=1)
```

where the only difference with respect to the publisher is the third argument, namely `callback`, in which a method that receives a `msg_type` and returns nothing is expected. For example, the `amazing_quote_subscriber_callback`.

```
    def amazing_quote_subscriber_callback(self, msg: AmazingQuote):
        """Method that is called when a new msg is received by the node."""

        self.get_logger().info(f"""
```

(continues on next page)

(continued from previous page)

```
        I have received the most amazing of quotes.
        It says

                '{msg.quote}'

        And was thought by the following genius

            -- {msg.philosopher_name}

        This latest quote had the id={msg.id}.
```

That callback method will be automatically called by ROS2, as one of the tasks performed by `rclpy.spin(Node)`. Depending on the `qos_profile`, it will not necessarily be the latest message.

> **Note**
>
> The message will **ALWAYS** take some time between being published and being received by the subscriber. The speed in which that will happen will depend not only on this Node's `rclpy.spin()`, but also on the `rclpy.spin()` of the publisher node and the communication channel.

## 20.7 Update the `setup.py`

As we already learned in *Making ros2 run work*, we must adjust the `setup.py` to refer to the Nodes we just created.

`~/ros2_tutorial_workspace/src/python_package_that_uses_the_messages/setup.py`

```
from setuptools import find_packages, setup

package_name = 'python_package_that_uses_the_messages'

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']),
    ],
    install_requires=['setuptools'],
    zip_safe=True,
    maintainer='root',
    maintainer_email='murilo.marinho@manchester.ac.uk',
    description='TODO: Package description',
    license='TODO: License declaration',
    tests_require=['pytest'],
    entry_points={
        'console_scripts': [
            'amazing_quote_publisher_node = python_package_that_uses_the_messages.
↪amazing_quote_publisher_node:main',
            'amazing_quote_subscriber_node = python_package_that_uses_the_messages.
```

(continues on next page)

(continued from previous page)

```
↪amazing_quote_subscriber_node:main'
        ],
    },
)
```

## 20.8 Build and source

Before we proceed, let us build and source once.

```
cd ~/ros2_tutorial_workspace
colcon build
source install/setup.bash
```

> **Note**
>
> For additional explanation and troubleshooting tips, see *Always source after you build*.

> **Warning**
>
> **colcon** will *not* work properly if your terminal has an active **venv**.

## 20.9 Testing Publisher and Subscriber

Whenever we need to open two or more terminal windows, remember that *Terminator is life*.

Let us open two terminals.

In the first terminal, we run

```
ros2 run python_package_that_uses_the_messages amazing_quote_publisher_node
```

Nothing, in particular, should happen now. The publisher is sending messages through the specific topic we defined, but we need at least one subscriber to interact with those messages.

Hence, in the second terminal, we run

```
ros2 run python_package_that_uses_the_messages amazing_quote_subscriber_node
```

which outputs

```
[INFO] [1753664072.638312553] [amazing_quote_subscriber_node]:
        I have received the most amazing of quotes.
        It says

                'Use the force, Pikachu!'

        And was thought by the following genius

            -- Uncle Ben
```

(continues on next page)

(continued from previous page)

```
        This latest quote had the id=37.

[INFO] [1753664073.121886428] [amazing_quote_subscriber_node]:
        I have received the most amazing of quotes.
        It says

               'Use the force, Pikachu!'

        And was thought by the following genius

            -- Uncle Ben

        This latest quote had the id=38.
```

**Note**

If there are any issues with either the publisher or the subscriber, this connection will not work. In the next section, we'll see strategies to help us troubleshoot and understand communication through topics.

**Warning**

Unless instructed otherwise, the publisher does **NOT** wait for a subscriber to connect before it starts publishing the messages. As shown in the case above, the first message we received started with *id>0*. If we delayed longer to start the publisher, we would have received later messages only.

Let's close each node with CTRL+C on each terminal before we proceed to the next tutorial.

CHAPTER

# TWENTYONE

# INSPECTING TOPICS (ROS2 TOPIC)

ROS2 has a tool to help us inspect topics. This is used with considerable frequency in practice to troubleshoot and speed up the development of publishers and subscribers. As usual, we can get more information on this tool as follows.

```
ros2 topic -h
```

which outputs the detailed information of the tool, as shown below. In particular, the highlighted fields are used quite frequently in practice.

```
usage: ros2 topic [-h]
                  [--include-hidden-topics]
                  Call `ros2 topic <command>
                  -h` for more detailed usage.
                  ...

Various topic related sub-commands

options:
  -h, --help            show this help message
                        and exit
  --include-hidden-topics
                        Consider hidden topics
                        as well

Commands:
  bw     Display bandwidth used by topic
  delay  Display delay of topic from timestamp in header
  echo   Output messages from a topic
  find   Output a list of available topics of a given type
  hz     Print the average publishing rate to screen
  info   Print information about a topic
  list   Output a list of available topics
  pub    Publish a message to a topic
  type   Print a topic's type

  Call `ros2 topic <command> -h` for more detailed usage.
```

## 21.1 Start a publisher

During the development of a publisher, it is extremely useful to be able to check if topics are being properly made before we venture into making the subscribers. To see some of the tools for this job, we start by running the publisher Node we wrote in the last section.

> **Warning**
>
> Be sure to terminate the Nodes we used in the past section before proceeding (e.g. with CTRL+C), otherwise, the output will look different from what is described here.

```
ros2 run python_package_that_uses_the_messages amazing_quote_publisher_node
```

## 21.2 Getting all topics with `ros2 topic list`

In particular, when there are many topics, it is difficult to remember every name. To see all currently active topics, we can run

```
ros2 topic list
```

which, in this case, outputs

```
/amazing_quote
/parameter_events
/rosout
```

showing, in particular, the `/amazing_quote` topic what we were looking for.

> **Hint**
>
> The **`ros2 topic info`** is one of the main tools to find out typos in the names of topics. For example, if there was a typo in our topic we might find, in fact, two topics being listed, when we only expected one. For instance,
>
> ```
> /amazing_quote
> /amazing_quotes
> /parameter_events
> /rosout
> ```

## 21.3 `grep` is your new best friend

> **Note**
>
> If you want more information on **`grep`**, check the Ubuntu Manpage

When the list of topics is too large, we can use **`grep`** to help filter the output. E.g.

```
ros2 topic list | grep quote
```

which outputs only the lines that contain `quote`, that is

```
/amazing_quote
```

## 21.4 Getting quick info with `ros2 topic info`

To get some quick information on a topic, we can run

```
ros2 topic info /amazing_quote
```

which outputs the message type and the number of publishers and subscribers connected to that topic

```
Type: package_with_interfaces/msg/AmazingQuote
Publisher count: 1
Subscription count: 0
```

## 21.5 Checking topic contents with `ros2 topic echo`

The **`ros2 topic echo`** is the main tool that we can use to inspect topic activity. We can check all the options of **`ros2 topic echo`** with the command below. The output is quite long so it's not replicated here.

```
ros2 topic echo -h
```

To inspect the topic whose name we already know, we run

```
ros2 topic echo /amazing_quote
```

which outputs the following

```
id: 6
quote: Use the force, Pikachu!
philosopher_name: Uncle Ben
---
id: 7
quote: Use the force, Pikachu!
philosopher_name: Uncle Ben
---
id: 8
quote: Use the force, Pikachu!
philosopher_name: Uncle Ben
---
id: 9
quote: Use the force, Pikachu!
philosopher_name: Uncle Ben
---
id: 10
quote: Use the force, Pikachu!
philosopher_name: Uncle Ben
---
id: 11
quote: Use the force, Pikachu!
philosopher_name: Uncle Ben
---
```

## 21.6 `grep` is still your best friend

Whenever the topic is too crowded or the messages too fast, it might be difficult to pinpoint a single field we are looking for. In that case, **`grep`** can also help.

For example let us say that we want to see only the `id` fields of the messages. We can do

```
ros2 topic echo /amazing_quote | grep id
```

which will output only the lines with that pattern, e.g.

```
id: 1550
id: 1551
id: 1552
id: 1553
```

## 21.7 Measuring publishing frequency with `ros2 topic hz`

There are situations in which we are interested in knowing if the topics are receiving messages at an expected rate, without particular interest in the contents of the messages. We can do so with

```
ros2 topic hz /amazing_quote
```

which will output, after some time,

```
   WARNING: topic [/amazing_quote] does not appear to be published yet
average rate: 2.000
    min: 0.500s max: 0.500s std dev: 0.00007s window: 4
average rate: 2.000
    min: 0.500s max: 0.500s std dev: 0.00013s window: 7
average rate: 2.000
    min: 0.500s max: 0.500s std dev: 0.00011s window: 9
```

We must wait for a while until messages are received so that the tool can measure the frequency properly. You probably have noticed that the frequency measured by **`ros2 topic hz`** is compatible with the period of the `Timer` in our publisher Node.

## 21.8 Stop the publisher

Now we have exhausted all relevant tools that can give us information related to the publisher. Let us close the publisher with `CTRL+C` so that we can evaluate how these tools can help us analyze a subscriber.

## 21.9 Start the subscriber and get basic info

```
ros2 run python_package_that_uses_the_messages amazing_quote_subscriber_node
```

When only the subscriber is running, we can still get the basic info on the topic, e.g.

```
ros2 topic list
```

which also outputs

```
/amazing_quote
/parameter_events
/rosout
```

and

```
ros2 topic info /amazing_quote
```

which, differently from before, outputs

```
Type: package_with_interfaces/msg/AmazingQuote
Publisher count: 0
Subscription count: 1
```

## 21.10 Testing your subscribers with `ros2 topic pub`

To somewhat quickly evaluate a subscriber, we can use the **`ros2 topic pub`**. It allows us to publish messages to check the behavior of our subscribers.

In our case, we can send an **`AmazingQuote`** using YAML (YAML Ain't Markup Language) (More info). You can also refer to the YAML Cheat Sheet at QuickRef.ME.

```
ros2 topic pub /amazing_quote \
package_with_interfaces/msg/AmazingQuote \
'{
id: 1994,
quote: So you're telling me there's a chance,
philosopher_name: Lloyd
}'
```

> **Note**
>
> To improve readability, the command above uses the escape character \. You can see more on this at the bash docs. You can also refer to the **`bash`** Cheat Sheet at QuickRef.ME.

which will result in our subscriber outputting

```
[INFO] [1684222464.960446589] [amazing_quote_subscriber_node]:
        I have received the most amazing of quotes.
        It says

               'So you're telling me there's a chance'

        And was though by the following genius

            -- Lloyd

        This latest quote had the id=1994.

[INFO] [1684222465.953452826] [amazing_quote_subscriber_node]:
        I have received the most amazing of quotes.
```

(continues on next page)

(continued from previous page)

```
It says

       'So you're telling me there's a chance'

And was though by the following genius

    -- Lloyd

This latest quote had the id=1994.
```

For complicated messages, properly writing the message on the terminal can be a handful. In that case, it might be better to make a minimal script to test the subscriber instead. Refer to *Create the Node with a publisher*.

CHAPTER

TWENTYTWO

# PARAMETERS AND LAUNCH FILES: CREATING CONFIGURABLE NODES

The Nodes we have made in the past few sections are interesting because they take advantage of the interprocess communication provided by ROS2.

Other capabilities of ROS2 that we must take advantage of are ROS2 parameters and ROS2 launch files. We can use them to modify the behavior of Nodes without having to modify their source code.

For Python users, that might sound less appealing than for users of compiled languages. However, users of your package might not want nor be able to modify the source code directly, if the package is installable or part of a larger system with multiple users.

## 22.1 Create the package

First, let us create an **ament_python** package that depends on our `packages_with_interfaces` and build from there.

```
cd ~/ros2_tutorial_workspace/src
ros2 pkg create python_package_that_uses_parameters_and_launch_files \
--build-type ament_python \
--dependencies rclpy package_with_interfaces
```

## 22.2 Overview

Before we start exploring the elements of the package, let us

1. Create the Node with a configurable publisher using parameters, mostly as we saw in *Create the Node with a publisher*.
2. Create a launch file to configure the Node without modifying its source code.

## 22.3 Create the Node using parameters

**TL;DR Using parameters in a Node**

1. Declare the parameter with `Node.declare_parameter()`, usually in the class's `__init__`.
2. Get the parameter with `Node.get_parameter()` either once or continuously.

**In this step, we'll work on this.**

```
python_package_that_uses_parameters_and_launch_files/
|-- launch
|   `-- peanut_butter_falcon_quote_publisher_launch.py
|-- package.xml
|-- python_package_that_uses_parameters_and_launch_files
|   |-- __init__.py
|   `-- amazing_quote_configurable_publisher_node.py
|-- resource
|   `-- python_package_that_uses_parameters_and_launch_files
|-- setup.cfg
|-- setup.py
`-- test
    |-- test_copyright.py
    |-- test_flake8.py
    `-- test_pep257.py
```

For the sake of the example, let us suppose that we want to make an `AmazingQuote` publisher that is, now, configurable.

Let's start by creating an `amazing_quote_configurable_publisher_node.py` in `python_package_that_uses_parameters_and_launch_files/python_package_that_uses_parameters_and_launch_fil` with the following contents

`amazing_quote_configurable_publisher_node.py`

```
import rclpy
from rclpy.node import Node
from package_with_interfaces.msg import AmazingQuote


class AmazingQuoteConfigurablePublisherNode(Node):
    """A configurable ROS2 Node that publishes a configurable amazing quote."""

    def __init__(self):
        super().__init__('amazing_quote_configurable_publisher_node')

        # Periodically-obtained parameters
        self.declare_parameter('quote', 'Use the force, Pikachu!')
        self.declare_parameter('philosopher_name', 'Uncle Ben')

        # One-off parameters
        self.declare_parameter('topic_name', 'amazing_quote')
        topic_name: str = self.get_parameter('topic_name').get_parameter_value().string_
↪value
        self.declare_parameter('period', 0.5)
        timer_period: float = self.get_parameter('period').get_parameter_value().double_
↪value

        self.configurable_amazing_quote_publisher = self.create_publisher(
            msg_type=AmazingQuote,
            topic=topic_name,
            qos_profile=1)
```

(continues on next page)

(continued from previous page)

```
        self.timer = self.create_timer(timer_period, self.timer_callback)

        self.incremental_id: int = 0

    def timer_callback(self):
        """Method that is periodically called by the timer."""

        quote: str = self.get_parameter('quote').get_parameter_value().string_value
        philosopher_name: str = self.get_parameter('philosopher_name').get_parameter_
↪value().string_value

        amazing_quote = AmazingQuote()
        amazing_quote.id = self.incremental_id
        amazing_quote.quote = quote
        amazing_quote.philosopher_name = philosopher_name

        self.configurable_amazing_quote_publisher.publish(amazing_quote)

        self.incremental_id = self.incremental_id + 1


def main(args=None):
    """
    The main function.
    :param args: Not used directly by the user, but used by ROS2 to configure
    certain aspects of the Node.
    """
    try:
        rclpy.init(args=args)

        amazing_quote_configurable_publisher_node =␣
↪AmazingQuoteConfigurablePublisherNode()

        rclpy.spin(amazing_quote_configurable_publisher_node)
    except KeyboardInterrupt:
        pass
    except Exception as e:
        print(e)


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

## 22.4 Don't forget to declare the parameter!

> **Note**
>
> According to the official documentation, it is possible to work with undeclared parameters, but I recommend against this for basic usage.

It's easy to forget it, but `Node.get_parameter()` will not work if the parameter was not first declared with `Node.declare_parameter()`. Don't forget it!

## 22.5 One-off parameters

For one-off parameters, we just get them once after declaring them. Because we're using those attributes directly in the `__init__` method, they are not made attributes of the class, but they could be.

```
        # One-off parameters
        self.declare_parameter('topic_name', 'amazing_quote')
        topic_name: str = self.get_parameter('topic_name').get_parameter_value().string_
↪value
        self.declare_parameter('period', 0.5)
        timer_period: float = self.get_parameter('period').get_parameter_value().double_
↪value

        self.configurable_amazing_quote_publisher = self.create_publisher(
            msg_type=AmazingQuote,
            topic=topic_name,
            qos_profile=1)

        self.timer = self.create_timer(timer_period, self.timer_callback)
```

In this case, we're making the topic name and publication periodicity as one-off configurable parameters.

## 22.6 Continuously-obtained parameters

> **Note**
>
> According to the official documentation, it is possible to assign callbacks to manage changes in parameters. It is not the best-documented feature and has some caveats, so we will skip that for now.

For parameters that we obtain continuously through the lifetime of the Node, we can, for example, declare them in the `__init__` method, like so

```
        # Periodically-obtained parameters
        self.declare_parameter('quote', 'Use the force, Pikachu!')
        self.declare_parameter('philosopher_name', 'Uncle Ben')
```

then obtain them in another method, like so

```
    def timer_callback(self):
        """Method that is periodically called by the timer."""

        quote: str = self.get_parameter('quote').get_parameter_value().string_value
```

(continues on next page)

(continued from previous page)

```
        philosopher_name: str = self.get_parameter('philosopher_name').get_parameter_
↪value().string_value

        amazing_quote = AmazingQuote()
        amazing_quote.id = self.incremental_id
        amazing_quote.quote = quote
        amazing_quote.philosopher_name = philosopher_name

        self.configurable_amazing_quote_publisher.publish(amazing_quote)

        self.incremental_id = self.incremental_id + 1
```

In this example, we are making the `quote` and the `philosopher_name` as configurable parameters that can be changed continuously, during the lifetime of the Node. After they are changed, the node will publish a message with different contents.

## 22.7 Truly configurable: using `_launch.py` files

**TL;DR Using launch files**

1. (Once) Create a `launch` folder in the project.
2. Create the launch file named as `launch/<something>_launch.py`.
3. (Once) modify the `setup.py` to correctly install launch files.

Differently from ROS1, in ROS2 we can use Python launch files. They are quite powerful, well documented, and mentioned first in the official documentation, so we will use them instead of XML or YAML files.

## 22.8 (Once) create the `launch` folder

**In this step, we'll work on this.**

```
python_package_that_uses_parameters_and_launch_files/
|-- launch
|   `-- peanut_butter_falcon_quote_publisher_launch.py
|-- package.xml
|-- python_package_that_uses_parameters_and_launch_files
|   |-- __init__.py
|   `-- amazing_quote_configurable_publisher_node.py
|-- resource
|   `-- python_package_that_uses_parameters_and_launch_files
|-- setup.cfg
|-- setup.py
`-- test
    |-- test_copyright.py
    |-- test_flake8.py
    `-- test_pep257.py
```

Well, without further ado

```
cd ~/ros2_tutorial_workspace/src/python_package_that_uses_parameters_and_launch_files
mkdir launch
```

## 22.9 Create the `launch` file

**In this step, we'll work on this.**

```
python_package_that_uses_parameters_and_launch_files/
|-- launch
|   `-- peanut_butter_falcon_quote_publisher_launch.py
|-- package.xml
|-- python_package_that_uses_parameters_and_launch_files
|   |-- __init__.py
|   `-- amazing_quote_configurable_publisher_node.py
|-- resource
|   `-- python_package_that_uses_parameters_and_launch_files
|-- setup.cfg
|-- setup.py
`-- test
    |-- test_copyright.py
    |-- test_flake8.py
    `-- test_pep257.py
```

Suppose that we are tired of all the meme quotes and want to make our Node publish a truly inspirational quote. We start by making the launch file named `peanut_butter_falcon_quote_publisher_launch.py` within the `launch` folder we just created, with the following contents

`peanut_butter_falcon_quote_publisher_launch.py`

```
from launch import LaunchDescription
from launch_ros.actions import Node


def generate_launch_description():
    return LaunchDescription([
        Node(
            output='screen',
            emulate_tty=True,
            package='python_package_that_uses_parameters_and_launch_files',
            executable='amazing_quote_configurable_publisher_node',
            name='peanut_butter_falcon_quote_publisher_node',
            parameters=[{
                "topic_name": "truly_inspirational_quote",
                "period": 0.25,
                "quote": "Yeah, you're gonna die, it's a matter of time. That ain't the question. The question's, "
                         "whether they're gonna have a good story to tell about you when you're gone",
                "philosopher_name": "Tyler",
```

(continues on next page)

(continued from previous page)

```
            }]
        )
    ])
```

We're relying on the `LaunchDescription`, which expects a list of `launch_ros.actions`.

```
from launch import LaunchDescription
from launch_ros.actions import Node
```

When using a `launch_ros.actions.Node`, we need to define which `package` it belongs to and the `executable` which must match the name we set for the executable in the `setup.py`

```
            package='python_package_that_uses_parameters_and_launch_files',
            executable='amazing_quote_configurable_publisher_node',
```

Besides the parameters, we can configure the name of the Node, such that each is unique

```
            name='peanut_butter_falcon_quote_publisher_node',
```

Finally, our parameters are defined using a dictionary within a list, namely

```
                "topic_name": "truly_inspirational_quote",
                "period": 0.25,
                "quote": "Yeah, you're gonna die, it's a matter of time. That ain't the␣
↪question. The question's, "
                         "whether they're gonna have a good story to tell about you when␣
↪you're gone",
                "philosopher_name": "Tyler",
```

## 22.10 The `setup.py`

**In this step, we'll work on this.**

```
python_package_that_uses_parameters_and_launch_files/
|-- launch
|   `-- peanut_butter_falcon_quote_publisher_launch.py
|-- package.xml
|-- python_package_that_uses_parameters_and_launch_files
|   |-- __init__.py
|   `-- amazing_quote_configurable_publisher_node.py
|-- resource
|   `-- python_package_that_uses_parameters_and_launch_files
|-- setup.cfg
|-- setup.py
`-- test
    |-- test_copyright.py
    |-- test_flake8.py
    `-- test_pep257.py
```

Modify the `setup.py` to look like this

setup.py

```
import os
from glob import glob
from setuptools import setup

package_name = 'python_package_that_uses_parameters_and_launch_files'

setup(
    name=package_name,
    version='0.0.0',
    packages=[package_name],
    data_files=[
        ('share/ament_index/resource_index/packages',
         ['resource/' + package_name]),
        ('share/' + package_name, ['package.xml']),
        (os.path.join('share', package_name, 'launch'), glob(os.path.join('launch',
↪'*launch.[pxy][yma]*'))),
    ],
    install_requires=['setuptools'],
    zip_safe=True,
    maintainer='murilo',
    maintainer_email='murilomarinho@ieee.org',
    description='TODO: Package description',
    license='TODO: License declaration',
    tests_require=['pytest'],
    entry_points={
        'console_scripts': [
            'amazing_quote_configurable_publisher_node = '
            'python_package_that_uses_parameters_and_launch_files.amazing_quote_
↪configurable_publisher_node:main',
        ],
    },
)
```

We have already seen a `setup.py` so many times we're almost calling it Wilson. The only difference is emphasized above inside the `data_files`, which is the line that will specify that launch files will be installed as well. Notice that the `setup.py` looks for files with a specific pattern in the folder `launch`, so be sure that your launch files have the correct name otherwise they might not be installed as expected.

## 22.11 Build and source

Before we proceed, let us build and source once.

```
cd ~/ros2_tutorial_workspace
colcon build
source install/setup.bash
```

> **Note**
>
> For additional explanation and troubleshooting tips, see *Always source after you build*.

> **Warning**
>
> **colcon** will *not* work properly if your terminal has an active **venv**.

CHAPTER

# TWENTYTHREE

# LAUNCH CONFIGURABLE NODES (ROS2 LAUNCH)

ROS2 has a tool to interact with launch files called **ros2 launch**.

We can obtain more information on it with

```
ros2 launch -h
```

which returns

```
usage: ros2 launch [-h] [-n] [-d] [-p | -s] [-a]
                   [--launch-prefix LAUNCH_PREFIX]
                   [--launch-prefix-filter LAUNCH_PREFIX_FILTER]
                   package_name [launch_file_name] [launch_arguments ...]

Run a launch file

positional arguments:
  package_name          Name of the ROS package which contains the launch
                        file
  launch_file_name      Name of the launch file
  launch_arguments      Arguments to the launch file; '<name>:=<value>' (for
                        duplicates, last one wins)

options:
  -h, --help            show this help message and exit
  -n, --noninteractive  Run the launch system non-interactively, with no
                        terminal associated
  -d, --debug           Put the launch system in debug mode, provides more
                        verbose output.
  -p, --print, --print-description
                        Print the launch description to the console without
                        launching it.
  -s, --show-args, --show-arguments
                        Show arguments that may be given to the launch file.
  -a, --show-all-subprocesses-output
                        Show all launched subprocesses' output by overriding
                        their output configuration using the
                        OVERRIDE_LAUNCH_PROCESS_OUTPUT envvar.
  --launch-prefix LAUNCH_PREFIX
                        Prefix command, which should go before all
                        executables. Command must be wrapped in quotes if it
                        contains spaces (e.g. --launch-prefix 'xterm -e gdb
```

(continues on next page)

(continued from previous page)

```
                        -ex run --args').
  --launch-prefix-filter LAUNCH_PREFIX_FILTER
                        Regex pattern for filtering which executables the
                        --launch-prefix is applied to by matching the
                        executable name.
```

Despite the large number of possible options, there are no notable examples of options that are of particular use to us right now.

We can call our Node, configured with our launch file, with

```
ros2 launch python_package_that_uses_parameters_and_launch_files peanut_butter_falcon_
↪quote_publisher_launch.py
```

which returns

```
[INFO] [launch]: All log files can be found below /home/murilo/.ros/log/2023-06-30-17-00-
↪07-522194-murilos-toaster-2963
[INFO] [launch]: Default logging verbosity is set to INFO
[INFO] [amazing_quote_configurable_publisher_node-1]: process started with pid [2964]
```

showing that the launch was successful.

**IN ANOTHER TERMINAL** we run

```
ros2 topic echo /truly_inspirational_quote
```

resulting in something similar to

```
id: 301
quote: Yeah, you're gonna die, it's a matter of time. That ain't the question. The␣
↪question's, whether they're gonna have a good story ...
philosopher_name: Tyler
---
id: 302
quote: Yeah, you're gonna die, it's a matter of time. That ain't the question. The␣
↪question's, whether they're gonna have a good story ...
philosopher_name: Tyler
---
id: 303
quote: Yeah, you're gonna die, it's a matter of time. That ain't the question. The␣
↪question's, whether they're gonna have a good story ...
philosopher_name: Tyler
---
```

And there you have it. Feeling inspired yet?

CHAPTER

# TWENTYFOUR

# INSPECTING PARAMETERS (ROS2 PARAM)

ROS2 has a tool to interact with launch files called **`ros2 param`**.

We can obtain more information on it with

```
ros2 param -h
```

which returns

```
usage: ros2 param [-h] Call `ros2 param <command> -h` for more detailed usage. ...

Various param related sub-commands

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

Commands:
  delete    Delete parameter
  describe  Show descriptive information about declared parameters
  dump      Show all of the parameters of a node in a YAML file format
  get       Get parameter
  list      Output a list of available parameters
  load      Load parameter file for a node
  set       Set parameter

  Call `ros2 param <command> -h` for more detailed usage.
```

As shown in the emphasized lines above, the **`ros2 param`** tool has a large number of useful commands to interact with parameters.

## 24.1 Launching the Node with parameters

**Hint**

If you left the Node running from the last section, just keep it that way and skip this.

```
ros2 launch \
python_package_that_uses_parameters_and_launch_files \
peanut_butter_falcon_quote_publisher_launch.py
```

## 24.2 List-up parameters with `ros2 param list`

> **Hint**
>
> Remember that *grep is your new best friend*.

Similar to other ROS2 commands, we can get a list of currently loaded parameters with

```
ros2 param list
```

which returns a well organized list showing the parameters of each active Node

```
/peanut_butter_falcon_quote_publisher_node:
  period
  philosopher_name
  quote
  topic_name
  use_sim_time
```

## 24.3 Obtain parameters with `ros2 param get`

To obtain the value of a parameter, we can do as follows

```
ros2 param get \
/peanut_butter_falcon_quote_publisher_node \
quote
```

which will return the current value of the parameter, in this case, the initial value we set in the launch file

```
String value is: Yeah, you're gonna die, it's a matter of time. That ain't the question.␣
↪The question's, whether they're gonna have a good story to tell about you when you're␣
↪gone
```

## 24.4 Let's check the output of the Node

> **Hint**
>
> If you left **`ros2 topic echo`** running from the last section, just keep it that way and skip this.

Before the next step, as we did in the past section, we do, **IN ANOTHER TERMINAL WINDOW**

```
ros2 topic echo /truly_inspirational_quote
```

## 24.5 Assign values to parameters with `ros2 param set`

For testing and regular usage, setting parameters from the command line is extremely helpful. Similar to how we are able to publish messages to topics using a ROS2 tool, we can set a parameter with the following syntax

```
ros2 param set \
/peanut_butter_falcon_quote_publisher_node \
quote \
"You got a good-guy heart. You can't do shit about it, that's just who you are. You're a␣
↪hero."
```

If everything is correct, we'll get

```
Set parameter successful
```

**Important**

Some errors are easy to debug, such as when we get the name of the Node wrong

```
Node not found
```

but because of the interaction between the **terminal**, **ros2 param** itself, and the syntax of the services, its easy to find cryptic error messages. At first, always suppose that there's a typo somewhere.

Changing parameters is not instantaneous and, after the change becomes visible in the Node, our Node might have to loop once before it updates itself. We will be able to see that change as follows in the terminal window running **ros2 topic echo**

```
id: 2220
quote: Yeah, you're gonna die, it's a matter of time. That ain't the question. The␣
↪question's, whether they're gonna have a good story ...
philosopher_name: Tyler
---
id: 2221
quote: You got a good-guy heart. You can't do shit about it, that's just who you are. You
↪'re a hero.
philosopher_name: Tyler
---
id: 2222
quote: You got a good-guy heart. You can't do shit about it, that's just who you are. You
↪'re a hero.
philosopher_name: Tyler
---
id: 2223
quote: You got a good-guy heart. You can't do shit about it, that's just who you are. You
↪'re a hero.
philosopher_name: Tyler
---
id: 2224
quote: You got a good-guy heart. You can't do shit about it, that's just who you are. You
↪'re a hero.
philosopher_name: Tyler
```

## 24.6 Save parameters to a file with `ros2 param dump`

Words are sometimes little happy accidents. This usage of the word dump has no relation whatsoever to, for example, Peter got dumped by Sarah and went to Hawaii. Dump files are usually related to crashes and unresponsive programs, so this name puzzles me since ROS: the first.

While we wait for someone to come and correct me on my claims above, just think about this as a weird name for **`ros2 param print_to_screen_as_yaml`**. It prints the parameters in the terminal with a YAML file format. It is nice because it gives a bit more info than **`ros2 param list`**, but not so useful as-is. The trick is that we can put all that nicely formatted content into a file with

```
cd ~/ros2_tutorial_workspace/src
ros2 param dump \
/peanut_butter_falcon_quote_publisher_node \
> peanut_butter_falcon_quote_publisher_node.yaml
```

where we are using the > (see *bash redirections*) to overwrite the contents of the `peanut_butter_falcon_quote_publisher_node.yaml` file with the output of **`ros2 param dump`**, so be careful not to overwrite your precious files by mistake.

We can inspect the contents of the file with

```
cat peanut_butter_falcon_quote_publisher_node.yaml
```

which outputs

```
/peanut_butter_falcon_quote_publisher_node:
  ros__parameters:
    period: 0.25
    philosopher_name: Tyler
    quote: Yeah, you're gonna die, it's a matter of time. That ain't the question.
      The question's, whether they're gonna have a good story to tell about you when
      you're gone
    topic_name: truly_inspirational_quote
    use_sim_time: false
```

## 24.7 Load parameters from a file with `ros2 param load`

> **Warning**
>
> To proceed, end the `peanut_butter_falcon_quote_publisher_node` Node with CTRL+C.

As in the prior step, suppose that we have a file `peanut_butter_falcon_quote_publisher_node.yaml` with the parameters we love the most. What we can do with **`ros2 param load`** is load that file. Nicely predictable and understandable naming convention.

We can start the Node with the launch file

```
ros2 launch python_package_that_uses_parameters_and_launch_files \
peanut_butter_falcon_quote_publisher_launch.py
```

which, at the beginning, will have the parameters set in the `_launch.py`. We can then

```
cd ~/ros2_tutorial_workspace/src
ros2 param load \
/peanut_butter_falcon_quote_publisher_node \
peanut_butter_falcon_quote_publisher_node.yaml
```

which will return

```
Set parameter period successful
Set parameter philosopher_name successful
Set parameter quote successful
Set parameter topic_name successful
Set parameter use_sim_time successful
```

indicating that all parameters defined in the YAML were successfully loaded.

CHAPTER

TWENTYFIVE

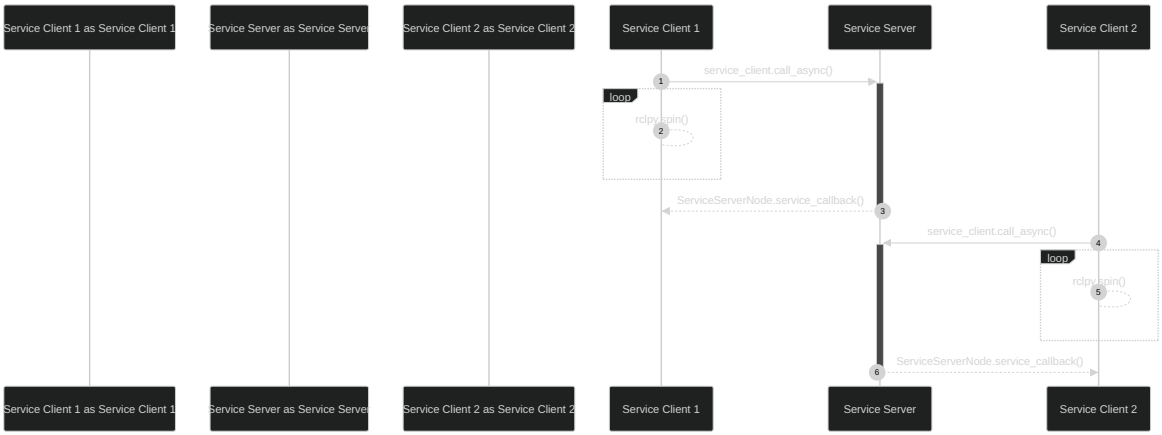
# AT YOUR SERVICE: SERVERS AND CLIENTS

Changed in version Jazzy: The contents of this session were expanded and the example simplified. The previous version is available at the Humble tutorials.

In some cases, we need means of communication in which each command has an associated response. That is where services come into play. We provide services by creating a `ServiceServer`. The service server will provide a service that can be accessed by one or more `ServiceClient`s.

In this sense, a service is much less of an abstract entity than a topic. Each service should only have a single service server that will receive a `Request` and provide a `Response`.

## 25.1 Diagram


Service Client 1 as Service Client 1
Service Server as Service Server
Service Client 2 as Service Client 2
Service Client 1
Service Server
Service Client 2
service_client.call_async()
loop
rclpy.spin()
ServiceServerNode.service_callback()
service_client.call_async()
loop
rclpy.spin()
ServiceServerNode.service_callback()
Service Client 1 as Service Client 1
Service Server as Service Server
Service Client 2 as Service Client 2
Service Client 1
Service Server
Service Client 2

## 25.2 Create the package

We start by creating a package to use the `Service` we first created in *The service file*.

```
cd ~/ros2_tutorial_workspace/src
ros2 pkg create python_package_that_uses_the_services \
--build-type ament_python \
--dependencies rclpy package_with_interfaces
```

### ros2 pkg create output

```
going to create a new package
package name: python_package_that_uses_the_services
destination directory: /root/ros2_tutorial_workspace/src
package format: 3
version: 0.0.0
description: TODO: Package description
maintainer: ['root <murilo.marinho@manchester.ac.uk>']
licenses: ['TODO: License declaration']
build type: ament_python
dependencies: ['rclpy', 'package_with_interfaces']
creating folder ./python_package_that_uses_the_services
creating ./python_package_that_uses_the_services/package.xml
creating source folder
creating folder ./python_package_that_uses_the_services/python_package_that_uses_the_
↪services
creating ./python_package_that_uses_the_services/setup.py
creating ./python_package_that_uses_the_services/setup.cfg
creating folder ./python_package_that_uses_the_services/resource
creating ./python_package_that_uses_the_services/resource/python_package_that_uses_the_
↪services
creating ./python_package_that_uses_the_services/python_package_that_uses_the_services/__
↪init__.py
creating folder ./python_package_that_uses_the_services/test
creating ./python_package_that_uses_the_services/test/test_copyright.py
creating ./python_package_that_uses_the_services/test/test_flake8.py
creating ./python_package_that_uses_the_services/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
BSL-1.0
BSD-2.0
BSD-2-Clause
BSD-3-Clause
GPL-3.0-only
LGPL-3.0-only
MIT
MIT-0
```

## 25.3 Overview

**File structure**

This will be the file structure for the `Service` tutorial. Highlighted are the main files for the `ServiceServer` and `ServiceClient`.

```
python_package_that_uses_the_services
|-- package.xml
|-- python_package_that_uses_the_services
|   |-- __init__.py
|   |-- add_points_service_client_introspection_node.py
|   |-- add_points_service_client_node.py
|   |-- add_points_service_server_introspection_node.py
|   `-- add_points_service_server_node.py
|-- resource
|   `-- python_package_that_uses_the_services
|-- setup.cfg
|-- setup.py
`-- test
    |-- test_copyright.py
    |-- test_flake8.py
    `-- test_pep257.py
```

Before we start exploring the elements of the package, let us

1. Create the Node with a Service Server.
2. Create the Node with a Service Client.
3. Update the `setup.py` so that **`ros2 run`** finds these programs.

## 25.4 Create the Node with a Service Server

**TL;DR Creating a service server**

1. Add new dependencies to `package.xml`
2. Import new services `from <package_name>.srv import <srv_name>`
3. In a subclass of `Node`
    1. create a callback `def callback(self, request, response):`
    2. create a service server with `self.service_server = self.create_service(...)`
4. Add the new Node to `setup.py`

Let's start by creating a `add_points_service_server_node.py`.

`~/ros2_tutorial_workspace/src/python_package_that_uses_the_services/python_package_that_uses_the_services/add_points_service_server_node.py`

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

(continues on next page)

(continued from previous page)

```
from package_with_interfaces.srv import AddPoints


class AddPointsServiceServerNode(Node):
    """A ROS2 Node with a Service Server for AddPoints."""

    def __init__(self):
        super().__init__('add_points_service_server')

        self.service_server = self.create_service(
            srv_type=AddPoints,
            srv_name='/add_points',
            callback=self.add_points_service_callback)

        self.service_server_call_count: int = 0

    def add_points_service_callback(self,
                                    request: AddPoints.Request,
                                    response: AddPoints.Response
                                    ) -> AddPoints.Response:
        """
        Adds the two points `a` and `b` in the request and returns the `result`.
        """

        response.result.x = request.a.x + request.b.x
        response.result.y = request.a.y + request.b.y
        response.result.z = request.a.z + request.b.z

        return response


def main(args=None):
    """
    The main function.
    :param args: Not used directly by the user, but used by ROS2 to configure
    certain aspects of the Node.
    """
    try:
        rclpy.init(args=args)

        add_points_service_server_node = AddPointsServiceServerNode()

        rclpy.spin(add_points_service_server_node)
    except KeyboardInterrupt:
        pass
    except Exception as e:
        print(e)


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

The code begins with an import to the service we created. No surprise here.

```
import rclpy
from rclpy.node import Node
from package_with_interfaces.srv import AddPoints


class AddPointsServiceServerNode(Node):
```

The Service Server must be initialised with the `create_service()`, as follows, with parameters that should by now be quite obvious to us.

```
        self.service_server = self.create_service(
            srv_type=AddPoints,
            srv_name='/add_points',
            callback=self.add_points_service_callback)
```

The Service Server receives a `AddPoints.Request` and returns a `AddPoints.Response`.

```
    def add_points_service_callback(self,
                                    request: AddPoints.Request,
                                    response: AddPoints.Response
                                    ) -> AddPoints.Response:
        """
        Adds the two points `a` and `b` in the request and returns the `result`.
        """
```

> **Warning**
>
> The API for the Service Server callback is a bit weird in that needs the `Response` as an argument. This API might change, but for now this is what we got.

We use the members of `AddPoints.Request` to calculate and populate the `AddPoints.Response`. At the end of the callback, we must return that `AddPoints.Request`.

```
        response.result.x = request.a.x + request.b.x
        response.result.y = request.a.y + request.b.y
        response.result.z = request.a.z + request.b.z

        return response
```

The Service Server was quite painless, but it doesn't do much. The Service Client might be a bit more on the painful side for the uninitiated.

## 25.5 Service Clients

In `rclpy`, service clients are implemented using an `asyncio` logic (more info). In this tutorial, we briefly introduce unavoidable `async` concepts in *Python's asyncio*. For extra understanding, please check the official documentation.

The reasons for `asyncio` are very simple. We do not know when the `Response` of a service will be available. Then, when a service is called, we do not want our `Node` with a `ServiceClient` to get stuck while waiting for it. Using `asyncio` allows the `Node` to do other things while the `Response` is received and processed.

In contrast with a `Message`, we need to worry about blocking the `Node` with a `ServiceServer` because a `Service` demands a response.

## 25.6 Create the Node with a Service Client (using a `callback`)

**TL;DR Creating a Service Client (using a `callback`)**

1. Add new dependencies to `package.xml`
2. Import new services `from <package_name>.srv import <srv_name>`
3. In a subclass of `Node`
    1. (*recommended*) wait for service to be available `service_client.wait_for_service(...)`.
    2. (*if periodic*) add a `Timer` with a proper `timer_callback()`
    3. create a callback for the future `def service_future_callback(self, future: Future):`
    4. create a Service Client with `self.service_client = self.create_client(...)`
4. Add the new Node to `setup.py`

### 25.6.1 The Node

**Note**

This example deviates somewhat from what is done in the official examples. This implementation shown herein uses a callback and `rclpy.spin()`. It has many practical applications, but it's no *panacea*.

We start by adding a `add_points_service_client_node.py` at `python_package_that_uses_the_services/python_package_that_uses_the_services` with the following contents.

`add_points_service_client_node.py`

```
import random

import rclpy
from rclpy.task import Future
from rclpy.node import Node

from package_with_interfaces.srv import AddPoints


class AddPointsServiceClientNode(Node):
    """A ROS2 Node with a Service Client for AddPoints, that call the service␣
↪periodically."""

    def __init__(self):
        super().__init__('add_points_service_client')

        self.service_client = self.create_client(
            srv_type=AddPoints,
            srv_name='/add_points')

        while not self.service_client.wait_for_service(timeout_sec=1.0):
            self.get_logger().info(f'service {self.service_client.srv_name} not␣
```

(continues on next page)

(continued from previous page)

```
↪available, waiting...')

        self.future: Future = None

        timer_period: float = 0.5
        self.timer = self.create_timer(
            timer_period_sec=timer_period,
            callback=self.timer_callback)

    def timer_callback(self):
        """Method that is periodically called by the timer."""

        request = AddPoints.Request()

        request.a.x = random.uniform(0, 1000)
        request.a.y = random.uniform(0, 1000)
        request.a.z = random.uniform(0, 1000)

        request.b.x = random.uniform(0, 1000)
        request.b.y = random.uniform(0, 1000)
        request.b.z = random.uniform(0, 1000)

        if self.future is not None and not self.future.done():
            self.future.cancel()  # Cancel the future. The callback will be called with␣
↪Future.result == None.
            self.get_logger().warn("Service Future canceled. The Node took too long to␣
↪process the service call."
                                   "Is the Service Server still alive?")
        self.future = self.service_client.call_async(request)
        self.future.add_done_callback(self.process_response)

    def process_response(self, future: Future):
        """Callback for the future, that will be called when it is done"""
        response = future.result()
        if response is not None:
            self.get_logger().info(f"The result was {(response.result.x, response.result.
↪y, response.result.z)}")
        else:
            self.get_logger().info("The response was None.")


def main(args=None):
    """
    The main function.
    :param args: Not used directly by the user, but used by ROS2 to configure
    certain aspects of the Node.
    """
    try:
        rclpy.init(args=args)

        add_points_service_client_node = AddPointsServiceClientNode()

```

(continues on next page)

(continued from previous page)

```
        rclpy.spin(add_points_service_client_node)
    except KeyboardInterrupt:
        pass
    except Exception as e:
        print(e)


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

### 25.6.2 Imports

To have access to the service, we import it with `from <package>.srv import <Service>`.

```
import random

import rclpy
from rclpy.task import Future
from rclpy.node import Node

from package_with_interfaces.srv import AddPoints
```

### 25.6.3 Instantiate a Service Client

We instantiate a Service Client with `Node.create_client()`. The values of `srv_type` and `srv_name` must match the ones used in the Service Server.

```
        self.service_client = self.create_client(
            srv_type=AddPoints,
            srv_name='/add_points')
```

### 25.6.4 (Recommended) Wait for the Service Server to be available

> **Warning**
>
> The order of execution and speed of Nodes depend on a complicated web of relationships between ROS2, the operating system, and the workload of the machine. It would be naive to expect the server to always be active before the client, even if the server Node is started before the client Node.

In many cases, having the result of the service is of particular importance (hence the use of a service and not messages). In that case, we have to wait until `service_client.wait_for_service()`, as shown below.

```
        while not self.service_client.wait_for_service(timeout_sec=1.0):
            self.get_logger().info(f'service {self.service_client.srv_name} not␣
↪available, waiting...')
```

### 25.6.5 Instantiate a `Future` as a class attribute

As part of the `async` framework, we instantiate a `Future` (More info). In this example it is important to have it as an attribute of the class so that we do not lose the reference to it after the callback.

```
        self.future: Future = None
```

### 25.6.6 Instantiate a Timer

Whenever periodic work must be done, it is recommended to use a `Timer`, as we already learned in *Use a Timer for periodic work (when using rclpy.spin())*.

```
        timer_period: float = 0.5
        self.timer = self.create_timer(
            timer_period_sec=timer_period,
            callback=self.timer_callback)
```

The need for a callback for the `Timer`, should also be no surprise.

```
    def timer_callback(self):
        """Method that is periodically called by the timer."""
```

### 25.6.7 Service Clients use `<srv>.Request()`

Given that services work in a request-response model, the Service Client must instantiate a suitable `<srv>.Request()` and populate its fields before making the service call, as shown below. To make the example more interesting, it randomly switches between two possible quotes.

```
        request = AddPoints.Request()

        request.a.x = random.uniform(0, 1000)
        request.a.y = random.uniform(0, 1000)
        request.a.z = random.uniform(0, 1000)

        request.b.x = random.uniform(0, 1000)
        request.b.y = random.uniform(0, 1000)
        request.b.z = random.uniform(0, 1000)
```

### 25.6.8 Make service calls with `call_async()`

The `async` framework in ROS2 is based on Python's `asyncio` that we already saw in *Python's asyncio*.

> **Note**
>
> At first glance, it might feel that all this trouble to use `async` is unjustified. However, Nodes in practice will hardly ever do one service call and be done. Many Nodes in a complex system will have a composition of many service servers, service clients, publishers, and subscribers. Blocking the entire Node while it waits for the result of a service is, in most cases, a bad design.

The recommended way to call a service is through `call_async()`, which is the reason why we are working with `async` logic. In general, the result of `call_async()`, a `Future`, will not have the result of the service call at the next line of our program.

There are many ways to address the use of a `Future`. One of them, specially tailored to interface `async` with callback-based frameworks is the `Future.add_done_callback()`. If the `Future` is already done by the time we call `add_done_callback()`, it is supposed to call the callback for us.

The benefit of this is that the callback will not block our resources until the response is ready. When the response is ready, and the ROS2 executor gets to processing `Future` callbacks, our callback will be called *automagically*.

```
        if self.future is not None and not self.future.done():
            self.future.cancel()  # Cancel the future. The callback will be called with␣
↪Future.result == None.
            self.get_logger().warn("Service Future canceled. The Node took too long to␣
↪process the service call."
                                   "Is the Service Server still alive?")
        self.future = self.service_client.call_async(request)
        self.future.add_done_callback(self.process_response)
```

Given that we are periodically calling the service, before replace the class `Future` with the next service call, we can check if the service call was done with `Future.done()`. If it is not done, we can use `Future.cancel()` so that our callback can handle this case as well. For instance, if the Service Server has been shutdown, the `Future` would never be done.

```
        if self.future is not None and not self.future.done():
            self.future.cancel()  # Cancel the future. The callback will be called with␣
↪Future.result == None.
            self.get_logger().warn("Service Future canceled. The Node took too long to␣
↪process the service call."
                                   "Is the Service Server still alive?")
        self.future = self.service_client.call_async(request)
        self.future.add_done_callback(self.process_response)
```

### 25.6.9 The Future callback

The callback for the `Future` must receive a `Future` as an argument. Having it as an attribute of the Node's class allows us to access ROS2 method such as `get_logger()` and other contextual information.

The result of the `Future` is obtained using `Future.result()`. The response might be `None` in some cases, so we must check it before trying to use the result, otherwise we will get a nasty exception.

```
    def process_response(self, future: Future):
        """Callback for the future, that will be called when it is done"""
        response = future.result()
        if response is not None:
            self.get_logger().info(f"The result was {(response.result.x, response.result.
↪y, response.result.z)}")
        else:
            self.get_logger().info("The response was None.")
```

## 25.7 Update the `setup.py`

As we already learned in *Making ros2 run work*, we must adjust the `setup.py` to refer to the Nodes we just created.

`setup.py`

```
from setuptools import find_packages, setup

package_name = 'python_package_that_uses_the_services'

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']),
    ],
    install_requires=['setuptools'],
    zip_safe=True,
    maintainer='root',
    maintainer_email='murilo.marinho@manchester.ac.uk',
    description='TODO: Package description',
    license='TODO: License declaration',
    tests_require=['pytest'],
    entry_points={
        'console_scripts': [
            'add_points_service_client_node = '
            'python_package_that_uses_the_services.add_points_service_client_node:main',
            'add_points_service_client_just_once_node = '
            'python_package_that_uses_the_services.add_points_service_client_just_once_
↪node:main',
            'add_points_service_server_node = '
            'python_package_that_uses_the_services.add_points_service_server_node:main',
            'add_points_service_client_introspection_node = '
            'python_package_that_uses_the_services.add_points_service_client_
↪introspection_node:main',
            'add_points_service_server_introspection_node = '
            'python_package_that_uses_the_services.add_points_service_server_
↪introspection_node:main',
        ],
    },
```

(continues on next page)

(continued from previous page)

```
)
```

## 25.8 Build and source

Before we proceed, let us build and source once.

```
cd ~/ros2_tutorial_workspace
colcon build
source install/setup.bash
```

> **Note**
>
> For additional explanation and troubleshooting tips, see *Always source after you build*.

> **Warning**
>
> **colcon** will *not* work properly if your terminal has an active **venv**.

## 25.9 Testing Service Server and Client

```
ros2 run python_package_that_uses_the_services add_points_service_client_node
```

when running the client Node, the server is still not active. In that case, the client node will keep waiting for it, as follows

```
[INFO] [1753667386.959416097] [add_points_service_client]: service /add_points not␣
↪available, waiting...
[INFO] [1753667387.967904375] [add_points_service_client]: service /add_points not␣
↪available, waiting...
[INFO] [1753667388.978200250] [add_points_service_client]: service /add_points not␣
↪available, waiting...
```

In another terminal, we run the **add_points_service_server_node**, as follows

```
ros2 run python_package_that_uses_the_services add_points_service_server_node
```

The server Node will output nothing, whereas the client Node will output, periodically,

```
[INFO] [1753667415.876223138] [add_points_service_client]: The result was (853.
↪122385593111, 613.3399959983066, 722.6376752208978)
[INFO] [1753667416.373657638] [add_points_service_client]: The result was (645.
↪418992882397, 560.9466217293334, 874.7214190239486)
[INFO] [1753667416.875945305] [add_points_service_client]: The result was (1270.
↪7448356640075, 345.69676803639936, 953.6879012399689)
[INFO] [1753667417.376013639] [add_points_service_client]: The result was (1203.
↪944887411107, 733.5131783020975, 927.902266740569)
[INFO] [1753667417.872921291] [add_points_service_client]: The result was (671.
↪9458297091917, 1210.490009902154, 545.6078440547075)
```

(continues on next page)

(continued from previous page)

```
[INFO] [1753667418.371451541] [add_points_service_client]: The result was (629.
↪0766519110047, 872.0699525880541, 581.7396957576223)
[INFO] [1753667418.873213958] [add_points_service_client]: The result was (579.
↪0485532702639, 1714.0365146695003, 396.1743388215037)
[INFO] [1753667419.375147834] [add_points_service_client]: The result was (545.
↪2849740451343, 1629.1832720438556, 945.1871456532875)
```

CHAPTER

# TWENTYSIX

# INSPECTING SERVICES (ROS2 SERVICE)

**ROS2** has a tool to help us inspect services. It is just as helpful as the tools for topics.

```
ros2 service -h
```

which outputs the detailed information of the tool, as shown below. In particular, the highlighted fields are used quite frequently in practice.

```
usage: ros2 service [-h] [--include-hidden-services] Call `ros2 service <command> -h`␣
↪for more detailed usage. ...

Various service related sub-commands

options:
  -h, --help            show this help message and exit
  --include-hidden-services
                        Consider hidden services as well

Commands:
  call  Call a service
  echo  Echo a service
  find  Output a list of available services of a given type
  info  Print information about a service
  list  Output a list of available services
  type  Output a service's type

  Call `ros2 service <command> -h` for more detailed usage.
```

## 26.1 Start a service server

Similar to the discussion about topics, it is good to be able to test service servers without having to develop a complete service client. Let's start by running the service server we created just now.

> **Warning**
>
> Be sure to terminate the Nodes we used in the past section before proceeding (e.g. with CTRL+C), otherwise, the output will look different from what is described here.

```
ros2 run python_package_that_uses_the_services add_points_service_server_node
```

## 26.2 Getting all services with `ros2 service list`

To see all currently active services, we run

```
ros2 service list
```

which, in this case, outputs

```
/add_points
/add_points_service_server/describe_parameters
/add_points_service_server/get_parameter_types
/add_points_service_server/get_parameters
/add_points_service_server/get_type_description
/add_points_service_server/list_parameters
/add_points_service_server/set_parameters
/add_points_service_server/set_parameters_atomically
```

To everyone's surprise, there are a lot of services beyond the one we created. We can address those when we talk about ROS2 parameters, for now, we ignore them.

## 26.3 Testing your service servers with ros2 service call

Like the discussion about topics, ROS2 has a tool to call a service from the terminal, called **`ros2 service call`**. The service must be specified and an instance of its request must be written using YAML. Back to our example, we can do

```
ros2 service call /add_points \
package_with_interfaces/srv/AddPoints \
'{
a: {
     x: 10,
     y: 11,
     z: 12
     },
b: {
     x: -10,
     y: -10,
     z: 22
   }
}'
```

which results in

```
requester: making request: package_with_interfaces.srv.AddPoints_Request(a=geometry_msgs.
↪msg.Point(x=10.0, y=11.0, z=12.0), b=geometry_msgs.msg.Point(x=-10.0, y=-10.0, z=22.0))

response:
package_with_interfaces.srv.AddPoints_Response(result=geometry_msgs.msg.Point(x=0.0, y=1.
↪0, z=34.0))
```

## 26.4 Testing your service clients

To the best of my knowledge, there is no tool inside **`ros2 service`** to allow us to experiment with service clients. For service clients the only way to test them is to make a minimal service server to interact with them. We've already done that, so this topic ends here.

## 26.5 How about `ros2 service echo`

Added in version Jazzy: Added this section.

**See also**

- Official tutorial on service introspection
- https://github.com/ros2/demos/blob/rolling/demo_nodes_py/demo_nodes_py/services/introspection.py

**We'll be working in these files**

```
python_package_that_uses_the_services
|-- package.xml
|-- python_package_that_uses_the_services
|   |-- __init__.py
|   |-- add_points_service_client_introspection_node.py
|   |-- add_points_service_client_node.py
|   |-- add_points_service_server_introspection_node.py
|   `-- add_points_service_server_node.py
|-- resource
|   `-- python_package_that_uses_the_services
|-- setup.cfg
|-- setup.py
`-- test
    |-- test_copyright.py
    |-- test_flake8.py
    `-- test_pep257.py
```

It might be good to know of the existence of *service introspection*. Although it seems to work the documentation might be ongoing, according to this discussion: https://github.com/ros-infrastructure/rep/pull/360.

After proper configuration of the server and client, **`ros2 service echo`** allows us to look at what information is being exchanged between participants. It helps not having to add endless `print` functions throughout the code that hurt performance.

In the folder `~/ros2_tutorial_workspace/src/python_package_that_uses_the_services/python_package_that_uses_the_services` we add the following two files. Note that we are using inheritance from the previously written `AddPointsServiceClientNode` and `AddPointsServiceServerNode` to minimize code replication.

### Introspection Client

For the `Node` with the service client, we must use the `configure_introspection` method. The argument requires a `service_event_qos_profile` and a `introspection_state`. The quality-of-service profile is within the same scope as the quality-of-service for messages and has a similar meaning. The `introspection_state` defines how much of the service can be introspected. Less information can be made accessible if that is necessary.

`add_points_service_client_introspection_node.py`

```
from .add_points_service_client_node import AddPointsServiceClientNode

import rclpy
from rclpy.service import ServiceIntrospectionState
from rclpy.qos import qos_profile_system_default

class AddPointsServiceClientIntrospectionNode(AddPointsServiceClientNode):

    def __init__(self):
        super().__init__()

        # https://github.com/ros2/demos/blob/rolling/demo_nodes_py/demo_nodes_py/services/introspection.py
        self.service_client.configure_introspection(
            clock=self.get_clock(),
            service_event_qos_profile=qos_profile_system_default,
            introspection_state=ServiceIntrospectionState.CONTENTS)

def main(args=None):
    """
    The main function.
    :param args: Not used directly by the user, but used by ROS2 to configure
    certain aspects of the Node.
    """
    try:
        rclpy.init(args=args)

        add_points_service_client_introspection_node = AddPointsServiceClientIntrospectionNode()

        rclpy.spin(add_points_service_client_introspection_node)
    except KeyboardInterrupt:
        pass
    except Exception as e:
        print(e)


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

### Introspection Server

The service server has the same syntax and requirements to activate introspection as the service client.

`add_points_service_server_introspection_node.py`

```python
from .add_points_service_server_node import AddPointsServiceServerNode

import rclpy
from rclpy.service import ServiceIntrospectionState
from rclpy.qos import qos_profile_system_default

class AddPointsServiceServerIntrospectionNode(AddPointsServiceServerNode):

    def __init__(self):
        super().__init__()

        # https://github.com/ros2/demos/blob/rolling/demo_nodes_py/demo_nodes_py/services/introspection.py
        self.service_server.configure_introspection(
            clock=self.get_clock(),
            service_event_qos_profile=qos_profile_system_default,
            introspection_state=ServiceIntrospectionState.CONTENTS)

def main(args=None):
    """
    The main function.
    :param args: Not used directly by the user, but used by ROS2 to configure
    certain aspects of the Node.
    """
    try:
        rclpy.init(args=args)

        add_points_service_server_introspection_node = AddPointsServiceServerIntrospectionNode()

        rclpy.spin(add_points_service_server_introspection_node)
    except KeyboardInterrupt:
        pass
    except Exception as e:
        print(e)


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

## 26.6 Build and source

Before we proceed, let us build and source once.

```
cd ~/ros2_tutorial_workspace
colcon build
source install/setup.bash
```

> **Note**
>
> For additional explanation and troubleshooting tips, see *Always source after you build*.

> **Warning**
>
> **colcon** will *not* work properly if your terminal has an active **venv**.

## 26.7 Finally the ros2 service echo

To be able to use **ros2 service echo**, we must be sure that the service server and **ros2 service echo** are active before the service client requests the service we want to inspect the entire exchange.

It might be a bit of a handful, but we will need three (properly sourced) terminals. In this order.

### (1) service server

```
ros2 run python_package_that_uses_the_services add_points_service_server_introspection_
↪node
```

### (2) ros2 service echo

```
ros2 service echo /add_points
```

### (3) service client

```
ros2 run python_package_that_uses_the_services add_points_service_client_introspection_
↪node
```

Each program will have it's own output, shown below. For the purposes of this section we can focus on the output of **ros2 service echo**. The other two outputs repeat what we have seen in the previous session, further guaranteeing that the introspection works without affecting the overall behavior of the nodes too much.

### ros2 service echo output

As you can see, we will receive a profoundly detailed output showing all aspects of the service, including the request sent, an acknowledgement of what was received, what was replied, and the acknowledgement of the reply!

```
info:
  event_type: REQUEST_SENT
  stamp:
    sec: 1761419221
    nanosec: 796638805
  client_gid:
  - 1
  - 15
  - 235
  - 125
  - 217
  - 40
  - 45
  - 142
  - 0
  - 0
  - 0
```

(continues on next page)

(continued from previous page)

```
  - 0
  - 0
  - 0
  - 20
  - 3
  sequence_number: 1
request:
- a:
    x: 104.79864813644902
    y: 422.844185397337
    z: 0.0
  b:
    x: 531.6723658565676
    y: 761.3967957778456
    z: 357.29827376582836
response: []
---
info:
  event_type: REQUEST_RECEIVED
  stamp:
    sec: 1761419221
    nanosec: 800073888
  client_gid:
  - 1
  - 15
  - 235
  - 125
  - 217
  - 40
  - 45
  - 142
  - 0
  - 0
  - 0
  - 0
  - 0
  - 0
  - 19
  - 4
  sequence_number: 1
request:
- a:
    x: 104.79864813644902
    y: 422.844185397337
    z: 0.0
  b:
    x: 531.6723658565676
    y: 761.3967957778456
    z: 357.29827376582836
response: []
---
info:
```

(continues on next page)

(continued from previous page)

```
  event_type: RESPONSE_SENT
  stamp:
    sec: 1761419221
    nanosec: 801308763
  client_gid:
  - 1
  - 15
  - 235
  - 125
  - 217
  - 40
  - 45
  - 142
  - 0
  - 0
  - 0
  - 0
  - 0
  - 0
  - 19
  - 4
  sequence_number: 1
request: []
response:
- result:
    x: 636.4710139930166
    y: 1184.2409811751827
    z: 357.29827376582836
---
info:
  event_type: RESPONSE_RECEIVED
  stamp:
    sec: 1761419221
    nanosec: 801765763
  client_gid:
  - 1
  - 15
  - 235
  - 125
  - 217
  - 40
  - 45
  - 142
  - 0
  - 0
  - 0
  - 0
  - 0
  - 0
  - 20
  - 3
  sequence_number: 1
```

(continues on next page)

(continued from previous page)

```
request: []
response:
- result:
    x: 636.4710139930166
    y: 1184.2409811751827
    z: 357.29827376582836
---
```

### service client output

For the service client, the output is the same as with the service client without introspection.

```
[INFO] [1761419221.816188138] [add_points_service_client]: The result was (636.
↪4710139930166, 1184.2409811751827, 357.29827376582836)
```

### service server output

Nothing is output in the service server, because our service server is not supposed to output anything to the screen.

CHAPTER

# TWENTYSEVEN

# CALL FOR ACTIONS: ACTION SERVERS AND CLIENTS

**Warning**

This topic is under construction and this might not even be its final form. Please feel free to open an issue if you spot any typos or other problems.

Added in version Jazzy: Added this section.

What about a mixture of messages and services? That is where actions come into play.

We use actions by creating an `ActionServer`. The `ActionServer` can be called by one or more `ActionClients`.

Similarly to a service, each action should only have a single `ActionServer` that will receive a goal (with a `Request`) and provide a `Result`. It will also provide `Feedback` through a suitable topic.

It can be argued that the main difference between a service and an action is the capability of providing feedback while the action is performed. A service, in contrast, only outputs a single, final result of the service call.

## 27.1 Objective

**Note**

Unless otherwise explicitly stated, we will always follow the International System of Units. This means distances in meters, angles in radians, time in seconds, and so on.

We can use an action to represent the first robot-like behavior of this tutorial in an illustrative manner.

Suppose that we have a simple robot that moves in 2D space and whose orientation is not important. Let it have a current position with respect to the $k$-ith iteration represented by (a 2D vector)

$$\mathbb{R}^2 \ni \boldsymbol{p}(k) = \begin{bmatrix} x(k) \\ y(k) \end{bmatrix}$$

and a desired position given by

$$\mathbb{R}^2 \ni \boldsymbol{p}_d = \begin{bmatrix} x_d \\ y_d \end{bmatrix}.$$

Suppose that we want to design an action server that takes this robot-like object from its current position $\boldsymbol{p}$ and moves it towards the goal $\boldsymbol{p}_d$ with a speed $s \in \mathbb{R}$. As feedback, it gives us the distance $d \in \mathbb{R}$ between the current position and the desired position.

The (Euclidean) distance $d$ will be calculated from the terms $x$, $x_d$, $y$, and $y_d$, as follows.

$$d = \sqrt{(x - x_d)^2 + (y - y_d)^2}. \tag{27.1}$$

The action server will update the position $\boldsymbol{p}(k)$ based on a simple constant speed motion of the robot. This can be mathematically described as follows.

$$\boldsymbol{p}(k+1) = \boldsymbol{p}(k) - s \left( \frac{\boldsymbol{p} - \boldsymbol{p}_d}{d} \right) T, \tag{27.2}$$

where $\boldsymbol{p}(k+1)$ represents the next position, $\boldsymbol{p}(k)$ the current position, and $T$ is the sampling time.

## 27.2 Create the package

We start by creating a package to use the action we first created in *The action file*.

```
cd ~/ros2_tutorial_workspace/src
ros2 pkg create python_package_that_uses_the_actions \
--build-type ament_python \
--dependencies rclpy package_with_interfaces
```

### ros2 pkg create output

```
going to create a new package
package name: python_package_that_uses_the_actions
destination directory: ~/ros2_tutorial_workspace/src
package format: 3
version: 0.0.0
description: TODO: Package description
maintainer: ['root <murilo.marinho@manchester.ac.uk>']
licenses: ['TODO: License declaration']
build type: ament_python
dependencies: ['rclpy', 'package_with_interfaces']
creating folder ./python_package_that_uses_the_actions
creating ./python_package_that_uses_the_actions/package.xml
creating source folder
creating folder ./python_package_that_uses_the_actions/python_package_that_uses_the_
↪actions
creating ./python_package_that_uses_the_actions/setup.py
creating ./python_package_that_uses_the_actions/setup.cfg
creating folder ./python_package_that_uses_the_actions/resource
creating ./python_package_that_uses_the_actions/resource/python_package_that_uses_the_
↪actions
creating ./python_package_that_uses_the_actions/python_package_that_uses_the_actions/__
↪init__.py
creating folder ./python_package_that_uses_the_actions/test
creating ./python_package_that_uses_the_actions/test/test_copyright.py
creating ./python_package_that_uses_the_actions/test/test_flake8.py
creating ./python_package_that_uses_the_actions/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:
```

(continues on next page)

(continued from previous page)

```
Apache-2.0
BSL-1.0
BSD-2.0
BSD-2-Clause
BSD-3-Clause
GPL-3.0-only
LGPL-3.0-only
MIT
MIT-0
```

## 27.3 Overview

This will be the file structure for the `Action` tutorial. Highlighted are the main files for the `ActionServer` and `ActionClient`.

**File structure**

```
python_package_that_uses_the_actions/
|-- package.xml
|-- python_package_that_uses_the_actions
|   |-- __init__.py
|   |-- move_straight_in_2d_action_client_node.py
|   `-- move_straight_in_2d_action_server_node.py
|-- resource
|   `-- python_package_that_uses_the_actions
|-- setup.cfg
|-- setup.py
`-- test
    |-- test_copyright.py
    |-- test_flake8.py
    `-- test_pep257.py
```

CHAPTER

# TWENTYEIGHT

# ACTION SERVERS

> **Warning**
>
> This topic is under construction and this might not even be its final form. Please feel free to open an issue if you spot any typos or other problems.

Added in version Jazzy: Added this section.

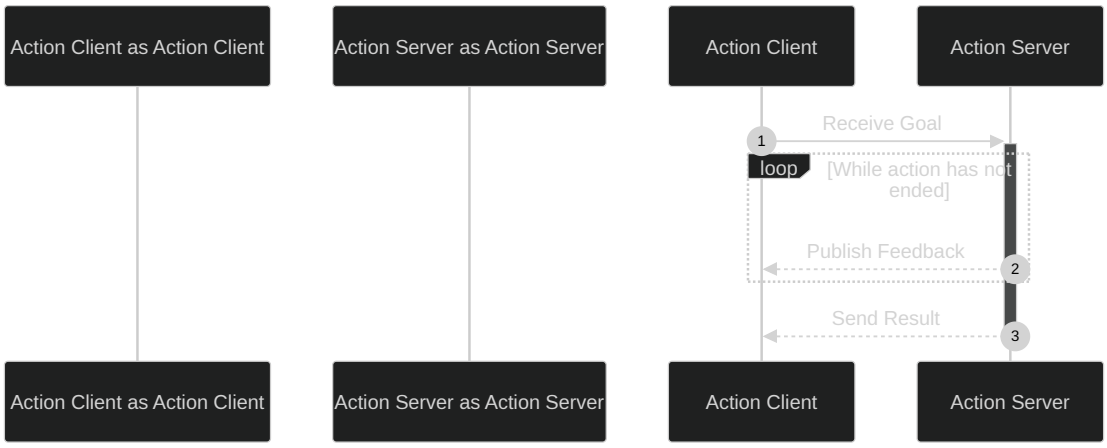
Action servers will rely indirectly on our capabilities of working well with messages and services. Although actions are unique in the way they work, they will repeat many of the structures we used so far. The action server, for instance, will be very much like a service server.

While being conscious of our objectives, for any action server, it is important to:

1. Receive the goal and process it in a meaningful way.
2. Publish feedback as it becomes available. Without feedback, an Action might always be replaced more effectively by a Service.
3. Set the final state of the goal and send a Result.

## 28.1 Diagram

This is the sequence diagram from the point of view of the action server. The major difference with a service server is the need to process and send feedback until the action is completed or aborted.


Action Client as Action Client
Action Server as Action Server
Action Client
Action Server
Receive Goal
1
loop
[While action has not ended]
Publish Feedback
2
Send Result
3
Action Client as Action Client
Action Server as Action Server
Action Client
Action Server

## 28.2 Files

The highlighted files below will be modified or created in this section.

**File structure**

```
python_package_that_uses_the_actions/
|-- package.xml
|-- python_package_that_uses_the_actions
|   |-- __init__.py
|   |-- move_straight_in_2d_action_client_node.py
|   `-- move_straight_in_2d_action_server_node.py
|-- resource
|   `-- python_package_that_uses_the_actions
|-- setup.cfg
|-- setup.py
`-- test
    |-- test_copyright.py
    |-- test_flake8.py
    `-- test_pep257.py
```

## 28.3 Action Server

In this section, we will do the following.

1. Create the node with an `ActionServer`.
2. Update the `setup.py` so that **`ros2 run`** finds the node.

Let us create the action server as follows.

`move_straight_in_2d_action_server_node.py`

```python
import time
from math import sqrt

import rclpy
from rclpy.action import ActionServer
from rclpy.action.server import ServerGoalHandle
from rclpy.node import Node

from geometry_msgs.msg import Point
from package_with_interfaces.action import MoveStraightIn2D


class MoveStraightIn2DActionServerNode(Node):
    """A ROS2 Node with an Action Server for MoveStraightIn2D."""

    def __init__(self):
        super().__init__('move_straight_in_2d_action_server')
        self.current_position = Point()
        self.MAX_ITERATIONS: int  = 100
        self.sampling_time: float = 0.01
```

(continues on next page)

(continued from previous page)

```
        self.action_server = ActionServer(
            self,
            MoveStraightIn2D,
            '/move_straight_in_2d',
            self.execute_callback)



    def get_distance(self, desired_position: Point) -> float:
        """
        Calculates the error norm (e.g. Euclidean distance) between the current position and the desired position.
        Notice that we have chosen to ignore the z-axis as this is a 2D motion.
        """

        x = self.current_position.x
        y = self.current_position.y
        xd = desired_position.x
        yd = desired_position.y

        return sqrt((x - xd) ** 2 + (y - yd) ** 2)

    def move_the_object_with_velocity(self, desired_position: Point, desired_speed: float = 1.0) -> None:
        """
        Moves the object with the desired speed for one iteration. Must be called until objective is reached or
        the controller times out.
        """
        distance = self.get_distance(desired_position)
        # Prevent us from dividing by zero or a small number we currently do not care about
        if distance < 0.01:
            return

        x_direction = (self.current_position.x - desired_position.x) / distance
        y_direction = (self.current_position.y - desired_position.y) / distance

        # Apply new position based on the desired velocity and direction
        self.current_position.x -= x_direction * desired_speed * self.sampling_time
        self.current_position.y -= y_direction * desired_speed * self.sampling_time


    def execute_callback(self, goal: ServerGoalHandle) -> MoveStraightIn2D.Result:
        """
        To be attached to the Action as its callback. Receives a goal position and tries to move to that position.
        It will return the Euclidean distance between the current position and the desired position as the feedback.
        If the goal is reached within a given threshold it will succeed, otherwise it will abort.
```

(continues on next page)

(continued from previous page)

```
        """
        desired_position = goal.request.desired_position

        self.get_logger().info(f'current_position is {self.current_position}.')
        self.get_logger().info(f'desired_position set to {desired_position}.')

        feedback_msg = MoveStraightIn2D.Feedback()

        # Let's limit the maximum number of iterations this can accept
        for i in range(self.MAX_ITERATIONS):
            distance = self.get_distance(desired_position)
            feedback_msg.distance = distance
            goal.publish_feedback(feedback_msg)

            self.move_the_object_with_velocity(desired_position)

            # We define a threshold to see if it managed to reach the goal or not.
            if distance < 0.01:
                goal.succeed()
                break

            # A sleep illustrating the time it would take in real life for a robot to␣
↪move
            time.sleep(self.sampling_time)

        result = MoveStraightIn2D.Result()
        result.final_position = self.current_position
        return result


def main(args=None):
    """
    The main function.
    :param args: Not used directly by the user, but used by ROS2 to configure certain␣
↪aspects of the Node.
    """
    try:
        rclpy.init(args=args)

        node = MoveStraightIn2DActionServerNode()

        rclpy.spin(node)
    except KeyboardInterrupt:
        pass
    except Exception as e:
        print(e)


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

As always, our class must inherit from `rclpy.node.Node`, so that it can be used as argument of `rclpy.spin()`.

An action server is an instance of `rclpy.action.ActionServer`. As input to the initializer of `ActionServer`, we need an action, a string that names this action server, and a suitable callback.

```
class MoveStraightIn2DActionServerNode(Node):
    """A ROS2 Node with an Action Server for MoveStraightIn2D."""

    def __init__(self):
        super().__init__('move_straight_in_2d_action_server')
        self.current_position = Point()
        self.MAX_ITERATIONS: int  = 100
        self.sampling_time: float = 0.01

        self.action_server = ActionServer(
            self,
            MoveStraightIn2D,
            '/move_straight_in_2d',
            self.execute_callback)
```

The callback is likely to be the most complex aspect of an action server. The callback must receive an instance of `ServerGoalHandle`. This will be managed automatically by ROS2. It must return an instance of the class `MyAction.Result` instance for any action `MyAction` you are serving. This must be done manually by the programmer of the server. Note that the `MyAction.Result` class is created for you via **`colcon build`**.

The request can be obtained using `goal.request` of type `MyAction.Request`. Given that our action has the field `desired_position` we can access it this way.

A feedback message must be sent as suitable. One can be created with the `MyAction.Feedback` class, which is created for you via **`colcon build`**. After creating a suitable message, it can be sent with `goal.publish_feedback`.

If the action is successful, it is important to set `goal.succeed`. There are instances in which you would like to abort the goal early. This does not apply to this example because it will fail naturally after it times out. If you do not set a goal as succeeded, it will automatically be aborted at the end of the callback.

Lastly, the method must return an instance of `MyAction.Result`. In our case, we have the field `final_position` in the action file description, therefore we populate it as needed.

```
    def execute_callback(self, goal: ServerGoalHandle) -> MoveStraightIn2D.Result:
        """
        To be attached to the Action as its callback. Receives a goal position and tries␣
↪to move to that position.
        It will return the Euclidean distance between the current position and the␣
↪desired position as the feedback.
        If the goal is reached within a given threshold it will succeed, otherwise it␣
↪will abort.
        """
        desired_position = goal.request.desired_position

        self.get_logger().info(f'current_position is {self.current_position}.')
        self.get_logger().info(f'desired_position set to {desired_position}.')

        feedback_msg = MoveStraightIn2D.Feedback()

        # Let's limit the maximum number of iterations this can accept
        for i in range(self.MAX_ITERATIONS):
            distance = self.get_distance(desired_position)
            feedback_msg.distance = distance
```

(continues on next page)

(continued from previous page)

```
            goal.publish_feedback(feedback_msg)

            self.move_the_object_with_velocity(desired_position)

            # We define a threshold to see if it managed to reach the goal or not.
            if distance < 0.01:
                goal.succeed()
                break

            # A sleep illustrating the time it would take in real life for a robot to␣
↪move
            time.sleep(self.sampling_time)

        result = MoveStraightIn2D.Result()
        result.final_position = self.current_position
        return result
```

The other methods are to support the important aspects of the action server. The `get_distance` method will compute the Euclidean distance between `current_position` and `desired_position`. Remember *The distance equation*. This is represented by the following piece of code.

```
    def get_distance(self, desired_position: Point) -> float:
        """
        Calculates the error norm (e.g. Euclidean distance) between the current position␣
↪and the desired position.
        Notice that we have chosen to ignore the z-axis as this is a 2D motion.
        """

        x = self.current_position.x
        y = self.current_position.y
        xd = desired_position.x
        yd = desired_position.y

        return sqrt((x - xd) ** 2 + (y - yd) ** 2)
```

The action server will update the `current_position` based on a simple constant speed motion of the point. Remember *The controller equation*. This is represented by the following piece of code.

```
    def move_the_object_with_velocity(self, desired_position: Point, desired_speed:␣
↪float = 1.0) -> None:
        """
        Moves the object with the desired speed for one iteration. Must be called until␣
↪objective is reached or
        the controller times out.
        """
        distance = self.get_distance(desired_position)
        # Prevent us from dividing by zero or a small number we currently do not care␣
↪about
        if distance < 0.01:
            return

        x_direction = (self.current_position.x - desired_position.x) / distance
```

(continues on next page)

(continued from previous page)

```
        y_direction = (self.current_position.y - desired_position.y) / distance

        # Apply new position based on the desired velocity and direction
        self.current_position.x -= x_direction * desired_speed * self.sampling_time
        self.current_position.y -= y_direction * desired_speed * self.sampling_time
```

Note that we use an arbitrary minimum distance as stop criterion. Do not expect the distance to ever be zero, given that we are using floating point representation for the numbers. If you do not add a stopping range it is likely that your node will be stuck forever.

## 28.4 Adjusting the `setup.py`

This file will include the directives for the server and client. The one for the server is highlighted below.

```
from setuptools import find_packages, setup

package_name = 'python_package_that_uses_the_actions'

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']),
    ],
    install_requires=['setuptools'],
    zip_safe=True,
    maintainer='root',
    maintainer_email='murilo.marinho@manchester.ac.uk',
    description='TODO: Package description',
    license='TODO: License declaration',
    tests_require=['pytest'],
    entry_points={
        'console_scripts': [
            'move_straight_in_2d_action_server_node = python_package_that_uses_the_
↪actions.move_straight_in_2d_action_server_node:main',
            'move_straight_in_2d_action_client_node = python_package_that_uses_the_
↪actions.move_straight_in_2d_action_client_node:main'
        ],
    },
)
```

## 28.5 Build and source

Before we proceed, let us build and source once.

```
cd ~/ros2_tutorial_workspace
colcon build
source install/setup.bash
```

**Note**

For additional explanation and troubleshooting tips, see *Always source after you build*.

**Warning**

**colcon** will *not* work properly if your terminal has an active **venv**.

## 28.6 Testing the Action Server

We will be working with two terminal windows. The first will run the action server. The second, **ros2 action send_goal**.

### Terminal 1: run action server

```
ros2 run python_package_that_uses_the_actions move_straight_in_2d_action_server_node
```

### Terminal 2: ros2 action send_goal

Please note the `--feedback` flag, used to allow us to see the feedback that comes from the server. Aside from that, the formation of the command is similar to topics and services.

```
ros2 action send_goal --feedback \
/move_straight_in_2d \
package_with_interfaces/action/MoveStraightIn2D \
'{
desired_position:{
    x: 1.0,
    y: 0.0,
    z: 0.0}
}'
```

This is what will be shown in each terminal

### Terminal 1: action server output

```
[INFO] [1759841973.596577423] [move_straight_in_2d_action_server]: current_position is␣
↪geometry_msgs.msg.Point(x=0.0, y=0.0, z=0.0).
[INFO] [1759841973.596757548] [move_straight_in_2d_action_server]: desired_position set␣
↪to geometry_msgs.msg.Point(x=1.0, y=0.0, z=0.0).
```

### Terminal 2: ros2 action send_goal output

```
Waiting for an action server to become available...
Sending goal:
     desired_position:
  x: 1.0
  y: 0.0
  z: 0.0
```

(continues on next page)

(continued from previous page)

```
Goal accepted with ID: 09c38042aaf846f49a87523dddf50900

Feedback:
    distance: 1.0

Feedback:
    distance: 0.9900000095367432

Feedback:
    distance: 0.9800000190734863

Feedback:
    distance: 0.9700000286102295

Feedback:
    distance: 0.9599999785423279

Feedback:
    distance: 0.949999988079071

Feedback:
    distance: 0.9399999976158142

Feedback:
    distance: 0.9300000071525574

Feedback:
    distance: 0.9200000166893005

Feedback:
    distance: 0.9100000262260437

Feedback:
    distance: 0.8999999761581421

Feedback:
    distance: 0.8899999856948853

Feedback:
    distance: 0.8799999952316284

Feedback:
    distance: 0.8700000047683716

Feedback:
    distance: 0.8600000143051147

Feedback:
    distance: 0.8500000238418579

Feedback:
```

(continues on next page)

(continued from previous page)

```
    distance: 0.8399999737739563

Feedback:
    distance: 0.8299999833106995

Feedback:
    distance: 0.8199999928474426

Feedback:
    distance: 0.8100000023841858

Feedback:
    distance: 0.800000011920929

Feedback:
    distance: 0.7900000214576721

Feedback:
    distance: 0.7799999713897705

Feedback:
    distance: 0.7699999809265137

Feedback:
    distance: 0.7599999904632568

Feedback:
    distance: 0.75

Feedback:
    distance: 0.7400000095367432

Feedback:
    distance: 0.7300000190734863

Feedback:
    distance: 0.7200000286102295

Feedback:
    distance: 0.7099999785423279

Feedback:
    distance: 0.699999988079071

Feedback:
    distance: 0.6899999976158142

Feedback:
    distance: 0.6800000071525574

Feedback:
    distance: 0.6700000166893005
```

(continues on next page)

(continued from previous page)

```
Feedback:
    distance: 0.6600000262260437

Feedback:
    distance: 0.6499999761581421

Feedback:
    distance: 0.6399999856948853

Feedback:
    distance: 0.6299999952316284

Feedback:
    distance: 0.6200000047683716

Feedback:
    distance: 0.6100000143051147

Feedback:
    distance: 0.6000000238418579

Feedback:
    distance: 0.5899999737739563

Feedback:
    distance: 0.5799999833106995

Feedback:
    distance: 0.5699999928474426

Feedback:
    distance: 0.5600000023841858

Feedback:
    distance: 0.550000011920929

Feedback:
    distance: 0.5400000214576721

Feedback:
    distance: 0.5299999713897705

Feedback:
    distance: 0.5199999809265137

Feedback:
    distance: 0.5099999904632568

Feedback:
    distance: 0.5
```

(continues on next page)

(continued from previous page)

```
Feedback:
    distance: 0.49000000953674316

Feedback:
    distance: 0.47999998927116394

Feedback:
    distance: 0.4699999988079071

Feedback:
    distance: 0.46000000834465027

Feedback:
    distance: 0.44999998807907104

Feedback:
    distance: 0.4399999976158142

Feedback:
    distance: 0.4300000071525574

Feedback:
    distance: 0.41999998688697815

Feedback:
    distance: 0.4099999964237213

Feedback:
    distance: 0.4000000059604645

Feedback:
    distance: 0.38999998569488525

Feedback:
    distance: 0.3799999952316284

Feedback:
    distance: 0.3700000047683716

Feedback:
    distance: 0.36000001430511475

Feedback:
    distance: 0.3499999940395355

Feedback:
    distance: 0.3400000035762787

Feedback:
    distance: 0.33000001311302185

Feedback:
```

(continues on next page)

(continued from previous page)

```
    distance: 0.3199999928474426

Feedback:
    distance: 0.3100000023841858

Feedback:
    distance: 0.30000001192092896

Feedback:
    distance: 0.28999999165534973

Feedback:
    distance: 0.2800000011920929

Feedback:
    distance: 0.27000001072883606

Feedback:
    distance: 0.25999999046325684

Feedback:
    distance: 0.25

Feedback:
    distance: 0.23999999463558197

Feedback:
    distance: 0.23000000417232513

Feedback:
    distance: 0.2199999988079071

Feedback:
    distance: 0.20999999344348907

Feedback:
    distance: 0.20000000298023224

Feedback:
    distance: 0.1899999976158142

Feedback:
    distance: 0.18000000715255737

Feedback:
    distance: 0.17000000178813934

Feedback:
    distance: 0.1599999964237213

Feedback:
    distance: 0.15000000596046448
```

(continues on next page)

(continued from previous page)

```
Feedback:
    distance: 0.14000000059604645

Feedback:
    distance: 0.12999999523162842

Feedback:
    distance: 0.11999999731779099

Feedback:
    distance: 0.10999999940395355

Feedback:
    distance: 0.10000000149011612

Feedback:
    distance: 0.09000000357627869

Feedback:
    distance: 0.07999999821186066

Feedback:
    distance: 0.07000000029802322

Feedback:
    distance: 0.05999999865889549

Feedback:
    distance: 0.05000000074505806

Feedback:
    distance: 0.03999999910593033

Feedback:
    distance: 0.029999999329447746

Feedback:
    distance: 0.019999999552965164

Feedback:
    distance: 0.009999999776482582

Result:
    final_position:
  x: 0.9900000000000007
  y: 0.0
  z: 0.0

Goal finished with status: SUCCEEDED
```

You can try **send_goal** with various positions. As long as the action server node is not closed, it will keep the state. As soon as the action server is closed, it will lose any internal state.

If the action server is unable to reach the desired goal within the time allowed in the action server, the goal will finish with status `ABORTED`. For instance, the last lines for a goal that is too far from the current position would be as follows. Even then, the node will not crash, and will accept further commands that might succeed or fail.

```
[...]

Feedback:
    distance: 98.0199966430664

Result:
    final_position:
  x: 1.9900000000000015
  y: 0.0
  z: 0.0

Goal finished with status: ABORTED
```

CHAPTER

# TWENTYNINE

# ACTION CLIENTS

**Warning**

This topic is under construction and this might not even be its final form. Please feel free to open an issue if you spot any typos or other problems.

Added in version Jazzy: Added this section.

An action client will be much like a service client, but more complicated. There are at least three callbacks and two futures involved.

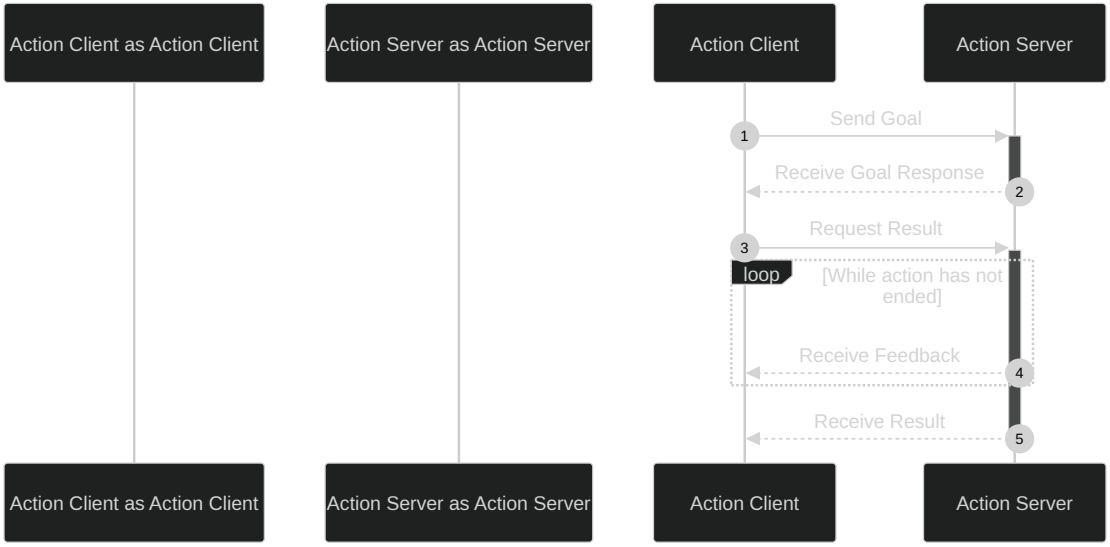
The reason for that is that processing the `Action` requires multiple steps.

To simplify the action server, our example code will only call the `Action` once and do nothing else.

Remember that when deploying actions in real applications they will be part of a more complex `Node` that might include publishers, subscribers, service servers/clients, and other actions server/clients. This means that it is important to take this complexity in consideration when designing your packages to make sure that an `Action` is the best way to communicate.

## 29.1 Diagram

This is the sequence diagram from the point of view of the action client. Note that because we are using `async` calls for the goal and the result, the node is free to do other tasks while those do not arrive.


Action Client as Action Client
Action Server as Action Server
Action Client
Action Server
Send Goal
1
Receive Goal Response
2
Request Result
3
loop
[While action has not ended]
Receive Feedback
4
Receive Result
5
Action Client as Action Client
Action Server as Action Server
Action Client
Action Server

## 29.2 Files

The highlighted file below will be modified or created in this section.

**File structure**

```
python_package_that_uses_the_actions/
|-- package.xml
|-- python_package_that_uses_the_actions
|   |-- __init__.py
|   |-- move_straight_in_2d_action_client_node.py
|   `-- move_straight_in_2d_action_server_node.py
|-- resource
|   `-- python_package_that_uses_the_actions
|-- setup.cfg
|-- setup.py
`-- test
    |-- test_copyright.py
    |-- test_flake8.py
    `-- test_pep257.py
```

## 29.3 Action Client

1. Create the Node with an `ActionClient`.
2. Update the `setup.py` so that **`ros2 run`** finds the node (if needed).

`move_straight_in_2d_action_client_node.py`

```python
import rclpy
from rclpy.action import ActionClient
from rclpy.action.client import ClientGoalHandle
from rclpy.node import Node
from rclpy.task import Future

from geometry_msgs.msg import Point
from package_with_interfaces.action import MoveStraightIn2D

class MoveStraightIn2DActionClientNode(Node):
    """A ROS2 Node with an Action Client for MoveStraightIn2D."""

    def __init__(self):
        super().__init__('move_straight_in_2d_action_client')

        self.action_client = ActionClient(self, MoveStraightIn2D, '/move_straight_in_2d')

        self.send_goal_future = None # This will be used in `send_goal`
        self.get_result_future = None # This will be used in 'goal_response_callback'

    def send_goal_async(self, desired_position: Point) -> None:
        goal_msg = MoveStraightIn2D.Goal()
        goal_msg.desired_position = desired_position
```

(continues on next page)

(continued from previous page)

```
24
25          while not self.action_client.wait_for_server(timeout_sec=1.0):
26              self.get_logger().info(f'action {self.action_client} not available, waiting..
↪.')
27
28          self.get_logger().info(f'Sending goal: {desired_position}.')
29
30          self.send_goal_future = self.action_client.send_goal_async(goal_msg, feedback_
↪callback=self.action_feedback_callback)
31          self.send_goal_future.add_done_callback(self.goal_response_callback)
32
33      def goal_response_callback(self, future: Future) -> None:
34          goal: ClientGoalHandle = future.result()
35
36          if not goal.accepted:
37              self.get_logger().info('Goal was rejected by the server.')
38              return
39          self.get_logger().info('Goal was accepted by the server.')
40
41          self.get_result_future = goal.get_result_async()
42          self.get_result_future.add_done_callback(self.action_result_callback)
43
44      def action_result_callback(self, future: Future) -> None:
45          response: MoveStraightIn2D.Response = future.result()
46          self.get_logger().info(f'Final position was: {response.result.final_position}.')
47
48      def action_feedback_callback(self, feedback_msg: MoveStraightIn2D.Feedback) -> None:
49          feedback = feedback_msg.feedback
50          self.get_logger().info(f'Received feedback distance: {feedback.distance}.')
51
52
53  def main(args=None):
54      """
55      The main function.
56      :param args: Not used directly by the user, but used by ROS2 to configure certain␣
↪aspects of the Node.
57      """
58      try:
59          rclpy.init(args=args)
60
61          node = MoveStraightIn2DActionClientNode()
62
63          # Send the goal once and then do nothing until the user shuts this node down.
64          desired_position = Point()
65          desired_position.x = 1.0
66          desired_position.y = -1.0
67          node.send_goal_async(desired_position)
68
69          rclpy.spin(node)
70      except KeyboardInterrupt:
71          pass
72      except Exception as e:
```

(continues on next page)

(continued from previous page)

```
        print(e)


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

As always, our class must inherit from `rclpy.node.Node`, so that it can be used as argument of `rclpy.spin()`.

An action client is an instance of `rclpy.action.ActionClient`. As input to the initializer of `ActionClient`, we need an action and a string that names the action server this client will interact with.

In the action client, we will have to manage two instances of `Future`. One for the goal and another for the result. It is important to keep them as attributes of the instance to guarantee they will not get out of scope before the result is obtained. This is done as follows.

```
    def __init__(self):
        super().__init__('move_straight_in_2d_action_client')

        self.action_client = ActionClient(self, MoveStraightIn2D, '/move_straight_in_2d')

        self.send_goal_future = None # This will be used in `send_goal`
        self.get_result_future = None # This will be used in 'goal_response_callback'
```

The following method can be used to send the action goal and trigger the entire process. Notably, suppose that a suitable action has already been instantiated and is sent as argument to this method. We wait for the action server to be available. Then, in something that should be familiar after working with services, we send the request asynchronously.

In the request, `send_goal_async`, we define a callback for the feedback. This is similar to a callback for subscribers. Whenever a feedback is sent, the method we assign as callback will be called.

After we send the goal request and assign a callback for the feedback, we assign another callback. This second callback is for the result of the goal request operation.

```
    def send_goal_async(self, desired_position: Point) -> None:
        goal_msg = MoveStraightIn2D.Goal()
        goal_msg.desired_position = desired_position

        while not self.action_client.wait_for_server(timeout_sec=1.0):
            self.get_logger().info(f'action {self.action_client} not available, waiting..
↪.')

        self.get_logger().info(f'Sending goal: {desired_position}.')

        self.send_goal_future = self.action_client.send_goal_async(goal_msg, feedback_
↪callback=self.action_feedback_callback)
        self.send_goal_future.add_done_callback(self.goal_response_callback)
```

After the goal is processed, the goal-related callback will be automatically called for us given that it was added to the respective future. The goal can be rejected, therefore we address that case by doing nothing. If the goal is accepted, we are ready to ask for a result.

We ask for a result asynchronously. We then assign a third callback for when a response is sent.

```
    def goal_response_callback(self, future: Future) -> None:
        goal: ClientGoalHandle = future.result()
```

(continues on next page)

(continued from previous page)

```
        if not goal.accepted:
            self.get_logger().info('Goal was rejected by the server.')
            return
        self.get_logger().info('Goal was accepted by the server.')

        self.get_result_future = goal.get_result_async()
        self.get_result_future.add_done_callback(self.action_result_callback)
```

The result callback will receive a future and process the result of the action somehow. In this toy example, we simply print the results to the screen.

```
    def action_result_callback(self, future: Future) -> None:
        response: MoveStraightIn2D.Response = future.result()
        self.get_logger().info(f'Final position was: {response.result.final_position}.')
```

Lastly, we see that the callback for the feedback is, again, very similar to that for a subscriber.

```
    def action_feedback_callback(self, feedback_msg: MoveStraightIn2D.Feedback) -> None:
        feedback = feedback_msg.feedback
        self.get_logger().info(f'Received feedback distance: {feedback.distance}.')
```

To simplify this example slightly, please notice that the action is instantiated in the `main` function, and the action is triggered only once.

When doing so, please remember to call the action after the node is instantiated and before `rclpy.spin()`. Otherwise, the object will not yet exist, or you will be locked in the spinner before sending the goal.

```
        node = MoveStraightIn2DActionClientNode()

        # Send the goal once and then do nothing until the user shuts this node down.
        desired_position = Point()
        desired_position.x = 1.0
        desired_position.y = -1.0
        node.send_goal_async(desired_position)

        rclpy.spin(node)
```

## 29.4 Build and source

Before we proceed, let us build and source once.

```
cd ~/ros2_tutorial_workspace
colcon build
source install/setup.bash
```

> **Note**
>
> For additional explanation and troubleshooting tips, see *Always source after you build*.

> **Warning**
>
> **colcon** will *not* work properly if your terminal has an active **venv**.

## 29.5 Testing the Action Client

We will be working with two terminal windows. The first will run the action server. The second, the client.

### Terminal 1: run action server

```
ros2 run python_package_that_uses_the_actions move_straight_in_2d_action_server_node
```

### Terminal 2: run action client

```
ros2 run python_package_that_uses_the_actions move_straight_in_2d_action_client_node
```

The output in each window should be something similar to

### Terminal 1: action server output

The action server will make the point move towards the goal. If it does not manage to reach the desired threshold within the allocated time, it will show, in this case, a warning that it was implicitly aborted.

```
[INFO] [1761753120.100902419] [move_straight_in_2d_action_server]: current_position is␣
↪geometry_msgs.msg.Point(x=0.0, y=0.0, z=0.0).
[INFO] [1761753120.101090669] [move_straight_in_2d_action_server]: desired_position set␣
↪to geometry_msgs.msg.Point(x=1.0, y=-1.0, z=0.0).
[WARN] [1761753121.235096837] [move_straight_in_2d_action_server.action_server]: Goal␣
↪state not set, assuming aborted. Goal ID: [  9  49  65  27 156  53  65  61 134 216 224␣
↪156 147 240 198  62]
```

### Terminal 2: action output

The action client will show the feedback as it happens and the final state of the action when the action server stops processing this goal.

```
[INFO] [1761753120.092259753] [move_straight_in_2d_action_client]: Sending goal:␣
↪geometry_msgs.msg.Point(x=1.0, y=-1.0, z=0.0).
[INFO] [1761753120.093185086] [move_straight_in_2d_action_client]: Goal was accepted by␣
↪the server.
[INFO] [1761753120.101412211] [move_straight_in_2d_action_client]: Received feedback␣
↪distance: 1.4142135381698608.
[INFO] [1761753120.112173919] [move_straight_in_2d_action_client]: Received feedback␣
↪distance: 1.404213547706604.
[INFO] [1761753120.123090294] [move_straight_in_2d_action_client]: Received feedback␣
↪distance: 1.3942135572433472.
[INFO] [1761753120.134245086] [move_straight_in_2d_action_client]: Received feedback␣
↪distance: 1.3842135667800903.
[INFO] [1761753120.145173920] [move_straight_in_2d_action_client]: Received feedback␣
↪distance: 1.3742135763168335.
[INFO] [1761753120.156160961] [move_straight_in_2d_action_client]: Received feedback␣
```

(continues on next page)

(continued from previous page)

```
↪distance: 1.3642135858535767.
[INFO] [1761753120.167298045] [move_straight_in_2d_action_client]: Received feedback␣
↪distance: 1.3542135953903198.
[INFO] [1761753120.178755836] [move_straight_in_2d_action_client]: Received feedback␣
↪distance: 1.344213604927063.
[INFO] [1761753120.189400378] [move_straight_in_2d_action_client]: Received feedback␣
↪distance: 1.3342136144638062.
[INFO] [1761753120.200484253] [move_straight_in_2d_action_client]: Received feedback␣
↪distance: 1.3242135047912598.
[INFO] [1761753120.211245128] [move_straight_in_2d_action_client]: Received feedback␣
↪distance: 1.314213514328003.
[INFO] [1761753120.222122295] [move_straight_in_2d_action_client]: Received feedback␣
↪distance: 1.304213523864746.
[INFO] [1761753120.234860420] [move_straight_in_2d_action_client]: Received feedback␣
↪distance: 1.2942135334014893.
[INFO] [1761753120.245268795] [move_straight_in_2d_action_client]: Received feedback␣
↪distance: 1.2842135429382324.
[INFO] [1761753120.257182295] [move_straight_in_2d_action_client]: Received feedback␣
↪distance: 1.2742135524749756.
[INFO] [1761753120.268218170] [move_straight_in_2d_action_client]: Received feedback␣
↪distance: 1.2642135620117188.
[INFO] [1761753120.279651420] [move_straight_in_2d_action_client]: Received feedback␣
↪distance: 1.254213571548462.
[INFO] [1761753120.290934753] [move_straight_in_2d_action_client]: Received feedback␣
↪distance: 1.244213581085205.
[INFO] [1761753120.302183086] [move_straight_in_2d_action_client]: Received feedback␣
↪distance: 1.2342135906219482.
[INFO] [1761753120.313913295] [move_straight_in_2d_action_client]: Received feedback␣
↪distance: 1.2242136001586914.
[INFO] [1761753120.325966795] [move_straight_in_2d_action_client]: Received feedback␣
↪distance: 1.2142136096954346.
[INFO] [1761753120.336757795] [move_straight_in_2d_action_client]: Received feedback␣
↪distance: 1.2042136192321777.
[INFO] [1761753120.347904086] [move_straight_in_2d_action_client]: Received feedback␣
↪distance: 1.1942135095596313.
[INFO] [1761753120.359854920] [move_straight_in_2d_action_client]: Received feedback␣
↪distance: 1.1842135190963745.
[INFO] [1761753120.371660170] [move_straight_in_2d_action_client]: Received feedback␣
↪distance: 1.1742135286331177.
[INFO] [1761753120.382972545] [move_straight_in_2d_action_client]: Received feedback␣
↪distance: 1.1642135381698608.
[INFO] [1761753120.396173336] [move_straight_in_2d_action_client]: Received feedback␣
↪distance: 1.154213547706604.
[INFO] [1761753120.406693836] [move_straight_in_2d_action_client]: Received feedback␣
↪distance: 1.1442135572433472.
[INFO] [1761753120.418283586] [move_straight_in_2d_action_client]: Received feedback␣
↪distance: 1.1342135667800903.
[INFO] [1761753120.430012878] [move_straight_in_2d_action_client]: Received feedback␣
↪distance: 1.1242135763168335.
[INFO] [1761753120.440861920] [move_straight_in_2d_action_client]: Received feedback␣
↪distance: 1.1142135858535767.
[INFO] [1761753120.452906586] [move_straight_in_2d_action_client]: Received feedback␣
```

(continues on next page)

(continued from previous page)

```
↪distance: 1.1042135953903198.
[INFO] [1761753120.464116961] [move_straight_in_2d_action_client]: Received feedback␣
↪distance: 1.094213604927063.
[INFO] [1761753120.475175961] [move_straight_in_2d_action_client]: Received feedback␣
↪distance: 1.0842136144638062.
[INFO] [1761753120.486437628] [move_straight_in_2d_action_client]: Received feedback␣
↪distance: 1.0742135047912598.
[INFO] [1761753120.497304670] [move_straight_in_2d_action_client]: Received feedback␣
↪distance: 1.064213514328003.
[INFO] [1761753120.508036878] [move_straight_in_2d_action_client]: Received feedback␣
↪distance: 1.054213523864746.
[INFO] [1761753120.519005753] [move_straight_in_2d_action_client]: Received feedback␣
↪distance: 1.0442135334014893.
[INFO] [1761753120.529962128] [move_straight_in_2d_action_client]: Received feedback␣
↪distance: 1.0342135429382324.
[INFO] [1761753120.540856920] [move_straight_in_2d_action_client]: Received feedback␣
↪distance: 1.0242135524749756.
[INFO] [1761753120.551795461] [move_straight_in_2d_action_client]: Received feedback␣
↪distance: 1.0142135620117188.
[INFO] [1761753120.563487586] [move_straight_in_2d_action_client]: Received feedback␣
↪distance: 1.004213571548462.
[INFO] [1761753120.575029170] [move_straight_in_2d_action_client]: Received feedback␣
↪distance: 0.9942135810852051.
[INFO] [1761753120.585385753] [move_straight_in_2d_action_client]: Received feedback␣
↪distance: 0.9842135906219482.
[INFO] [1761753120.596445128] [move_straight_in_2d_action_client]: Received feedback␣
↪distance: 0.9742135405540466.
[INFO] [1761753120.607320045] [move_straight_in_2d_action_client]: Received feedback␣
↪distance: 0.9642135500907898.
[INFO] [1761753120.617953295] [move_straight_in_2d_action_client]: Received feedback␣
↪distance: 0.954213559627533.
[INFO] [1761753120.628741545] [move_straight_in_2d_action_client]: Received feedback␣
↪distance: 0.9442135691642761.
[INFO] [1761753120.640129795] [move_straight_in_2d_action_client]: Received feedback␣
↪distance: 0.9342135787010193.
[INFO] [1761753120.651860920] [move_straight_in_2d_action_client]: Received feedback␣
↪distance: 0.9242135882377625.
[INFO] [1761753120.664198961] [move_straight_in_2d_action_client]: Received feedback␣
↪distance: 0.9142135381698608.
[INFO] [1761753120.674831961] [move_straight_in_2d_action_client]: Received feedback␣
↪distance: 0.904213547706604.
[INFO] [1761753120.686480336] [move_straight_in_2d_action_client]: Received feedback␣
↪distance: 0.8942135572433472.
[INFO] [1761753120.696874711] [move_straight_in_2d_action_client]: Received feedback␣
↪distance: 0.8842135667800903.
[INFO] [1761753120.708105420] [move_straight_in_2d_action_client]: Received feedback␣
↪distance: 0.8742135763168335.
[INFO] [1761753120.720106128] [move_straight_in_2d_action_client]: Received feedback␣
↪distance: 0.8642135858535767.
[INFO] [1761753120.731688795] [move_straight_in_2d_action_client]: Received feedback␣
↪distance: 0.854213535785675.
[INFO] [1761753120.744141753] [move_straight_in_2d_action_client]: Received feedback␣
```

(continues on next page)

(continued from previous page)

```
↪distance: 0.8442135453224182.
[INFO] [1761753120.755073711] [move_straight_in_2d_action_client]: Received feedback␣
↪distance: 0.8342135548591614.
[INFO] [1761753120.767188003] [move_straight_in_2d_action_client]: Received feedback␣
↪distance: 0.8242135643959045.
[INFO] [1761753120.778392795] [move_straight_in_2d_action_client]: Received feedback␣
↪distance: 0.8142135739326477.
[INFO] [1761753120.789267920] [move_straight_in_2d_action_client]: Received feedback␣
↪distance: 0.8042135834693909.
[INFO] [1761753120.800152128] [move_straight_in_2d_action_client]: Received feedback␣
↪distance: 0.7942135334014893.
[INFO] [1761753120.811231461] [move_straight_in_2d_action_client]: Received feedback␣
↪distance: 0.7842135429382324.
[INFO] [1761753120.822597753] [move_straight_in_2d_action_client]: Received feedback␣
↪distance: 0.7742135524749756.
[INFO] [1761753120.833374087] [move_straight_in_2d_action_client]: Received feedback␣
↪distance: 0.7642135620117188.
[INFO] [1761753120.845155503] [move_straight_in_2d_action_client]: Received feedback␣
↪distance: 0.7542135715484619.
[INFO] [1761753120.857314462] [move_straight_in_2d_action_client]: Received feedback␣
↪distance: 0.7442135810852051.
[INFO] [1761753120.868129378] [move_straight_in_2d_action_client]: Received feedback␣
↪distance: 0.7342135906219482.
[INFO] [1761753120.879274587] [move_straight_in_2d_action_client]: Received feedback␣
↪distance: 0.7242135405540466.
[INFO] [1761753120.890035378] [move_straight_in_2d_action_client]: Received feedback␣
↪distance: 0.7142135500907898.
[INFO] [1761753120.900985587] [move_straight_in_2d_action_client]: Received feedback␣
↪distance: 0.704213559627533.
[INFO] [1761753120.911983378] [move_straight_in_2d_action_client]: Received feedback␣
↪distance: 0.6942135691642761.
[INFO] [1761753120.923409587] [move_straight_in_2d_action_client]: Received feedback␣
↪distance: 0.6842135787010193.
[INFO] [1761753120.934035170] [move_straight_in_2d_action_client]: Received feedback␣
↪distance: 0.6742135882377625.
[INFO] [1761753120.945181837] [move_straight_in_2d_action_client]: Received feedback␣
↪distance: 0.6642135381698608.
[INFO] [1761753120.957020503] [move_straight_in_2d_action_client]: Received feedback␣
↪distance: 0.654213547706604.
[INFO] [1761753120.968711462] [move_straight_in_2d_action_client]: Received feedback␣
↪distance: 0.6442135572433472.
[INFO] [1761753120.980621087] [move_straight_in_2d_action_client]: Received feedback␣
↪distance: 0.6342135667800903.
[INFO] [1761753120.991584587] [move_straight_in_2d_action_client]: Received feedback␣
↪distance: 0.6242135763168335.
[INFO] [1761753121.003069045] [move_straight_in_2d_action_client]: Received feedback␣
↪distance: 0.6142135858535767.
[INFO] [1761753121.015031753] [move_straight_in_2d_action_client]: Received feedback␣
↪distance: 0.604213535785675.
[INFO] [1761753121.027384962] [move_straight_in_2d_action_client]: Received feedback␣
↪distance: 0.5942135453224182.
[INFO] [1761753121.037966962] [move_straight_in_2d_action_client]: Received feedback␣
```

(continues on next page)

(continued from previous page)

```
↪distance: 0.5842135548591614.
[INFO] [1761753121.048998128] [move_straight_in_2d_action_client]: Received feedback␣
↪distance: 0.5742135643959045.
[INFO] [1761753121.061886295] [move_straight_in_2d_action_client]: Received feedback␣
↪distance: 0.5642135739326477.
[INFO] [1761753121.072371337] [move_straight_in_2d_action_client]: Received feedback␣
↪distance: 0.5542135834693909.
[INFO] [1761753121.084468795] [move_straight_in_2d_action_client]: Received feedback␣
↪distance: 0.5442135334014893.
[INFO] [1761753121.096903712] [move_straight_in_2d_action_client]: Received feedback␣
↪distance: 0.5342135429382324.
[INFO] [1761753121.107056378] [move_straight_in_2d_action_client]: Received feedback␣
↪distance: 0.5242135524749756.
[INFO] [1761753121.118114628] [move_straight_in_2d_action_client]: Received feedback␣
↪distance: 0.5142135620117188.
[INFO] [1761753121.128596212] [move_straight_in_2d_action_client]: Received feedback␣
↪distance: 0.5042135715484619.
[INFO] [1761753121.140370212] [move_straight_in_2d_action_client]: Received feedback␣
↪distance: 0.4942135512828827.
[INFO] [1761753121.152598753] [move_straight_in_2d_action_client]: Received feedback␣
↪distance: 0.48421356081962585.
[INFO] [1761753121.165302628] [move_straight_in_2d_action_client]: Received feedback␣
↪distance: 0.474213570356369.
[INFO] [1761753121.178417045] [move_straight_in_2d_action_client]: Received feedback␣
↪distance: 0.4642135500907898.
[INFO] [1761753121.189332878] [move_straight_in_2d_action_client]: Received feedback␣
↪distance: 0.45421355962753296.
[INFO] [1761753121.199914878] [move_straight_in_2d_action_client]: Received feedback␣
↪distance: 0.4442135691642761.
[INFO] [1761753121.211306087] [move_straight_in_2d_action_client]: Received feedback␣
↪distance: 0.4342135488986969.
[INFO] [1761753121.223468503] [move_straight_in_2d_action_client]: Received feedback␣
↪distance: 0.42421355843544006.
[INFO] [1761753121.237240878] [move_straight_in_2d_action_client]: Final position was:␣
↪geometry_msgs.msg.Point(x=0.7071067811865476, y=-0.7071067811865476, z=0.0).
```

CHAPTER

# THIRTY

# FRAME TRANSFORMATIONS

> **Warning**
>
> This topic is under construction and this might not even be its final form. Please feel free to open an issue if you spot any typos or other problems.

Unless made of flexible materials, the robots we use and the objects they interact with can be well simplified as rigid bodies.

Terminology dictates that a *position* and an *orientation* are inherent elements of rigid bodies, related to information about their instantaneous reference frames. In this sense, a *rotation* and a *translation* would represent the position and the orientation of a rigid body with respect to a reference frame and that implies dislocation.

However, mathematically, positions and orientations are *always* represented with respect to a reference frame. I believe that, because of this, beyond this tutorial, in robotics and computer science, you will often see position/translation and orientation/rotation used interchangeably.

Please allow me to use only the terms *translation* and *rotation* henceforth to keep the discussion consistent with how it is utilised in `tf2`, namely, as a `geometry_msgs/msg/TransformStamped`.

We start by looking at how it is done in **ROS2**, go through the related mathematics, then create an example to show how to create these transformations.

## 30.1 Transformations in ROS2

Let us take a look at message most commonly used in **ROS2** to represent translation and rotation, `TransformStamped.msg`, part of the package `geometry_msgs`.

We can see its contents with the following command.

```
ros2 interface show geometry_msgs/msg/TransformStamped
```

It results in the slightly intricate message below.

#### Results for `geometry_msgs/msg/TransformStamped`.

```
# This expresses a transform from coordinate frame header.frame_id
# to the coordinate frame child_frame_id at the time of header.stamp
#
# This message is mostly used by the
# <a href="https://docs.ros.org/en/rolling/p/tf2/">tf2</a> package.
# See its documentation for more information.
```

(continues on next page)

(continued from previous page)

```
#
# The child_frame_id is necessary in addition to the frame_id
# in the Header to communicate the full reference for the transform
# in a self contained message.

# The frame id in the header is used as the reference frame of this transform.
std_msgs/Header header
        builtin_interfaces/Time stamp
                int32 sec
                uint32 nanosec
        string frame_id

# The frame id of the child frame to which this transform points.
string child_frame_id

# Translation and rotation in 3-dimensions of child_frame_id from header.frame_id.
Transform transform
        Vector3 translation
                float64 x
                float64 y
                float64 z
        Quaternion rotation
                float64 x 0
                float64 y 0
                float64 z 0
                float64 w 1
```

To simplify the discussion, these are the contents of `TransformStamped.msg`.

```
# This expresses a transform from coordinate frame header.frame_id
# to the coordinate frame child_frame_id at the time of header.stamp
#
# This message is mostly used by the
# <a href="https://docs.ros.org/en/rolling/p/tf2/">tf2</a> package.
# See its documentation for more information.
#
# The child_frame_id is necessary in addition to the frame_id
# in the Header to communicate the full reference for the transform
# in a self contained message.

# The frame id in the header is used as the reference frame of this transform.
std_msgs/Header header

# The frame id of the child frame to which this transform points.
string child_frame_id

# Translation and rotation in 3-dimensions of child_frame_id from header.frame_id.
Transform transform
```

Note that it is a message composed of two other messages, and a built-in.

The `Header` is an essential timestamp used throughout **ROS2**. We can see its contents with the following command.

```
ros2 interface show std_msgs/msg/Header
```

The command results in the following output.

```
# Standard metadata for higher-level stamped data types.
# This is generally used to communicate timestamped data
# in a particular coordinate frame.

# Two-integer timestamp that is expressed as seconds and nanoseconds.
builtin_interfaces/Time stamp
        int32 sec
        uint32 nanosec

# Transform frame with which this data is associated.
string frame_id
```

The elements `stamp` and `frame_id` are used by many popular packages **ROS2** to display and relate translational and rotational data of robots and objects.

Then, to the most important part of this section, we have the `Transform.msg`, whose contents can be obtained with the following command.

```
ros2 interface show geometry_msgs/msg/Transform
```

The command outputs the following.

```
# This represents the transform between two coordinate frames in free space.

Vector3 translation
        float64 x
        float64 y
        float64 z
Quaternion rotation
        float64 x 0
        float64 y 0
        float64 z 0
        float64 w 1
```

In this message type, the `translation` and `rotation` can be clearly seen.

## 30.2 Translations

Translations $\boldsymbol{t} \in \mathbb{R}^3$ can be easily represented using imaginary numbers, such as

$$\boldsymbol{t} \triangleq t_x\hat{\imath} + t_y\hat{\jmath} + t_z\hat{k}, \tag{30.1}$$

where $\hat{\imath}^2 = \hat{\jmath}^2 = \hat{k}^2 = \hat{\imath}\hat{\jmath}\hat{k} = -1$.

If you thought this would never be useful, think again. It's about to be!

## 30.3 Rotations: Quaternions

**Note**

I'm not explaining this (in a failed attempt) to show off. **ROS2** represents rotations as quaternions, so this knowledge is needed in the short term as well.

Whereas translations are intuitive and can be easily processed with A-levels mathematics, rotations demand extra thought. However, the complexity is exaggerated in multiple sources to some extent.

In **ROS2**, rotations are represented as quaternions. They have several benefits over other representations, which you can see in related literature on quaternions. For robotics, it is easier to think of rotations directly in quaternions instead of relying on intermediary ways, such as Euler angles. The reason is that the particularities of quaternions will eventually catch up to you, no matter how long and how far one might try to run away from them.

Let's rip the trademarked plaster brand out. The following equation represents the formation of a rotation quaternion.

$$\boldsymbol{r} \triangleq \cos\left(\frac{\phi}{2}\right) + \boldsymbol{v}\sin\left(\frac{\phi}{2}\right), \tag{30.2}$$

where $\boldsymbol{v}^2 = -1$. This means that the rotation axis $\boldsymbol{v}$ can be any imaginary number such that $||\boldsymbol{v}|| = 1$.

The easiest way to think about a rotation using quaternions is to think about the axis of rotation $\boldsymbol{v}$ and the angle of rotation $\phi$. Then, you construct the quaternion with the *rotation quaternion formation law*.

**Examples**

We can choose $\phi = 0$ and see how a quaternion represents no rotation. Regardless of rotation axis, this results in

$$\begin{aligned}\boldsymbol{r}_0 &\triangleq \cos\left(\frac{0}{2}\right) + \boldsymbol{v}\sin\left(\frac{0}{2}\right)\\ &= 1.\end{aligned}$$

Now, suppose that we have a rotation of $\phi = \pi$ radians (angles always in radians, don't forget!).

If we want such a rotation about the x-axis, we choose $\boldsymbol{v}_1 = \hat{\imath}$. Therefore, this rotation would be correctly represented by

$$\begin{aligned}\boldsymbol{r}_1 &\triangleq \cos\left(\frac{\pi}{2}\right) + \hat{\imath}\sin\left(\frac{\pi}{2}\right)\\ &= \hat{\imath}.\end{aligned}$$

If we want such a rotation about the z-axis, we choose $\boldsymbol{v}_2 = \hat{k}$, correctly represented by

$$\begin{aligned}\boldsymbol{r}_2 &\triangleq \cos\left(\frac{\pi}{2}\right) + \hat{k}\sin\left(\frac{\pi}{2}\right)\\ &= \hat{k}.\end{aligned}$$

Any $\phi \in \mathbb{R}$ is acceptable and, more importantly, any $\boldsymbol{v}$ is acceptable as long as it is imaginary and has norm one. For instance, $\boldsymbol{v}_3 = -\frac{\sqrt{2}}{2}\hat{\imath} + \frac{\sqrt{2}}{2}\hat{k}$ leads to the valid rotation quaternion

$$\begin{aligned} \boldsymbol{r}_3 &\triangleq \cos\left(\frac{\pi}{2}\right) + \left(-\frac{\sqrt{2}}{2}\hat{\imath} + \frac{\sqrt{2}}{2}\hat{k}\right)\sin\left(\frac{\pi}{2}\right) \\ &= \left(-\frac{\sqrt{2}}{2}\hat{\imath} + \frac{\sqrt{2}}{2}\hat{k}\right). \end{aligned}$$

**Error**

There are also common pitfalls.

1. Using a rotation axis that is not unit norm does **not** represent a rotation quaternion.

   Using $\boldsymbol{v}_4 = -\hat{\imath} + \hat{k}$ is **not** a valid rotation quaternion because its norm is not one, $||\boldsymbol{v}_4|| = \sqrt{2}$.

2. Misunderstanding the rotation angle.

   Note that, inside the quaternion, we have $\frac{\phi}{2}$. For instance, the quaternion

   $$\boldsymbol{r}_5 \triangleq \cos\left(\frac{\pi}{4}\right) + \hat{\jmath}\sin\left(\frac{\pi}{4}\right),$$

   represents a rotation of $\phi_5 = \frac{\pi}{2}$ about the y-axis, **not** $\frac{\pi}{4}$.

### 30.3.1 Sequential rotations

Sequential rotations can be obtained via quaternion multiplication. For instance, to obtain the result of a rotation of $\phi = \pi$ about the x-axis followed by a rotation of the same angle about the z-axis, we do

$$\boldsymbol{r}_{1,2} \triangleq \boldsymbol{r}_1 \boldsymbol{r}_2,$$

which results, in this case,

$$\begin{aligned} \boldsymbol{r}_{1,2} &= \left[\cos\left(\frac{\pi}{2}\right) + \hat{\imath}\sin\left(\frac{\pi}{2}\right)\right]\left[\cos\left(\frac{\pi}{2}\right) + \hat{k}\sin\left(\frac{\pi}{2}\right)\right] \\ &= \hat{\imath}\hat{k} \\ &= -\hat{\jmath}. \end{aligned}$$

The product of any two quaternions can always be done algebraically, by respecting the multiplication between the basis elements. The result can be generalised by the quaternion multiplication. This generalization will not be repeated here as we will do it programmatically later.

### 30.3.2 Inverse rotations

The inverse rotation can be obtained by flipping the axis of rotation. This is equivalent to obtaining the so-called quaternion conjugate of unit-norm quaternions.

$$\boldsymbol{r}^* \triangleq \cos\left(\frac{\phi}{2}\right) - \boldsymbol{v}\sin\left(\frac{\phi}{2}\right). \tag{30.3}$$

We can see that this is indeed the inverse rotation by noticing that the rotation multiplied by its conjugate results in 1, that is, the non rotation.

$$\begin{aligned}
\boldsymbol{r}\boldsymbol{r}^* = \boldsymbol{r}^*\boldsymbol{r} &= \left[\cos\left(\frac{\phi}{2}\right) - \boldsymbol{v}\sin\left(\frac{\phi}{2}\right)\right]\left[\cos\left(\frac{\phi}{2}\right) + \boldsymbol{v}\sin\left(\frac{\phi}{2}\right)\right] \\
&= \cos^2\left(\frac{\phi}{2}\right) - \boldsymbol{v}\boldsymbol{v}\sin^2\left(\frac{\phi}{2}\right) \\
&= \cos^2\left(\frac{\phi}{2}\right) - (-1)\sin^2\left(\frac{\phi}{2}\right) \\
&= \cos^2\left(\frac{\phi}{2}\right) + \sin^2\left(\frac{\phi}{2}\right) \\
&= 1.
\end{aligned}$$

## 30.4 Connecting these ideas

For the purposes of this section, we want to connect a `translation` (in the code) to a *translation* (from the equation) and a `rotation` (in the code) to a *rotation* (from the equation).

For a `translation`, we have the following.

- `x` will store the value of the term related to $\hat{\imath}$ in $\boldsymbol{t}$, that is $t_x$.
- `y` will store the value of the term is related to $\hat{\jmath}$ in $\boldsymbol{t}$, that is $t_y$.
- `z` will store the value of the term is related to $\hat{k}$ in $\boldsymbol{t}$, that is $t_z$.

**Example**

To represent the following translation,

$$\boldsymbol{t}_1 \triangleq 1\hat{\imath} + 2\hat{\jmath} + 3\hat{k},$$

we would assign `translation.x = 1`, `translation.y = 2`, and `translation.z = 3` in our program.

Similarly, for a `rotation`, we have the following.

- `x` will store the value of the term related to $\hat{\imath}$ in $\boldsymbol{r}$.
- `y` will store the value of the term is related to $\hat{\jmath}$ in $\boldsymbol{r}$.
- `z` will store the value of the term is related to $\hat{k}$ in $\boldsymbol{r}$.

Lastly, we have, for a `rotation`,

- `w` will store the value of the term not related to any imaginary unit.

**Note**

You might be wondering why `w`, when the other dimensions might be already natural to you. We need to give it a name in our programs so that we can store its data. In alphabetical order, `w` comes before the letters we usually use for the x, y, and z dimensions.

**Example**

To represent the following elementary rotation about the x-axis

$$\boldsymbol{r}_1 \triangleq \cos\left(\frac{\pi}{2}\right) + \hat{\imath}\sin\left(\frac{\pi}{2}\right),$$

we would assign `rotation.w = cos(pi/2)`, `rotation.x = sin(pi/2)`, `rotation.y = 0`, and `rotation.z = 0` in our program.

## 30.5 Create the package

We will create a package to showcase the transformations from the previous section. We use `TransformedStamped` as it will be useful right away, when we talk about `tf2`.

To see how this would work, programmatically, we start by creating the `python_package_that_uses_geometry_msgs` package. Note that it must depend on `geometry_msgs`.

```
cd ~/ros2_tutorial_workspace/src
ros2 pkg create python_package_that_uses_geometry_msgs \
--build-type ament_python \
--dependencies geometry_msgs
```

### ros2 pkg create output

```
going to create a new package
package name: python_package_that_uses_geometry_msgs
destination directory: ~/ros2_tutorial_workspace/src
package format: 3
version: 0.0.0
description: TODO: Package description
maintainer: ['root <murilo.marinho@manchester.ac.uk>']
licenses: ['TODO: License declaration']
build type: ament_python
dependencies: ['geometry_msgs']
creating folder ./python_package_that_uses_geometry_msgs
creating ./python_package_that_uses_geometry_msgs/package.xml
creating source folder
creating folder ./python_package_that_uses_geometry_msgs/python_package_that_uses_
↪geometry_msgs
creating ./python_package_that_uses_geometry_msgs/setup.py
creating ./python_package_that_uses_geometry_msgs/setup.cfg
creating folder ./python_package_that_uses_geometry_msgs/resource
creating ./python_package_that_uses_geometry_msgs/resource/python_package_that_uses_
↪geometry_msgs
creating ./python_package_that_uses_geometry_msgs/python_package_that_uses_geometry_msgs/
↪__init__.py
creating folder ./python_package_that_uses_geometry_msgs/test
creating ./python_package_that_uses_geometry_msgs/test/test_copyright.py
creating ./python_package_that_uses_geometry_msgs/test/test_flake8.py
creating ./python_package_that_uses_geometry_msgs/test/test_pep257.py

[WARNING]: Unknown license 'TODO: License declaration'.  This has been set in the␣
```

(continues on next page)

(continued from previous page)

```
↪package.xml, but no LICENSE file has been created.
It is recommended to use one of the ament license identifiers:
Apache-2.0
BSL-1.0
BSD-2.0
BSD-2-Clause
BSD-3-Clause
GPL-3.0-only
LGPL-3.0-only
MIT
MIT-0
```

## 30.6 Package structure

Highlighted below are the files that we will create or modify.

```
python_package_that_uses_geometry_msgs/
|-- package.xml
|-- python_package_that_uses_geometry_msgs
|   |-- __init__.py
|   `-- create_stamped_transforms_node.py
|-- resource
|   `-- python_package_that_uses_geometry_msgs
|-- setup.cfg
|-- setup.py
`-- test
    |-- test_copyright.py
    |-- test_flake8.py
    `-- test_pep257.py
```

## 30.7 Add sample code

Create the following sample Python script. It will serve to show the operations we did earlier mathematically.

`create_stamped_transforms_node.py`

```
from math import cos, sin, pi
from geometry_msgs.msg import TransformStamped

import rclpy
from rclpy.node import Node

class CreateStampedTransformsNode(Node):
    """A ROS2 Node that creates TransformStamped objects."""

    def __init__(self):
        super().__init__('create_stamped_transforms_node')

    def create_and_print_stamped_transform(self):
        """Basic method showing the creation of TransformStamped objects."""

```

(continues on next page)

(continued from previous page)

```
        tfs = TransformStamped()

        # Initialize header with
        ## Timestamp equal to the current clock time
        tfs.header.stamp = self.get_clock().now().to_msg()
        ## Frame of reference equals `world`
        tfs.header.frame_id = 'world'
        ## The transform will be of the frame will be of this tag
        tfs.child_frame_id = "object_1"

        # Set the translation of the transform as a circle in the x-y plane
        tfs.transform.translation.x = 1.0
        tfs.transform.translation.y = 2.0
        tfs.transform.translation.z = 3.0

        # Set the rotation (Quaternion) of the transform as a rotation about the x-axis
        tfs.transform.rotation.w = cos(pi / 2.0)
        tfs.transform.rotation.x = sin(pi / 2.0)
        tfs.transform.rotation.y = 0.0
        tfs.transform.rotation.z = 0.0

        self.get_logger().info(f"This transform has translation:"
                               f" {tfs.transform.translation} "
                               f"and rotation:"
                               f" {tfs.transform.rotation}.")

def main(args=None):
    """
    The main function.
    :param args: Not used directly by the user, but used by ROS2 to configure certain␣
↪aspects of the Node.
    """
    try:
        rclpy.init(args=args)

        node = CreateStampedTransformsNode()
        node.create_and_print_stamped_transform()

    except KeyboardInterrupt:
        pass
    except Exception as e:
        print(e)



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

Note that the only novelty will be the excerpt below. We create an instance of `TransformStamped` and start by adding information related to the header. Each transform has a frame of reference, called `header.frame_id`. Then, the actual frame this transform represents is held by `child_frame_id`.

The values corresponding to example translation $\boldsymbol{t}_1 \triangleq 1\hat{\imath} + 2\hat{\jmath} + 3\hat{k}$ and the example rotation $\boldsymbol{r}_1 \triangleq \cos\left(\frac{\pi}{2}\right) + \hat{\imath}\sin\left(\frac{\pi}{2}\right)$

are assigned in the highlight lines below.

```
    def create_and_print_stamped_transform(self):
        """Basic method showing the creation of TransformStamped objects."""

        tfs = TransformStamped()

        # Initialize header with
        ## Timestamp equal to the current clock time
        tfs.header.stamp = self.get_clock().now().to_msg()
        ## Frame of reference equals `world`
        tfs.header.frame_id = 'world'
        ## The transform will be of the frame will be of this tag
        tfs.child_frame_id = "object_1"

        # Set the translation of the transform as a circle in the x-y plane
        tfs.transform.translation.x = 1.0
        tfs.transform.translation.y = 2.0
        tfs.transform.translation.z = 3.0

        # Set the rotation (Quaternion) of the transform as a rotation about the x-axis
        tfs.transform.rotation.w = cos(pi / 2.0)
        tfs.transform.rotation.x = sin(pi / 2.0)
        tfs.transform.rotation.y = 0.0
        tfs.transform.rotation.z = 0.0

        self.get_logger().info(f"This transform has translation:"
                               f" {tfs.transform.translation} "
                               f"and rotation:"
                               f" {tfs.transform.rotation}.")
```

## 30.8 Update the `setup.py`

As usual, we add the necessary entry point in `setup.py`.

`setup.py`

```
from setuptools import find_packages, setup

package_name = 'python_package_that_uses_geometry_msgs'

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']),
    ],
    install_requires=['setuptools'],
    zip_safe=True,
    maintainer='root',
```

(continues on next page)

(continued from previous page)

```
    maintainer_email='murilo.marinho@manchester.ac.uk',
    description='TODO: Package description',
    license='TODO: License declaration',
    tests_require=['pytest'],
    entry_points={
        'console_scripts': [
            "create_stamped_transforms_node = python_package_that_uses_geometry_msgs.
↪create_stamped_transforms_node:main",
        ],
    },
)
```

## 30.9 Build and source

Before we proceed, let us build and source once.

```
cd ~/ros2_tutorial_workspace
colcon build
source install/setup.bash
```

**Note**

For additional explanation and troubleshooting tips, see *Always source after you build*.

**Warning**

**colcon** will *not* work properly if your terminal has an active **venv**.

## 30.10 Run Example

We run our newly created program as follows.

```
ros2 run python_package_that_uses_geometry_msgs create_stamped_transforms_node
```

The result will be as follows.

```
This transform has translation: geometry_msgs.msg.Vector3(x=1.0, y=2.0, z=3.0) and␣
↪rotation: geometry_msgs.msg.Quaternion(x=1.0, y=0.0, z=0.0, w=6.123233995736766e-17).
```

Note that the results are reasonably close to the ones we calculated mathematically. However, given the limitations on current computers related to floating point accuracy, you can always expect a level of inaccuracy. This is not limited or affected by the use of quaternions, this is an inherent limitation of our computers.

CHAPTER

# THIRTYONE

# USING TF2

**Warning**

This topic is under construction and this might not even be its final form. Please feel free to open an issue if you spot any typos or other problems.

**See also**

An official `tf2` user guide is available at https://docs.ros.org/en/jazzy/Tutorials/Intermediate/Tf2/Introduction-To-Tf2.html.

Perhaps the most important benefit of `tf2` is how it connects with other **ROS2** parts, such as `rviz2`. It is a common language used across well-established **ROS2** packages.

The usage is not too far from what you have learned so far, but there a few aspects that differ.

## 31.1 Create the package

**Note**

The package is called `tf2_ros` in **ROS2**.

We start by creating the `python_package_that_uses_tf2` package. Please note that it must depend on `tf2_ros`.

```
cd ~/ros2_tutorial_workspace/src
ros2 pkg create python_package_that_uses_tf2 \
--build-type ament_python \
--dependencies geometry_msgs tf2_ros
```

### ros2 pkg create output

```
going to create a new package
package name: python_package_that_uses_tf2
destination directory: ~/ros2_tutorial_workspace/src
package format: 3
version: 0.0.0
```

(continues on next page)

(continued from previous page)

```
description: TODO: Package description
maintainer: ['root <murilo.marinho@manchester.ac.uk>']
licenses: ['TODO: License declaration']
build type: ament_python
dependencies: ['geometry_msgs', 'tf2_ros']
creating folder ./python_package_that_uses_tf2
creating ./python_package_that_uses_tf2/package.xml
creating source folder
creating folder ./python_package_that_uses_tf2/python_package_that_uses_tf2
creating ./python_package_that_uses_tf2/setup.py
creating ./python_package_that_uses_tf2/setup.cfg
creating folder ./python_package_that_uses_tf2/resource
creating ./python_package_that_uses_tf2/resource/python_package_that_uses_tf2
creating ./python_package_that_uses_tf2/python_package_that_uses_tf2/__init__.py
creating folder ./python_package_that_uses_tf2/test
creating ./python_package_that_uses_tf2/test/test_copyright.py
creating ./python_package_that_uses_tf2/test/test_flake8.py
creating ./python_package_that_uses_tf2/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
BSL-1.0
BSD-2.0
BSD-2-Clause
BSD-3-Clause
GPL-3.0-only
LGPL-3.0-only
MIT
MIT-0
```

## 31.2 Package structure

In this section, we will create or modify the following files.

```
python_package_that_uses_tf2/
|-- package.xml
|-- python_package_that_uses_tf2
|   |-- __init__.py
|   |-- tf2_broadcaster_node.py
|   `-- tf2_listener_node.py
|-- resource
|   `-- python_package_that_uses_tf2
|-- setup.cfg
|-- setup.py
`-- test
    |-- test_copyright.py
    |-- test_flake8.py
    `-- test_pep257.py
```

## 31.3 Creating the broadcaster

The broadcaster is made with the following piece of code.

tf2_broadcaster_node.py

```python
from math import sin, cos, pi
from geometry_msgs.msg import TransformStamped

import rclpy
from rclpy.node import Node

import tf2_ros

class TF2BroadcasterNode(Node):
    """A ROS2 Node that broadcasts a TransformStamped in compliance with `tf2_ros2`."""

    def __init__(self):
        super().__init__('tf2_broadcaster_node')

        # Robot name
        self.robot_name = "robot_1"

        # Initialize the transform broadcaster
        ## Note that this object is part of `tf2_ros`, not `rclpy`
        self.transform_broadcaster = tf2_ros.TransformBroadcaster(self)

        self.timer_period: float = 0.01
        self.timer_elapsed_time: float = 0
        self.timer = self.create_timer(self.timer_period, self.timer_callback)

    def timer_callback(self):
        # We define some parameters of our trajectory.
        # Let's make the translation be a circle in 2D with radius of 0.2 meters and␣
↪frequency of 0.1 Hz.
        # The rotation will be the robot spinning about the z-axis with the same␣
↪frequency.
        trajectory_radius = 0.2
        trajectory_frequency = 0.1

        # Create an instance of a TransformStamped, used by the broadcaster
        tfs = TransformStamped()

        ## Initialize header with
        # Timestamp equal to the current clock time
        tfs.header.stamp = self.get_clock().now().to_msg()
        # Frame of reference
        tfs.header.frame_id = 'world'
        # The object whose transform this message is about
        tfs.child_frame_id = self.robot_name

        # Set the translation of the transform as a circle in the x-y plane
        tfs.transform.translation.x = trajectory_radius * cos(2 * pi * trajectory_
↪frequency * self.timer_elapsed_time)
```

(continues on next page)

(continued from previous page)

```
        tfs.transform.translation.y = trajectory_radius * sin(2 * pi * trajectory_
↪frequency * self.timer_elapsed_time)
        tfs.transform.translation.z = 0.0

        # Set the rotation (Quaternion) of the transform as a rotation about the z-axis
        phi: float = 2.0 * pi * trajectory_frequency * self.timer_elapsed_time
        tfs.transform.rotation.w = cos(phi/2.0)
        tfs.transform.rotation.x = 0.0
        tfs.transform.rotation.y = 0.0
        tfs.transform.rotation.z = sin(phi/2.0)

        # Send the transformation
        self.transform_broadcaster.sendTransform(tfs)

        # Update internal time counter
        self.timer_elapsed_time += self.timer_period


def main(args=None):
    """
    The main function.
    :param args: Not used directly by the user, but used by ROS2 to configure certain␣
↪aspects of the Node.
    """
    try:
        rclpy.init(args=args)

        node = TF2BroadcasterNode()

        rclpy.spin(node)
    except KeyboardInterrupt:
        pass
    except Exception as e:
        print(e)

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

Most of this should be familiar by now.

Focusing on the main novelty, we create the broadcaster with `tf2_ros.TransformBroadcaster`. We need to input a `rclpy.node.Node` for the constructor, so we input it as `self`.

As minor things, we have a tag for this frame as `robot_name`. This could, for example, be a **`ros2`**. We leave it hard-coded for simplicity. We also keep time in the object's scope with `timer_elapsed_time` so that it persists across callback calls.

```
    def __init__(self):
        super().__init__('tf2_broadcaster_node')

        # Robot name
        self.robot_name = "robot_1"

```

(continues on next page)

(continued from previous page)

```
        # Initialize the transform broadcaster
        ## Note that this object is part of `tf2_ros`, not `rclpy`
        self.transform_broadcaster = tf2_ros.TransformBroadcaster(self)

        self.timer_period: float = 0.01
        self.timer_elapsed_time: float = 0
        self.timer = self.create_timer(self.timer_period, self.timer_callback)
```

We add two parameters to be able to adjust the trajectory properties if desired. Again, these could be configurable parameters but we leave them hard-coded for simplicity.

```
        # We define some parameters of our trajectory.
        # Let's make the translation be a circle in 2D with radius of 0.2 meters and␣
↪frequency of 0.1 Hz.
        # The rotation will be the robot spinning about the z-axis with the same␣
↪frequency.
        trajectory_radius = 0.2
        trajectory_frequency = 0.1
```

We create the `TransformStamped` as follows. Our rotation is simple so we build Quaternion for the rotation directly. We set the reference frame, `header.frame_id`, as `world` and `child_frame_id` as this robot's frame.

```
        tfs = TransformStamped()

        ## Initialize header with
        # Timestamp equal to the current clock time
        tfs.header.stamp = self.get_clock().now().to_msg()
        # Frame of reference
        tfs.header.frame_id = 'world'
        # The object whose transform this message is about
        tfs.child_frame_id = self.robot_name

        # Set the translation of the transform as a circle in the x-y plane
        tfs.transform.translation.x = trajectory_radius * cos(2 * pi * trajectory_
↪frequency * self.timer_elapsed_time)
        tfs.transform.translation.y = trajectory_radius * sin(2 * pi * trajectory_
↪frequency * self.timer_elapsed_time)
        tfs.transform.translation.z = 0.0

        # Set the rotation (Quaternion) of the transform as a rotation about the z-axis
        phi: float = 2.0 * pi * trajectory_frequency * self.timer_elapsed_time
        tfs.transform.rotation.w = cos(phi/2.0)
        tfs.transform.rotation.x = 0.0
        tfs.transform.rotation.y = 0.0
        tfs.transform.rotation.z = sin(phi/2.0)
```

Finally, we send the transform with `transform_broadcaster.sendTransform` and update the local time counter for the trajectory.

```
        # Send the transformation
        self.transform_broadcaster.sendTransform(tfs)
```

(continues on next page)

(continued from previous page)

```
        # Update internal time counter
        self.timer_elapsed_time += self.timer_period
```

## 31.4 Creating the listener

The listener is made with the following piece of code.

tf2_listener_node.py

```
import rclpy
from rclpy.node import Node

import tf2_ros

class TF2ListenerNode(Node):
    """A ROS2 Node that listens to the `tf` topic in compliance with `tf2_ros2`."""

    def __init__(self):
        super().__init__('tf2_listener_node')

        # Setting up the TransformListener
        self.transform_listener_buffer = tf2_ros.Buffer()
        self.transform_listener = tf2_ros.TransformListener(self.transform_listener_
↪buffer, self)

        # Information about the transform we want to listen to
        self.parent_name = "world"
        self.child_name = "robot_1"

        self.timer_period: float = 0.1
        self.timer = self.create_timer(self.timer_period, self.timer_callback)

    def timer_callback(self):

        try:
            tfs =   self.transform_listener_buffer.lookup_transform(
                    self.parent_name,
                    self.child_name,
                    rclpy.time.Time())

            self.get_logger().info(f"Transform: {tfs}")

        except tf2_ros.TransformException as e:

            self.get_logger().error(
                f'Could not get transform from `{self.parent_name}` to `{self.child_name}
↪`: {e}')


def main(args=None):
    """
    The main function.
```

(continues on next page)

(continued from previous page)

```
    :param args: Not used directly by the user, but used by ROS2 to configure certain␣
↪aspects of the Node.
    """
    try:
        rclpy.init(args=args)

        node = TF2ListenerNode()

        rclpy.spin(node)
    except KeyboardInterrupt:
        pass
    except Exception as e:
        print(e)

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

We use an instance of `tf2_ros.TransformListener` to manage the interaction with `tf2` for us. The only step that might be unfamiliar is that you need to create an instance to a buffer which is used in the initializer of `TransformListener`.

We also add the frame information that we want to listen to. This will be used in another step. One extension of this would be to make this configurable as a parameter. We do not do that here to not increase complexity.

```
        # Setting up the TransformListener
        self.transform_listener_buffer = tf2_ros.Buffer()
        self.transform_listener = tf2_ros.TransformListener(self.transform_listener_
↪buffer, self)

        # Information about the transform we want to listen to
        self.parent_name = "world"
        self.child_name = "robot_1"
```

We then create an instance of `Timer` and add the callback. In the callback, we use the buffer we created in the previous step. We will call `lookup_transform` and it will need the parent frame tag, the child name tag, and the time of lookup. We add exception handling in case the transform is not available or not available in the time requested.

The object created with `rclpy.time.Time()` is equivalent to a time of zero, and `lookup_transform` returns the latest transformation available.

```
    def timer_callback(self):

        try:
            tfs =   self.transform_listener_buffer.lookup_transform(
                    self.parent_name,
                    self.child_name,
                    rclpy.time.Time())

            self.get_logger().info(f"Transform: {tfs}")

        except tf2_ros.TransformException as e:

            self.get_logger().error(
```

(continues on next page)

(continued from previous page)

```
                f'Could not get transform from `{self.parent_name}` to `{self.child_name}
↪`: {e}')
```

## 31.5 The `setup.py`

We add the usual entry points.

`setup.py`

```
from setuptools import find_packages, setup

package_name = 'python_package_that_uses_tf2'

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']),
    ],
    install_requires=['setuptools'],
    zip_safe=True,
    maintainer='root',
    maintainer_email='murilo.marinho@manchester.ac.uk',
    description='TODO: Package description',
    license='TODO: License declaration',
    extras_require={
        'test': [
            'pytest',
        ],
    },
    entry_points={
        'console_scripts': [
            "tf2_broadcaster_node = python_package_that_uses_tf2.tf2_broadcaster_
↪node:main",
            "tf2_listener_node = python_package_that_uses_tf2.tf2_listener_node:main"
        ],
    },
)
```

## 31.6 Build and source

Before we proceed, let us build and source once.

```
cd ~/ros2_tutorial_workspace
colcon build
source install/setup.bash
```

**Note**

For additional explanation and troubleshooting tips, see *Always source after you build*.

**Warning**

**colcon** will *not* work properly if your terminal has an active **venv**.

## 31.7 Run Example

We can show the integrated example as follows.

### Terminal 1: tf2 broadcaster

We start by running the broadcaster.

```
ros2 run python_package_that_uses_tf2 tf2_broadcaster_node
```

### Terminal 2: tf2 listener

Then, we run the listener.

```
ros2 run python_package_that_uses_tf2 tf2_listener_node
```

Our broadcaster won't output anything to the screen.

The output of the listener will be as long as you allow it to be. The first line indicates a normal behavior, in which subscribers to not have instant access to topics. That is properly handled by our listener. When available, you can see the output.

```
[ERROR] [1762112353.248708886] [tf2_listener_node]: Could not transform world to robot_
↪1: "world" passed to lookupTransform argument target_frame does not exist.
[INFO] [1762112353.340386011] [tf2_listener_node]: Transform: geometry_msgs.msg.
↪TransformStamped(header=std_msgs.msg.Header(stamp=builtin_interfaces.msg.
↪Time(sec=1762112353, nanosec=335937928), frame_id='world'), child_frame_id='robot_1',␣
↪transform=geometry_msgs.msg.Transform(translation=geometry_msgs.msg.Vector3(x=0.
↪1144842511890406, y=-0.16399042186510032, z=0.0), rotation=geometry_msgs.msg.
↪Quaternion(x=0.0, y=0.0, z=-0.46236775104102995, w=0.8866882557005366)))
[INFO] [1762112353.440609636] [tf2_listener_node]: Transform: geometry_msgs.msg.
↪TransformStamped(header=std_msgs.msg.Header(stamp=builtin_interfaces.msg.
↪Time(sec=1762112353, nanosec=436304803), frame_id='world'), child_frame_id='robot_1',␣
↪transform=geometry_msgs.msg.Transform(translation=geometry_msgs.msg.Vector3(x=0.
↪12455755609760884, y=-0.1564781621153285, z=0.0), rotation=geometry_msgs.msg.
↪Quaternion(x=0.0, y=0.0, z=-0.43428804928984416, w=0.9007740506053791)))
```

For now, you have to believe I'm right and that this means the broadcaster is sending a circular trajectory. We will see more about this in visualization and simulation.

## 31.8 What is happening?

The broadcaster and listener are communicating through a **ROS2** topic. We can check that with the following command. Please ensure that both nodes are running before checking this.

```
ros2 topic list
```

This should output the following.

```
/parameter_events
/rosout
/tf
```

We can obtain more information about the `/tf` topic with the following command.

```
ros2 topic info /tf
```

The output of the command should be as follows.

```
Type: tf2_msgs/msg/TFMessage
Publisher count: 1
Subscription count: 1
```

You might be asking yourself the following.

> What nonsense is this? Wasn't it supposed to be a `geometry_msgs/msg/TransformStamped`?? I also didn't define no topic `/tf`?????

You will notice that although `tf2` uses topics, publishers, and subscribers, it has another layer of complexity. For instance, most of the time we can expect many frames (think of dozens or hundreds) to be published in the same topic, making a regular subscriber not as useful and too busy.

The broadcaster and listener, although using a `geometry_msgs/msg/TransformStamped` in our nodes, will convert back-and-forth between `tf2_msgs/msg/TFMessage` and `geometry_msgs/msg/TransformStamped` when interacting with the topic `/tf`. Also, because of that, you can think of the topic `/tf` as a *protected* topic. No such concept of *protected* topics seems to exist in **ROS2**, so that is not enforced, you can publish and subscribe normally to that topic. Basically, in this context, I mean to say that if you attempt to use `/tf` without using `tf2_ros` facilities, you are likely to be disappointed.

Either way, there is no reason to fear. You will notice that `tf2_msgs/msg/TFMessage` is simply an array of `geometry_msgs/msg/TransformStamped`.

You can see that with the following command.

```
ros2 interface show tf2_msgs/msg/TFMessage
```

This should result in the following, where the important line is highlighted.

```
geometry_msgs/TransformStamped[] transforms
        #
        #
        std_msgs/Header header
                builtin_interfaces/Time stamp
                        int32 sec
                        uint32 nanosec
                string frame_id
        string child_frame_id
```

(continues on next page)

(continued from previous page)

```
        Transform transform
                Vector3 translation
                        float64 x
                        float64 y
                        float64 z
                Quaternion rotation
                        float64 x 0
                        float64 y 0
                        float64 z 0
                        float64 w 1
```

There will be a degree of pointlessness when attempting to parse through the `/tf` output in the terminal. We will also not attempt to do that here. To help you with that, we will have visualisation and simulation tools shown in following sections.

CHAPTER

# THIRTYTWO

# RVIZ2 FOR TF2

**Warning**

This topic is under construction and this might not even be its final form. Please feel free to open an issue if you spot any typos or other problems.

**See also**

An official **rviz2** user guide is available at: https://docs.ros.org/en/jazzy/Tutorials/Intermediate/RViz/RViz-User-Guide/RViz-User-Guide.html.

## 32.1 rviz2 Installation

**rviz2** has already been installed in *ROS2 Installation*.

## 32.2 What is rviz2?

Possibly the easiest description for **rviz2**, available from https://github.com/ros2/rviz, is *ROS 3D Robot Visualizer*.

Given that **rviz2** lacks any integration with simulation engines, it wouldn't be usually called a robotics simulator. Nonetheless, for most robotics applications in their initial stages of development, a visualization is an important step forward.

## 32.3 Using rviz2 with tf2

One interesting capability of **rviz2** is the visualization of `tf2` in a relatively friendly manner. We can test that quickly with the `tf2` example we developed in the previous section.

Up until now, I showered you with equations that you might or might not have understood. After showing you how it works in **rviz2**, hopefully you will begin to believe me.

### Terminal 1: tf2 broadcaster

From our previous example of `tf2`.

```
ros2 run python_package_that_uses_tf2 tf2_broadcaster_node
```

### Terminal 2: rviz2

We run **rviz2** with the following command.

```
ros2 run rviz2 rviz2
```

The user interface of **rviz2** should appear. However, it still does not know what you are trying to do with it, so we will make two modifications to be able to visualize the output of our `tf2_broadcaster_node`.

1. In *Displays* → *Fixed Frame*, we change the text from *map* to *world*.
2. Then, we add the `tf2` visualisation with the sequence *Add* → *rviz_default_plugins* → *TF*.

**Note**

Starting **rviz2** with the following command allows you to skip the first step.

```
ros2 run rviz2 rviz2 -f world
```

This is summarised in the video below.

This should be enough to start showing the frames. Using the mouse, can you zoom in and rotate the window to focus on the transforms of interest.

If everything worked as expected, you should see a `robot_1` frame rotating about a `world` frame. Now, you can easily visualize the frames indicating that the frame marked by `robot_1` indeed moves in circles, with the rotational frame also rotating in the same frequency.

With these settings, additional broadcast transforms will also be visible. Using **rviz2** menus, you can filter the transforms that are relevant for you at any given point.

## 32.4 Other visualisation tools

Depending on the sensorial information, **rviz2** can be the correct tool. It is important, however, to know that it is not the only tool. One convenient tool for visualizing images is **rqt_image_view**.

The integrated sample can be executed as follows.

**Danger**

Please note that the (Lena or Lenna) image has been banned by most publishers. Do not use it in your reports or conference submissions. This is part of the integrated sample, therefore shown here, but this usage is not endorsed or recommended.

- https://www.nature.com/articles/s41565-018-0337-2
- https://journals.ieeeauthorcenter.ieee.org/create-your-ieee-journal-article/create-graphics-for-your-article/

Nature:

> We would like to let our authors, reviewers and readers know that, with immediate effect, we no longer consider submissions containing the Lena (sometimes 'Lenna') image. This decision was taken in consultation with relevant journal editors and affects all Nature Research journals.

IEEE:

> Lena Image

> IEEE's diversity statement and supporting policies such as the IEEE Code of Ethics speak to IEEE's commitment to promoting an inclusive and equitable culture that welcomes all. In alignment with this culture and with respect to the wishes of the subject of the image, Lena Forsén, IEEE will no longer accept submitted papers which include the "Lena image."

**Terminal 1: Publish sample images.**

```
ros2 run rqt_image_view image_publisher
```

**Terminal 2: Run the bridge**

```
ros2 run rqt_image_view rqt_image_view /images
```

It should work by showing color variations of the controversial image. We will use the visualiser, not the publisher, in some examples in future sections.

CHAPTER

# THIRTYTHREE

# THE SUPPORT LIBRARY `NOTTF2`

**Warning**

This topic is under construction and this might not even be its final form. Please feel free to open an issue if you spot any typos or other problems.

**Danger**

You do not need this support library if you're just creating basic rotations. You can create easy rotations directly using `math.cos`, `math.sin`, and the class `Quaternion` part of `geometry_msgs`. The support library becomes useful if you need to, for instance, multiply quaternions to obtain relative rotations with `quaternion_multiply`.

In `rclpy`, `tf2` does not (currently?) have quaternion operations. But I got you covered. Kinda. It's more use at your own risk type of arrangement.

For the library to be properly found in our environment, it needs to be installed in the system's Python. We can do so as follows.

```
python3 -m pip install nottf2 --break-system-packages
```

**Caution**

The `nottf2` library is provided as-is, with no warranty. Use at your own risk (See `MIT` license for more details).

The creator of `nottf2`, me, will not accept any blame or liability for any issues, including but limited to broken robots or broken code in your assignments. If you feel it's insufficient or unreliable, please create and use your own implementation, or someone else's that fits your needs.

CHAPTER

# THIRTYFOUR

# CREATE ROS2 PACKAGE THAT USES NOTTF2

Let's first create our sample package, as follows, that will depend on `geometry_msgs`.

```
cd ~/ros2_tutorial_workspace/src
ros2 pkg create python_package_that_uses_nottf2 \
--build-type ament_python \
--dependencies geometry_msgs
```

We will be presented with the usual output.

### ros2 pkg create output

```
going to create a new package
package name: python_package_that_uses_nottf2
destination directory: ~/ros2_tutorial_workspace/src
package format: 3
version: 0.0.0
description: TODO: Package description
maintainer: ['root <murilo.marinho@manchester.ac.uk>']
licenses: ['TODO: License declaration']
build type: ament_python
dependencies: []
creating folder ./python_package_that_uses_nottf2
creating ./python_package_that_uses_nottf2/package.xml
creating source folder
creating folder ./python_package_that_uses_nottf2/python_package_that_uses_nottf2
creating ./python_package_that_uses_nottf2/setup.py
creating ./python_package_that_uses_nottf2/setup.cfg
creating folder ./python_package_that_uses_nottf2/resource
creating ./python_package_that_uses_nottf2/resource/python_package_that_uses_nottf2
creating ./python_package_that_uses_nottf2/python_package_that_uses_nottf2/__init__.py
creating folder ./python_package_that_uses_nottf2/test
creating ./python_package_that_uses_nottf2/test/test_copyright.py
creating ./python_package_that_uses_nottf2/test/test_flake8.py
creating ./python_package_that_uses_nottf2/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
BSL-1.0
```

(continues on next page)

(continued from previous page)

```
BSD-2.0
BSD-2-Clause
BSD-3-Clause
GPL-3.0-only
LGPL-3.0-only
MIT
MIT-0
```

CHAPTER

# THIRTYFIVE

# PACKAGE STRUCTURE

Highlighted below are the files that we will create or modify.

```
python_package_that_uses_nottf2/
|-- package.xml
|-- python_package_that_uses_nottf2
|   |-- __init__.py
|   `-- operations_showcase.py
|-- resource
|   `-- python_package_that_uses_nottf2
|-- setup.cfg
|-- setup.py
`-- test
    |-- test_copyright.py
    |-- test_flake8.py
    `-- test_pep257.py
```

CHAPTER

# THIRTYSIX

## ADD SAMPLE CODE FOR NOTTF2

Create the following sample Python script. It will serve to show the operations we did earlier mathematically.

operations_showcase.py

```python
from math import pi
from geometry_msgs.msg import Quaternion
import marinholab.nottf2 as ntf2

def main():
    phi: float = pi

    r1: Quaternion = ntf2.rx(phi=phi)
    print(f"The rotation of {phi} radians about the x-axis is r1={r1}.")

    r1_conj : Quaternion = ntf2.rotation_inverse(r1)
    print(f"The inverse rotation of r1 is r1_conj={r1_conj}.")

    r2: Quaternion = ntf2.rz(phi=phi)
    print(f"The rotation of {phi} radians about the z-axis is r2={r2}.")

    r12: Quaternion = ntf2.quaternion_multiply(r1, r2)
    print(f"The quaternion multiplication of r12=r1r2 is r12={r12}.")

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

CHAPTER

# THIRTYSEVEN

## UPDATE THE SETUP.PY

As usual, we add the necessary entry point in `setup.py`. We also add `nottf2` as an usual Python dependency, although, in **colcon**, this currently seems to be mostly cosmetic.

setup.py

```
from rosdistro.cli.rosdistro import depends
from setuptools import find_packages, setup

package_name = 'python_package_that_uses_nottf2'

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']),
    ],
    install_requires=['setuptools', 'nottf2'],
    zip_safe=True,
    maintainer='root',
    maintainer_email='murilo.marinho@manchester.ac.uk',
    description='TODO: Package description',
    license='TODO: License declaration',
    tests_require=['pytest'],
    entry_points={
        'console_scripts': [
            'operations_showcase = python_package_that_uses_nottf2.operations_
↪showcase:main',
        ],
    },
)
```

CHAPTER

# THIRTYEIGHT

# BUILD AND SOURCE

Before we proceed, let us build and source once.

```
cd ~/ros2_tutorial_workspace
colcon build
source install/setup.bash
```

**Note**

For additional explanation and troubleshooting tips, see *Always source after you build*.

**Warning**

**`colcon`** will *not* work properly if your terminal has an active **`venv`**.

CHAPTER

# THIRTYNINE

# RUN EXAMPLE

We run our newly created program as follows.

```
ros2 run python_package_that_uses_nottf2 operations_showcase
```

The result will be as follows.

```
The rotation of 3.141592653589793 radians about the x-axis is r1=geometry_msgs.msg.
↪Quaternion(x=1.0, y=0, z=0, w=6.123233995736766e-17).
The inverse rotation of r1 is r1_conj=geometry_msgs.msg.Quaternion(x=-1.0, y=0, z=0, w=6.
↪123233995736766e-17).
The rotation of 3.141592653589793 radians about the z-axis is r2=geometry_msgs.msg.
↪Quaternion(x=0, y=0, z=1.0, w=6.123233995736766e-17).
The quaternion multiplication of r12=r1r2 is r12=geometry_msgs.msg.Quaternion(x=6.
↪123233995736766e-17, y=-1.0, z=6.123233995736766e-17, w=3.749399456654644e-33).
```

Note that the results are reasonably close to the ones we calculated mathematically. However, given the limitations on current computers related to floating point accuracy, you can **always** expect a level of inaccuracy. This is **not** limited or affected by the use of quaternions, this is an inherent limitation of our computers.

CHAPTER

FORTY

# ABOUT GAZEBO

Added in version Jazzy: This section.

**Warning**

This topic is under construction and this might not even be its final form. Please feel free to open an issue if you spot any typos or other problems.

**See also**

Click here for the official docs from the developers of **Gazebo**.

**Gazebo** is a robot simulator frequently mentioned when **ROS2** is used. They are officially different projects although there are clear connections.

Until **ROS2 Humble**, **Gazebo** worked in a different way and had a different name. The older project has been renamed to **Gazebo Classic**. You can see more about the reasons behind this in the announcement "A new era for Gazebo".

**Warning**

This means that you will find a large amount of legacy code online, for instance with the prefix `ign`. This clearly won't work for **Gazebo** Harmonic without proper adjustments.

Given that this new way to use **Gazebo** has started in **ROS2 Jazzy**, we can expect that some edges will be rough and some functionalities missing. Nonetheless, it is reasonable to believe this will be the way to operate **Gazebo** in the foreseeable future.

For **ROS2 Jazzy**, we use **Gazebo Harmonic**. This replicates the strategy of having a LTS version for a given piece of software. This makes it easier for the users to learn and trust a platform without it constantly changing.

Beauty and usability are in the eyes of the beholder. That said, one major improvement in **Gazebo** is the motion towards not needing two robot description formats. It has always been the case that a robot would have to be described twice, once for **ROS2** and another for **Gazebo Classic**. Although this switch might still be ongoing and some more advanced functionalities might be missing, we will work with a single `.sdf` whenever possible.

If some of these terms are not clear to you now, worry not. They will become clearer as we progress in the tutorials.

CHAPTER

# FORTYONE

# GAZEBO INSTALLATION

Added in version Jazzy: This section.

**Warning**

This topic is under construction and this might not even be its final form. Please feel free to open an issue if you spot any typos or other problems.

The following command will install **Gazebo Harmonic** and all the pairing libraries for **ROS2 Jazzy**. This is currently the only version known to be compatible with this tutorial.

```
#!/bin/bash
set -e

# Update and upgrade apt-get
sudo apt-get update
sudo apt-get upgrade -y

# https://gazebosim.org/docs/harmonic/install_ubuntu/
sudo apt-get install -y curl lsb-release gnupg

# Install gazebo harmonic (https://gazebosim.org/docs/harmonic/install/)
sudo curl https://packages.osrfoundation.org/gazebo.gpg --output /usr/share/keyrings/
↪pkgs-osrf-archive-keyring.gpg
echo "deb [arch=$(dpkg --print-architecture) signed-by=/usr/share/keyrings/pkgs-osrf-
↪archive-keyring.gpg] http://packages.osrfoundation.org/gazebo/ubuntu-stable $(lsb_
↪release -cs) main" | sudo tee /etc/apt/sources.list.d/gazebo-stable.list > /dev/null
sudo apt-get update
sudo apt-get install -y gz-harmonic

## Install pairing ROS pairing libraries
sudo apt-get install -y ros-jazzy-ros-gz
```

**Danger**

If your intention is to follow these tutorials, please do not attempt to install it following any other documentation. This could lead to a different version being installed and this tutorial is unlikely to work.

Here is the description of the packages we are installing. You can notice that the packages are being used in the commands to add the **Gazebo Harmonic** packages to our **apt** sources. We're just telling **apt** where to find the packages

and install those.

| | |
|---|---|
| curl | Helps download files from the terminal. |
| lsb-release | The lsb_release command is a simple tool to help identify the Linux distribution being used and its compliance with the Linux Standard Base. |
| gnupg | **GnuPG** is an universal crypto engine which can be used directly from a command line prompt, from shell scripts, or from other programs. |

## 41.1 Running Gazebo

**Note**

The `gz sim` command might not work if you have issues in your network configuration. You might need to enable this through your firewall.

```
sudo ufw allow 10317:10318/udp
```

**Danger**

Obviously do not do this unless you need to, check with only `gz sim` first. Exposing ports through the firewall might leave your computer exposed to malicious actions.

Content from `ROS 2 Networking and Communication`

> https://github.com/UoMMScRobotics/UOMDocumentationForLeoRover/blob/main/Further_reading/Networking.md by https://github.com/Https404PaigeNotFound

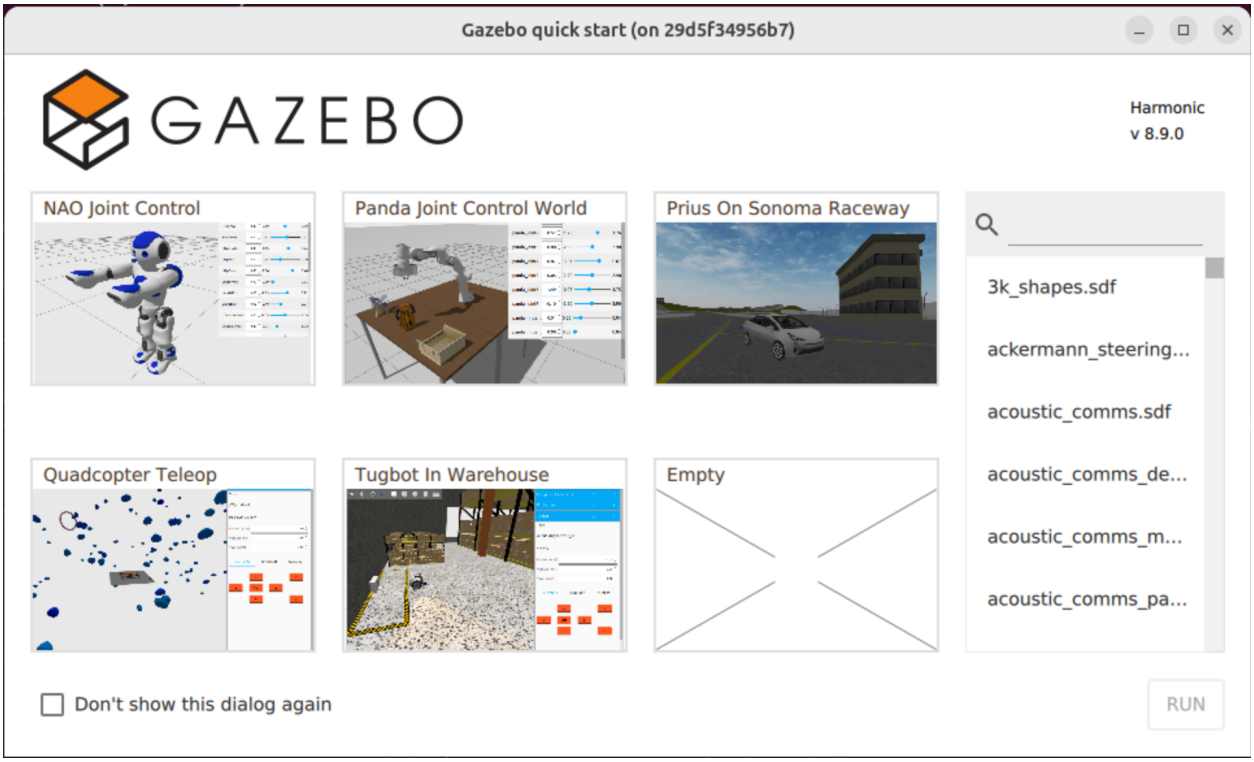
After installation, **`Gazebo Harmonic`** can be run with the following command

```
gz sim
```

Which should result in something similar to the following, if the installation went well.

**Danger**

The `3k_shapes.sdf` is current freezing all my machines. Click at your own risk. Other scenes seem to be working in general.



CHAPTER

# FORTYTWO

# USING GAZEBO

Added in version Jazzy: This section.

**Warning**

This topic is under construction and this might not even be its final form. Please feel free to open an issue if you spot any typos or other problems.

**Hint**

You can start the simulator with the following command.

```
gz sim
```

In this section we will see the basic operations related to **Gazebo** and create our own `sdf` file.

## 42.1 Basic functionality

These two official tutorials cover the basic functionality of **Gazebo**. They are very well made with up-to-date images and videos. Please go through them to familiarise yourself with the basic functionality.

- Understanding the GUI.
- Manipulating Models.

## 42.2 Fun with plugins

Check the guides below for basic functionality.

- Apply force torque plugin.
- Mouse drag plugin.

## 42.3 World files `.sdf`

**See also**

Official documentation: https://gazebosim.org/docs/harmonic/sdf_worlds/

Let us start with the sample scene `shapes.sdf`. This example is available in the official repository. It is shown below to save you a click. The format is `xml`, which should be a familiar file format by now.

### Contents of `shapes.sdf`

```
<?xml version="1.0" ?>
<sdf version="1.11">
  <!--
    Try moving a model using the command in the following CDATA block::
  -->
  <![CDATA[
    gz service -s /world/shapes/set_pose \
               --reqtype gz.msgs.Pose --reptype gz.msgs.Boolean \
               --timeout 300 --req 'name: "box", position: {z: 5.0}'
  ]]>
  <world name="shapes">
    <scene>
      <ambient>1.0 1.0 1.0</ambient>
      <background>0.8 0.8 0.8</background>
    </scene>

    <light type="directional" name="sun">
      <cast_shadows>true</cast_shadows>
      <pose>0 0 10 0 0 0</pose>
      <diffuse>0.8 0.8 0.8 1</diffuse>
      <specular>0.2 0.2 0.2 1</specular>
      <attenuation>
        <range>1000</range>
        <constant>0.9</constant>
        <linear>0.01</linear>
        <quadratic>0.001</quadratic>
      </attenuation>
      <direction>-0.5 0.1 -0.9</direction>
    </light>

    <model name="ground_plane">
      <static>true</static>
      <link name="link">
        <collision name="collision">
          <geometry>
            <plane>
              <normal>0 0 1</normal>
              <size>100 100</size>
            </plane>
          </geometry>
        </collision>
        <visual name="visual">
          <geometry>
```

(continues on next page)

(continued from previous page)

```
          <plane>
            <normal>0 0 1</normal>
            <size>100 100</size>
          </plane>
        </geometry>
        <material>
          <ambient>0.8 0.8 0.8 1</ambient>
          <diffuse>0.8 0.8 0.8 1</diffuse>
          <specular>0.8 0.8 0.8 1</specular>
        </material>
      </visual>
    </link>
  </model>

  <model name="box">
    <pose>0 0 0.5 0 0 0</pose>
    <link name="box_link">
      <inertial>
        <inertia>
          <ixx>0.16666</ixx>
          <ixy>0</ixy>
          <ixz>0</ixz>
          <iyy>0.16666</iyy>
          <iyz>0</iyz>
          <izz>0.16666</izz>
        </inertia>
        <mass>1.0</mass>
      </inertial>
      <collision name="box_collision">
        <geometry>
          <box>
            <size>1 1 1</size>
          </box>
        </geometry>
      </collision>

      <visual name="box_visual">
        <geometry>
          <box>
            <size>1 1 1</size>
          </box>
        </geometry>
        <material>
          <ambient>1 0 0 1</ambient>
          <diffuse>1 0 0 1</diffuse>
          <specular>1 0 0 1</specular>
        </material>
      </visual>
    </link>
  </model>

  <model name="cylinder">
```

(continues on next page)

(continued from previous page)

```
    <pose>0 -1.5 0.5 0 0 0</pose>
    <link name="cylinder_link">
      <inertial>
        <inertia>
          <ixx>0.1458</ixx>
          <ixy>0</ixy>
          <ixz>0</ixz>
          <iyy>0.1458</iyy>
          <iyz>0</iyz>
          <izz>0.125</izz>
        </inertia>
        <mass>1.0</mass>
      </inertial>
      <collision name="cylinder_collision">
        <geometry>
          <cylinder>
            <radius>0.5</radius>
            <length>1.0</length>
          </cylinder>
        </geometry>
      </collision>

      <visual name="cylinder_visual">
        <geometry>
          <cylinder>
            <radius>0.5</radius>
            <length>1.0</length>
          </cylinder>
        </geometry>
        <material>
          <ambient>0 1 0 1</ambient>
          <diffuse>0 1 0 1</diffuse>
          <specular>0 1 0 1</specular>
        </material>
      </visual>
    </link>
  </model>

  <model name="sphere">
    <pose>0 1.5 0.5 0 0 0</pose>
    <link name="sphere_link">
      <inertial>
        <inertia>
          <ixx>0.1</ixx>
          <ixy>0</ixy>
          <ixz>0</ixz>
          <iyy>0.1</iyy>
          <iyz>0</iyz>
          <izz>0.1</izz>
        </inertia>
        <mass>1.0</mass>
      </inertial>
```

(continues on next page)

(continued from previous page)

```
      <collision name="sphere_collision">
        <geometry>
          <sphere>
            <radius>0.5</radius>
          </sphere>
        </geometry>
      </collision>

      <visual name="sphere_visual">
        <geometry>
          <sphere>
            <radius>0.5</radius>
          </sphere>
        </geometry>
        <material>
          <ambient>0 0 1 1</ambient>
          <diffuse>0 0 1 1</diffuse>
          <specular>0 0 1 1</specular>
        </material>
      </visual>
    </link>
  </model>

  <model name="capsule">
    <pose>0 -3.0 0.5 0 0 0</pose>
    <link name="capsule_link">
      <inertial>
        <inertia>
          <ixx>0.074154</ixx>
          <ixy>0</ixy>
          <ixz>0</ixz>
          <iyy>0.074154</iyy>
          <iyz>0</iyz>
          <izz>0.018769</izz>
        </inertia>
        <mass>1.0</mass>
      </inertial>
      <collision name="capsule_collision">
        <geometry>
          <capsule>
            <radius>0.2</radius>
            <length>0.6</length>
          </capsule>
        </geometry>
      </collision>
      <visual name="capsule_visual">
        <geometry>
          <capsule>
            <radius>0.2</radius>
            <length>0.6</length>
          </capsule>
        </geometry>
```

(continues on next page)

(continued from previous page)

```
        <material>
          <ambient>1 1 0 1</ambient>
          <diffuse>1 1 0 1</diffuse>
          <specular>1 1 0 1</specular>
        </material>
      </visual>
    </link>
  </model>

  <model name="ellipsoid">
    <pose>0 3.0 0.5 0 0 0</pose>
    <link name="ellipsoid_link">
      <inertial>
        <inertia>
          <ixx>0.068</ixx>
          <ixy>0</ixy>
          <ixz>0</ixz>
          <iyy>0.058</iyy>
          <iyz>0</iyz>
          <izz>0.026</izz>
        </inertia>
        <mass>1.0</mass>
      </inertial>
      <collision name="ellipsoid_collision">
        <geometry>
          <ellipsoid>
            <radii>0.2 0.3 0.5</radii>
          </ellipsoid>
        </geometry>
      </collision>
      <visual name="ellipsoid_visual">
        <geometry>
          <ellipsoid>
            <radii>0.2 0.3 0.5</radii>
          </ellipsoid>
        </geometry>
        <material>
          <ambient>1 0 1 1</ambient>
          <diffuse>1 0 1 1</diffuse>
          <specular>1 0 1 1</specular>
        </material>
      </visual>
    </link>
  </model>

  <model name="cone">
    <pose>0 4.5 0.5 0 0 0</pose>
    <link name="cone_link">
      <inertial auto="true">
        <density>1</density>
      </inertial>
      <collision name="cone_collision">
```

(continues on next page)

(continued from previous page)

```
        <geometry>
          <cone>
            <radius>0.5</radius>
            <length>1.0</length>
          </cone>
        </geometry>
      </collision>

      <visual name="cone_visual">
        <geometry>
          <cone>
            <radius>0.5</radius>
            <length>1.0</length>
          </cone>
        </geometry>
        <material>
          <ambient>1 0.47 0 1</ambient>
          <diffuse>1 0.47 0 1</diffuse>
          <specular>1 0.47 0 1</specular>
        </material>
      </visual>
    </link>
  </model>
 </world>
</sdf>
```

As you could have noticed, even simple world files can have a relatively complex format. The example `shapes.sdf`, that has only a few simple objects, is difficult to parse visually. For most things, it is easier to use **Gazebo** itself and limit and direct changes in `.sdf` files.

Adding or removing things directly on **Gazebo** is sometimes not repeatable with possible GUI changes across versions. In addition, it is not as natural for a text-based tutorial such as this one. In those scenarios, I will show how to edit the file directly.

This tutorial will limit itself on using default `.sdf` files and making small modifications. Our interest is to be able obtain and send useful information from **Gazebo** to **ROS**.

You will notice in this file some of the most common elements of `.sdf`. These are not the only elements and this list is not meant to be comprehensive.

- <sdf> is the highest-level element, defining the entire `.sdf`.
- <world name="world_name"> encloses the elements in this world and gives it a name. This name, rather than the name of the file, will be used when exposing topics and services.
- <scene> has broad elements of the scene, such as background color.
- **Entities can include:**
  - <light> a source of light
  - <model> a model in the simulation, such as simple shapes or a robot, that can include other information such as <link>, <visual>, and <collision>.

> **Note**
>
> If your application demands it, you will need specialised programs to create 3D models, such as Blender. It won't be usually manageable to create complicated models directly on **Gazebo**.

In this example, because we're using simple shapes supported by **Gazebo**, they are defined with <geometry> elements.

## 42.4 Gazebo topics and services

> **Caution**
>
> Internal **Gazebo** topics and services are **not** the same as **ROS2** topics and services.

A **Gazebo** scene will have a number of internal topics and services. We can interact with them using **gz** commands. The messages that flow through **Gazebo** are based on protobuf, not **ROS2** IDL.

Let us use an example with sensors. This example is available in the official repository.

### Contents of `sensors_demo.sdf`

```
<?xml version="1.0" ?>
<!--
  Load several rendering sensors.
-->
<sdf version="1.6">
  <world name="lidar_sensor">
    <physics name="1ms" type="ignored">
      <max_step_size>0.001</max_step_size>
      <real_time_factor>1.0</real_time_factor>
    </physics>
    <plugin
      filename="gz-sim-physics-system"
      name="gz::sim::systems::Physics">
    </plugin>
    <plugin
      filename="gz-sim-sensors-system"
      name="gz::sim::systems::Sensors">
      <render_engine>ogre2</render_engine>
    </plugin>
    <plugin
      filename="gz-sim-user-commands-system"
      name="gz::sim::systems::UserCommands">
    </plugin>
    <plugin
      filename="gz-sim-scene-broadcaster-system"
      name="gz::sim::systems::SceneBroadcaster">
    </plugin>


    <gui fullscreen="0">

```

(continues on next page)

(continued from previous page)

```
    <!-- 3D scene -->
    <plugin filename="MinimalScene" name="3D View">
      <gz-gui>
        <title>3D View</title>
        <property type="bool" key="showTitleBar">false</property>
        <property type="string" key="state">docked</property>
      </gz-gui>

      <engine>ogre2</engine>
      <scene>scene</scene>
      <ambient_light>0.4 0.4 0.4</ambient_light>
      <background_color>0.8 0.8 0.8</background_color>
      <camera_pose>-6 0 6 0 0.5 0</camera_pose>
    </plugin>

    <!-- Plugins that add functionality to the scene -->
    <plugin filename="EntityContextMenuPlugin" name="Entity context menu">
      <gz-gui>
        <property key="state" type="string">floating</property>
        <property key="width" type="double">5</property>
        <property key="height" type="double">5</property>
        <property key="showTitleBar" type="bool">false</property>
      </gz-gui>
    </plugin>
    <plugin filename="GzSceneManager" name="Scene Manager">
      <gz-gui>
        <property key="resizable" type="bool">false</property>
        <property key="width" type="double">5</property>
        <property key="height" type="double">5</property>
        <property key="state" type="string">floating</property>
        <property key="showTitleBar" type="bool">false</property>
      </gz-gui>
    </plugin>
    <plugin filename="InteractiveViewControl" name="Interactive view control">
      <gz-gui>
        <property key="resizable" type="bool">false</property>
        <property key="width" type="double">5</property>
        <property key="height" type="double">5</property>
        <property key="state" type="string">floating</property>
        <property key="showTitleBar" type="bool">false</property>
      </gz-gui>
    </plugin>
    <plugin filename="CameraTracking" name="Camera Tracking">
      <gz-gui>
        <property key="resizable" type="bool">false</property>
        <property key="width" type="double">5</property>
        <property key="height" type="double">5</property>
        <property key="state" type="string">floating</property>
        <property key="showTitleBar" type="bool">false</property>
      </gz-gui>
    </plugin>
    <!-- World control -->
```

(continues on next page)

(continued from previous page)

```
    <plugin filename="WorldControl" name="World control">
      <gz-gui>
        <title>World control</title>
        <property type="bool" key="showTitleBar">false</property>
        <property type="bool" key="resizable">false</property>
        <property type="double" key="height">72</property>
        <property type="double" key="z">1</property>

        <property type="string" key="state">floating</property>
        <anchors target="3D View">
          <line own="left" target="left"/>
          <line own="bottom" target="bottom"/>
        </anchors>
      </gz-gui>

      <play_pause>true</play_pause>
      <step>true</step>
      <start_paused>true</start_paused>
      <use_event>true</use_event>

    </plugin>

    <!-- World statistics -->
    <plugin filename="WorldStats" name="World stats">
      <gz-gui>
        <title>World stats</title>
        <property type="bool" key="showTitleBar">false</property>
        <property type="bool" key="resizable">false</property>
        <property type="double" key="height">110</property>
        <property type="double" key="width">290</property>
        <property type="double" key="z">1</property>

        <property type="string" key="state">floating</property>
        <anchors target="3D View">
          <line own="right" target="right"/>
          <line own="bottom" target="bottom"/>
        </anchors>
      </gz-gui>

      <sim_time>true</sim_time>
      <real_time>true</real_time>
      <real_time_factor>true</real_time_factor>
      <iterations>true</iterations>
    </plugin>

    <!-- Inspector -->
    <plugin filename="ComponentInspector" name="Component inspector">
      <gz-gui>
        <property type="string" key="state">docked</property>
      </gz-gui>
    </plugin>

```

(continues on next page)

(continued from previous page)

```
    <!-- Entity tree -->
    <plugin filename="EntityTree" name="Entity tree">
      <gz-gui>
        <property type="string" key="state">docked</property>
      </gz-gui>
    </plugin>

    <plugin filename="ImageDisplay" name="Image Display">
      <gz-gui>
        <title>RGB camera</title>
        <property key="state" type="string">floating</property>
        <property type="double" key="width">350</property>
        <property type="double" key="height">315</property>
      </gz-gui>
      <topic>camera</topic>
      <topic_picker>false</topic_picker>
    </plugin>
    <plugin filename="ImageDisplay" name="Image Display 2">
      <gz-gui>
        <title>Depth camera</title>
        <property key="state" type="string">floating</property>
        <property type="double" key="width">350</property>
        <property type="double" key="height">315</property>
        <property type="double" key="x">500</property>
      </gz-gui>
      <topic>depth_camera</topic>
      <topic_picker>false</topic_picker>
    </plugin>
    <plugin filename="ImageDisplay" name="Image Display 3">
      <gz-gui>
        <title>RGBD: image</title>
        <property key="state" type="string">floating</property>
        <property type="double" key="width">350</property>
        <property type="double" key="height">315</property>
        <property type="double" key="y">320</property>
      </gz-gui>
      <topic>rgbd_camera/image</topic>
      <topic_picker>false</topic_picker>
    </plugin>
    <plugin filename="ImageDisplay" name="Image Display 3">
      <gz-gui>
        <title>RGBD: depth</title>
        <property key="state" type="string">floating</property>
        <property type="double" key="width">350</property>
        <property type="double" key="height">315</property>
        <property type="double" key="x">500</property>
        <property type="double" key="y">320</property>
      </gz-gui>
      <topic>rgbd_camera/depth_image</topic>
      <topic_picker>false</topic_picker>
    </plugin>
    <plugin filename="ImageDisplay" name="Image Display 5">
```

(continues on next page)

(continued from previous page)

```
      <gz-gui>
        <title>Thermal camera</title>
        <property key="state" type="string">floating</property>
        <property type="double" key="width">350</property>
        <property type="double" key="height">315</property>
        <property type="double" key="x">500</property>
        <property type="double" key="y">640</property>
      </gz-gui>
      <topic>thermal_camera</topic>
      <topic_picker>false</topic_picker>
    </plugin>
  </gui>

  <light type="directional" name="sun">
    <cast_shadows>true</cast_shadows>
    <pose>0 0 10 0 0 0</pose>
    <diffuse>0.8 0.8 0.8 1</diffuse>
    <specular>0.2 0.2 0.2 1</specular>
    <attenuation>
      <range>1000</range>
      <constant>0.9</constant>
      <linear>0.01</linear>
      <quadratic>0.001</quadratic>
    </attenuation>
    <direction>-0.5 0.1 -0.9</direction>
  </light>

  <model name="ground_plane">
    <static>true</static>
    <link name="link">
      <collision name="collision">
        <geometry>
          <!--plane>
            <normal>0 0 1</normal>
            <size>100 100</size>
          </plane-->
          <box>
            <size>20 20 0.1</size>
          </box>
        </geometry>
      </collision>
      <visual name="visual">
        <geometry>
          <!--plane>
            <normal>0 0 1</normal>
            <size>100 100</size>
          </plane-->
          <box>
            <size>20 20 0.1</size>
          </box>
        </geometry>
        <material>
```

(continues on next page)

(continued from previous page)

```
          <ambient>0.8 0.8 0.8 1</ambient>
          <diffuse>0.8 0.8 0.8 1</diffuse>
          <specular>0.8 0.8 0.8 1</specular>
        </material>
      </visual>
    </link>
  </model>

  <model name="box">
    <pose>0 -1 0.5 0 0 0</pose>
    <link name="box_link">
      <inertial>
        <inertia>
          <ixx>1</ixx>
          <ixy>0</ixy>
          <ixz>0</ixz>
          <iyy>1</iyy>
          <iyz>0</iyz>
          <izz>1</izz>
        </inertia>
        <mass>1.0</mass>
      </inertial>
      <collision name="box_collision">
        <geometry>
          <box>
            <size>1 1 1</size>
          </box>
        </geometry>
      </collision>

      <visual name="box_visual">
        <geometry>
          <box>
            <size>1 1 1</size>
          </box>
        </geometry>
        <material>
          <ambient>1 0 0 1</ambient>
          <diffuse>1 0 0 1</diffuse>
          <specular>1 0 0 1</specular>
        </material>
      </visual>
    </link>
  </model>

  <model name="cameras_alone">
    <pose>2.5 0 1.5 0 0.0 3.14</pose>
    <link name="link">
      <pose>0.05 0.05 0.05 0 0 0</pose>
      <inertial>
        <mass>0.1</mass>
        <inertia>
```

(continues on next page)

(continued from previous page)

```
            <ixx>0.000166667</ixx>
            <iyy>0.000166667</iyy>
            <izz>0.000166667</izz>
          </inertia>
        </inertial>
        <collision name="collision">
          <geometry>
            <box>
              <size>0.1 0.1 0.1</size>
            </box>
          </geometry>
        </collision>
        <visual name="visual">
          <geometry>
            <box>
              <size>0.1 0.1 0.1</size>
            </box>
          </geometry>
        </visual>
        <sensor name="cameras_alone" type="camera">
          <camera>
            <horizontal_fov>1.047</horizontal_fov>
            <image>
              <width>320</width>
              <height>240</height>
            </image>
            <clip>
              <near>0.1</near>
              <far>100</far>
            </clip>
          </camera>
          <always_on>1</always_on>
          <update_rate>30</update_rate>
          <visualize>true</visualize>
          <topic>camera_alone</topic>
          <enable_metrics>true</enable_metrics>
        </sensor>
        <sensor name="depth_camera1" type="depth_camera">
          <update_rate>10</update_rate>
          <topic>depth_camera</topic>
          <enable_metrics>true</enable_metrics>
          <camera>
            <horizontal_fov>1.05</horizontal_fov>
            <image>
              <width>320</width>
              <height>240</height>
              <format>R_FLOAT32</format>
            </image>
            <clip>
              <near>0.1</near>
              <far>10.0</far>
            </clip>
```

(continues on next page)

(continued from previous page)

```
        </camera>
        </sensor>
    </link>
    <static>true</static>
  </model>

  <model name="camera_with_lidar">
    <pose>4 0 0.5 0 0.0 3.14</pose>
    <link name="link">
      <pose>0.05 0.05 0.05 0 0 0</pose>
      <inertial>
        <mass>0.1</mass>
        <inertia>
          <ixx>0.000166667</ixx>
          <iyy>0.000166667</iyy>
          <izz>0.000166667</izz>
        </inertia>
      </inertial>
      <collision name="collision">
        <geometry>
          <box>
            <size>0.1 0.1 0.1</size>
          </box>
        </geometry>
      </collision>
      <visual name="visual">
        <geometry>
          <box>
            <size>0.1 0.1 0.1</size>
          </box>
        </geometry>
      </visual>
      <sensor name="camera" type="camera">
        <camera>
          <horizontal_fov>1.047</horizontal_fov>
          <image>
            <width>320</width>
            <height>240</height>
          </image>
          <clip>
            <near>0.1</near>
            <far>100</far>
          </clip>
        </camera>
        <always_on>1</always_on>
        <update_rate>30</update_rate>
        <visualize>true</visualize>
        <topic>camera</topic>
      </sensor>

      <sensor name='gpu_lidar' type='gpu_lidar'>"
        <topic>lidar</topic>
```

(continues on next page)

(continued from previous page)

```
        <update_rate>10</update_rate>
        <enable_metrics>true</enable_metrics>
        <ray>
          <scan>
            <horizontal>
              <samples>640</samples>
              <resolution>1</resolution>
              <min_angle>-1.396263</min_angle>
              <max_angle>1.396263</max_angle>
            </horizontal>
            <vertical>
              <samples>1</samples>
              <resolution>0.01</resolution>
              <min_angle>0</min_angle>
              <max_angle>0</max_angle>
            </vertical>
          </scan>
          <range>
            <min>0.08</min>
            <max>10.0</max>
            <resolution>0.01</resolution>
          </range>
        </ray>
        <visualize>true</visualize>
      </sensor>
    </link>

    <static>true</static>
  </model>

  <model name="rgbd_camera">
    <pose>5 0 0.5 0 0.0 3.14</pose>
    <link name="link">
      <pose>0.05 0.05 0.05 0 0 0</pose>
      <inertial>
        <mass>0.1</mass>
        <inertia>
          <ixx>0.000166667</ixx>
          <iyy>0.000166667</iyy>
          <izz>0.000166667</izz>
        </inertia>
      </inertial>
      <collision name="collision">
        <geometry>
          <box>
            <size>0.1 0.1 0.1</size>
          </box>
        </geometry>
      </collision>
      <visual name="visual">
        <geometry>
          <box>
```

(continues on next page)

(continued from previous page)

```
            <size>0.1 0.1 0.1</size>
          </box>
        </geometry>
      </visual>
      <sensor name="rgbd_camera" type="rgbd_camera">
        <camera>
          <horizontal_fov>1.047</horizontal_fov>
          <image>
            <width>320</width>
            <height>240</height>
          </image>
          <clip>
            <near>0.1</near>
            <far>100</far>
          </clip>
        </camera>
        <always_on>1</always_on>
        <update_rate>30</update_rate>
        <visualize>true</visualize>
        <topic>rgbd_camera</topic>
        <enable_metrics>true</enable_metrics>
      </sensor>
    </link>
  </model>

  <model name="thermal_camera">
    <pose>3 0 0.5 0 0.0 3.14</pose>
    <link name="link">
      <pose>0.05 0.05 0.05 0 0 0</pose>
      <inertial>
        <mass>0.1</mass>
        <inertia>
          <ixx>0.000166667</ixx>
          <iyy>0.000166667</iyy>
          <izz>0.000166667</izz>
        </inertia>
      </inertial>
      <collision name="collision">
        <geometry>
          <box>
            <size>0.1 0.1 0.1</size>
          </box>
        </geometry>
      </collision>
      <visual name="visual">
        <geometry>
          <box>
            <size>0.1 0.1 0.1</size>
          </box>
        </geometry>
      </visual>
      <sensor name="thermal_camera" type="thermal_camera">
```

(continues on next page)

(continued from previous page)

```
          <camera>
            <horizontal_fov>1.047</horizontal_fov>
            <image>
              <width>320</width>
              <height>240</height>
            </image>
            <clip>
              <near>0.1</near>
              <far>100</far>
            </clip>
          </camera>
          <always_on>1</always_on>
          <update_rate>30</update_rate>
          <visualize>true</visualize>
          <topic>thermal_camera</topic>
        </sensor>
      </link>
    </model>

    <include>
      <pose>0 1 3 0.0 0.0 1.57</pose>
      <uri>
      https://fuel.gazebosim.org/1.0/OpenRobotics/models/Construction Cone
      </uri>
    </include>

  </world>
</sdf>
```

To enable physical simulation, including sensor, we can see that a new tag, <plugins> is added. These are paramount to guarantee that sensors will work. Further, they create important topics and services in **Gazebo**. Here are the plugins active on this file.

```
    <physics name="1ms" type="ignored">
      <max_step_size>0.001</max_step_size>
      <real_time_factor>1.0</real_time_factor>
    </physics>
    <plugin
      filename="gz-sim-physics-system"
      name="gz::sim::systems::Physics">
    </plugin>
    <plugin
      filename="gz-sim-sensors-system"
      name="gz::sim::systems::Sensors">
      <render_engine>ogre2</render_engine>
    </plugin>
    <plugin
      filename="gz-sim-user-commands-system"
      name="gz::sim::systems::UserCommands">
    </plugin>
    <plugin
      filename="gz-sim-scene-broadcaster-system"
```

(continues on next page)

(continued from previous page)

```
      name="gz::sim::systems::SceneBroadcaster">
    </plugin>
```

We will now start to interact with **Gazebo**. For this section, suppose that we have the scene always open with the following command.

**Tip**

You will see online, very frequently, the command using `-v4`. For instance, as follows. This allows **Gazebo** to output many messages to the terminal in which it was opened to help you understand what is happening.

```
gz sim -v4 sensors_demo.sdf
```

```
gz sim sensors_demo.sdf
```

A frequently used tool to allow us to inspect internal Gazebo topics will be, in another terminal, to run the following command. The `-l` flag allows us to list **Gazebo** topics.

```
gz topic -l
```

If the correct scene is running on **Gazebo**, the output should be similar to the following output.

```
/camera
/camera_alone
/camera_info
/clock
/depth_camera
/depth_camera/performance_metrics
/depth_camera/points
/gazebo/resource_paths
/gui/camera/pose
/gui/currently_tracked
/gui/track
/lidar
/lidar/points
/rgbd_camera/camera_info
/rgbd_camera/depth_image
/rgbd_camera/image
/rgbd_camera/performance_metrics
/rgbd_camera/points
/sensors/marker
/stats
/thermal_camera
/world/lidar_sensor/clock
/world/lidar_sensor/dynamic_pose/info
/world/lidar_sensor/pose/info
/world/lidar_sensor/scene/deletion
/world/lidar_sensor/scene/info
/world/lidar_sensor/state
/world/lidar_sensor/stats
/world/lidar_sensor/light_config
```

(continues on next page)

(continued from previous page)

```
/world/lidar_sensor/material_color
```

Further information can be obtained about topics, for instance, the type of message that flows through them. We can check what is the message type used in one of the sensors as follows.

```
gz topic -i --topic /rgbd_camera/image
```

The output of that command should be similar to the following.

```
Publishers [Address, Message Type]:
  tcp://172.16.191.128:40679, gz.msgs.Image
Subscribers [Address, Message Type]:
  tcp://172.16.191.128:40229, gz.msgs.Image
```

Information about services, similarly, can be obtained with the following command.

```
gz service -l
```

Resulting in the following output.

### Services in `sensors_demo.sdf`

```
/camera/set_rate
/camera_alone/set_rate
/depth_camera/set_rate
/gazebo/resource_paths/add
/gazebo/resource_paths/get
/gazebo/resource_paths/resolve
/gazebo/worlds
/gui/camera/view_control
/gui/camera/view_control/reference_visual
/gui/camera/view_control/sensitivity
/gui/follow
/gui/follow/offset
/gui/move_to
/gui/move_to/pose
/lidar/set_rate
/rgbd_camera/set_rate
/sensors/marker
/sensors/marker/list
/sensors/marker_array
/server_control
/thermal_camera/set_rate
/world/lidar_sensor/control
/world/lidar_sensor/control/state
/world/lidar_sensor/create
/world/lidar_sensor/create/blocking
/world/lidar_sensor/create_multiple
/world/lidar_sensor/create_multiple/blocking
/world/lidar_sensor/declare_parameter
/world/lidar_sensor/disable_collision
/world/lidar_sensor/disable_collision/blocking
/world/lidar_sensor/enable_collision
```

(continues on next page)

(continued from previous page)

```
/world/lidar_sensor/enable_collision/blocking
/world/lidar_sensor/entity/system/add
/world/lidar_sensor/generate_world_sdf
/world/lidar_sensor/get_parameter
/world/lidar_sensor/gui/info
/world/lidar_sensor/level/set_performer
/world/lidar_sensor/light_config
/world/lidar_sensor/light_config/blocking
/world/lidar_sensor/list_parameters
/world/lidar_sensor/playback/control
/world/lidar_sensor/remove
/world/lidar_sensor/remove/blocking
/world/lidar_sensor/scene/graph
/world/lidar_sensor/scene/info
/world/lidar_sensor/set_parameter
/world/lidar_sensor/set_physics
/world/lidar_sensor/set_physics/blocking
/world/lidar_sensor/set_pose
/world/lidar_sensor/set_pose/blocking
/world/lidar_sensor/set_pose_vector
/world/lidar_sensor/set_pose_vector/blocking
/world/lidar_sensor/set_spherical_coordinates
/world/lidar_sensor/set_spherical_coordinates/blocking
/world/lidar_sensor/state
/world/lidar_sensor/state_async
/world/lidar_sensor/system/info
/world/lidar_sensor/visual_config
/world/lidar_sensor/visual_config/blocking
/world/lidar_sensor/wheel_slip
/world/lidar_sensor/wheel_slip/blocking
```

We can also obtain information about each service. For instance, with the command.

```
gz service -i --service /rgbd_camera/set_rate
```

Resulting in the following output.

```
Service providers [Address, Request Message Type, Response Message Type]:
  tcp://172.16.191.128:36013, gz.msgs.Double, gz.msgs.Empty
```

There are multiple ways to interact with the **Gazebo** internal topics. This can be, for instance, done through the commandline using similar commands to the ones we have shown so far. Nonetheless, if you have spent so much time learning **ROS2**, it would be more convenient to find a way to **bridge** the topics and services from **Gazebo** and **ROS2**.

Nonetheless, when some piece of information is not flowing as expected, remember these commands to help troubleshoot.

## 42.5 The package `ros_gz_sim`

**See also**

The official repository: https://github.com/gazebosim/ros_gz/tree/jazzy/ros_gz_sim

The package `ros_gz_sim` has a few tools allowing us to control **Gazebo** over **ROS2**. Namely, the ability to start **Gazebo** and spawn (add) objects. These might depend on **Gazebo** topics and services, therefore make sure that the necessary plugins are enabled.

### 42.5.1 Launching Gazebo

For example, we can launch **Gazebo** directly from a launch file.

```
ros2 launch ros_gz_sim gz_sim.launch.py gz_args:=shapes.sdf
```

In this command, `gz_args` we add a representation file known to **Gazebo**. In this case, `shapes.sdf`. You can specify other scenes.

### 42.5.2 Spawn models

You can use the following command to spawn models in **Gazebo**.

#### Terminal 1: Run Gazebo

```
ros2 launch ros_gz_sim gz_sim.launch.py gz_args:=shapes.sdf
```

#### Terminal 2: Spawn model

```
ros2 launch ros_gz_sim gz_spawn_model.launch.py world:=shapes file:=$(ros2 pkg prefix --
↪share ros_gz_sim_demos)/models/vehicle/model.sdf entity_name:=my_vehicle x:=5.0 y:=5.0␣
↪z:=0.5
```

As arguments we have the

- model filename
- name for the entity inside **Gazebo**
- position coordinates of the model

**Note**

The scene specified in the argument `world` must be active in **Gazebo** for the command above to work.

This will add the model, in this case, `model.sdf` to the scene `shapes.sdf`. The output on the terminal will be as follows.

```
[INFO] [launch]: All log files can be found below /home/murilo/.ros/log/2025-11-06-11-19-
↪14-321361-murilo-VMware20-1-14158
[INFO] [launch]: Default logging verbosity is set to INFO
[INFO] [create-1]: process started with pid [14161]
[create-1] [WARN] [1762427959.367373569] [ros_gz_sim]: Waiting for service [/world/
↪shapes/create] to become available ...
```

(continues on next page)

(continued from previous page)

```
[create-1] [INFO] [1762427959.598035759] [ros_gz_sim]: Entity creation successful.
[INFO] [create-1]: process has finished cleanly [pid 14161]
```

## 42.6 The package `ros_gz_sim_demos`

**See also**

The official repository: https://github.com/gazebosim/ros_gz/tree/jazzy/ros_gz_sim_demos

This official package has many examples integrating **Gazebo** and **ROS2**. It showcases many functionalities that are useful for robotics applications.

You can verify all files usable for **Gazebo** demos in the following folder.

```
cd $(ros2 pkg prefix --share ros_gz_sim_demos)
nautilus .
```

The current file structure is shown in the dropdown below. There are many launch files, **rviz** configuration files, and a few `.sdf` models.

#### The structure of `ros_gz_sim_demos`.

```
/opt/ros/jazzy/share/ros_gz_sim_demos
├── cmake
│   ├── ros_gz_sim_demosConfig.cmake
│   └── ros_gz_sim_demosConfig-version.cmake
├── environment
│   ├── ament_prefix_path.dsv
│   ├── ament_prefix_path.sh
│   ├── path.dsv
│   ├── path.sh
│   └── ros_gz_sim_demos.dsv
├── launch
│   ├── air_pressure.launch.py
│   ├── battery.launch.py
│   ├── camera.launch.py
│   ├── depth_camera.launch.py
│   ├── diff_drive.launch.py
│   ├── gpu_lidar_bridge.launch.py
│   ├── gpu_lidar.launch.py
│   ├── image_bridge.launch.py
│   ├── imu.launch.py
│   ├── joint_states.launch.py
│   ├── magnetometer.launch.py
│   ├── navsat_gpsfix.launch.py
│   ├── navsat.launch.py
│   ├── rgbd_camera_bridge.launch.py
│   ├── rgbd_camera.launch.py
│   ├── robot_description_publisher.launch.py
│   ├── sdf_parser.launch.py
```

(continues on next page)

(continued from previous page)

```
│   ├── tf_bridge.launch.py
│   └── triggered_camera.launch.py
├── local_setup.bash
├── local_setup.dsv
├── local_setup.sh
├── local_setup.zsh
├── models
│   ├── cardboard_box
│   │   ├── materials
│   │   │   └── textures
│   │   │       └── cardboard_box.png
│   │   ├── meshes
│   │   │   └── cardboard_box.dae
│   │   ├── model.config
│   │   ├── model.sdf
│   │   └── thumbnails
│   │       ├── 1.png
│   │       ├── 2.png
│   │       ├── 3.png
│   │       ├── 4.png
│   │       └── 5.png
│   ├── double_pendulum_model.sdf
│   ├── rrbot.xacro
│   └── vehicle
│       ├── model.config
│       └── model.sdf
├── package.dsv
├── package.xml
├── rviz
│   ├── camera.rviz
│   ├── depth_camera.rviz
│   ├── diff_drive.rviz
│   ├── gpu_lidar_bridge.rviz
│   ├── gpu_lidar.rviz
│   ├── imu.rviz
│   ├── joint_states.rviz
│   ├── rgbd_camera_bridge.rviz
│   ├── rgbd_camera.rviz
│   ├── robot_description_publisher.rviz
│   ├── tf_bridge.rviz
│   └── vehicle.rviz
└── worlds
    ├── default.sdf
    └── vehicle.sdf
```

> **Note**
>
> The contents above are not retrieved automatically so they might not represent the latest version of the repository. See https://github.com/gazebosim/ros_gz/tree/jazzy/ros_gz_sim_demos .

### 42.6.1 The `camera.launch.py` demo

We can run this example with the following command.

```
ros2 launch ros_gz_sim_demos camera.launch.py
```

This will show **Gazebo**, with a couple of objects and a camera. The camera view is being rendered by **Gazebo**. The simulation is started. At the same time, **rviz2** is executed. This demo is important because cameras are difficult to simulate otherwise and being able to access them via **ROS2** allows you to create powerful image-based robot controllers and planners.

We can take a look at the topics created, while those programs are running, with the following command.

```
ros2 topic list
```

This will output the following.

```
/camera
/camera_info
/clicked_point
/initialpose
/move_base_simple/goal
/parameter_events
/rosout
/tf
/tf_static
```

The launch file, `camera.launch.py`, is currently performing the following steps.

- Starting **Gazebo** with the built-in scene `camera_sensor.sdf`.
- Running **rviz2** with the configuration file `camera.rviz`.
- Running `ros_gz_bridge` with the correct parameters to expose the camera sensor from **Gazebo** to **ROS2**.

This demo shows the camera simulation results on **rviz2** because it's possibly the most convenient way of showing that the information is correctly flowing through **ROS2**.

> **Note**
>
> Some unused topics such as `/tf` are defined in `camera.rviz`, but don't worry about those.

Below are the contents of the launch file.

#### The contents of `launch/camera.launch.py`

```
# Copyright 2019 Open Source Robotics Foundation, Inc.
#
# Licensed under the Apache License, Version 2.0 (the "License");
```

(continues on next page)

(continued from previous page)

```
# you may not use this file except in compliance with the License.
# You may obtain a copy of the License at
#
#     http://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
# See the License for the specific language governing permissions and
# limitations under the License.

import os

from ament_index_python.packages import get_package_share_directory

from launch import LaunchDescription
from launch.actions import DeclareLaunchArgument
from launch.actions import IncludeLaunchDescription
from launch.conditions import IfCondition
from launch.launch_description_sources import PythonLaunchDescriptionSource
from launch.substitutions import LaunchConfiguration

from launch_ros.actions import Node


def generate_launch_description():

    pkg_ros_gz_sim_demos = get_package_share_directory('ros_gz_sim_demos')
    pkg_ros_gz_sim = get_package_share_directory('ros_gz_sim')

    gz_sim = IncludeLaunchDescription(
        PythonLaunchDescriptionSource(
            os.path.join(pkg_ros_gz_sim, 'launch', 'gz_sim.launch.py')),
        launch_arguments={'gz_args': '-r camera_sensor.sdf'}.items(),
    )

    # RViz
    rviz = Node(
        package='rviz2',
        executable='rviz2',
        arguments=['-d', os.path.join(pkg_ros_gz_sim_demos, 'rviz', 'camera.rviz')],
        condition=IfCondition(LaunchConfiguration('rviz'))
    )

    # Bridge
    bridge = Node(
        package='ros_gz_bridge',
        executable='parameter_bridge',
        arguments=['/camera@sensor_msgs/msg/Image@gz.msgs.Image',
                   '/camera_info@sensor_msgs/msg/CameraInfo@gz.msgs.CameraInfo'],
        output='screen'
    )
```

(continues on next page)

(continued from previous page)

```
    return LaunchDescription([
        DeclareLaunchArgument('rviz', default_value='true',
                              description='Open RViz.'),
        gz_sim,
        bridge,
        rviz
    ])
```

Below are the contents of the rviz file.

### The contents of `rviz/camera.rviz`

```
Panels:
  - Class: rviz_common/Displays
    Help Height: 0
    Name: Displays
    Property Tree Widget:
      Expanded:
        - /Global Options1
        - /Camera1
        - /Camera1/Status1
        - /Image1
      Splitter Ratio: 0.5
    Tree Height: 557
  - Class: rviz_common/Selection
    Name: Selection
  - Class: rviz_common/Tool Properties
    Expanded:
      - /2D Nav Goal1
      - /Publish Point1
    Name: Tool Properties
    Splitter Ratio: 0.5886790156364441
  - Class: rviz_common/Views
    Expanded:
      - /Current View1
    Name: Views
    Splitter Ratio: 0.5
Visualization Manager:
  Class: ""
  Displays:
    - Alpha: 0.5
      Cell Size: 1
      Class: rviz_default_plugins/Grid
      Color: 160; 160; 164
      Enabled: true
      Line Style:
        Line Width: 0.029999999329447746
        Value: Lines
      Name: Grid
      Normal Cell Count: 0
      Offset:
```

(continues on next page)

(continued from previous page)

```
      X: 0
      Y: 0
      Z: 0
    Plane: XY
    Plane Cell Count: 10
    Reference Frame: <Fixed Frame>
    Value: true
  - Class: rviz_default_plugins/Camera
    Enabled: true
    Image Rendering: background and overlay
    Name: Camera
    Overlay Alpha: 0.5
    Queue Size: 10
    Topic: /camera
    Unreliable: false
    Value: true
    Visibility:
      Grid: true
      Image: true
      Value: true
    Zoom Factor: 1
  - Class: rviz_default_plugins/Image
    Enabled: true
    Max Value: 1
    Median window: 5
    Min Value: 0
    Name: Image
    Normalize Range: true
    Queue Size: 10
    Topic: /camera
    Unreliable: false
    Value: true
Enabled: true
Global Options:
  Background Color: 48; 48; 48
  Fixed Frame: camera/link/camera
  Frame Rate: 30
Name: root
Tools:
  - Class: rviz_default_plugins/MoveCamera
  - Class: rviz_default_plugins/Select
  - Class: rviz_default_plugins/FocusCamera
  - Class: rviz_default_plugins/Measure
    Line color: 128; 128; 0
  - Class: rviz_default_plugins/SetInitialPose
    Topic: /initialpose
  - Class: rviz_default_plugins/SetGoal
    Topic: /move_base_simple/goal
  - Class: rviz_default_plugins/PublishPoint
    Single click: true
    Topic: /clicked_point
Transformation:
```

(continues on next page)

(continued from previous page)

```
    Current:
      Class: rviz_default_plugins/TF
  Value: true
  Views:
    Current:
      Class: rviz_default_plugins/Orbit
      Distance: 19.73822784423828
      Enable Stereo Rendering:
        Stereo Eye Separation: 0.05999999865889549
        Stereo Focal Distance: 1
        Swap Stereo Eyes: false
        Value: false
      Focal Point:
        X: 0
        Y: 0
        Z: 0
      Focal Shape Fixed Size: true
      Focal Shape Size: 0.05000000074505806
      Invert Z Axis: false
      Name: Current View
      Near Clip Distance: 0.009999999776482582
      Pitch: 0.7903980016708374
      Target Frame: <Fixed Frame>
      Value: Orbit (rviz)
      Yaw: 0.785398006439209
    Saved: ~
Window Geometry:
  Camera:
    collapsed: false
  Displays:
    collapsed: false
  Height: 702
  Hide Left Dock: false
  Hide Right Dock: false
  Image:
    collapsed: false
  QMainWindow State:␣
↪000000ff00000000fd0000000400000000000001560000000268fc0200000008fb0000001200530065006c0065006300740069006
  Selection:
    collapsed: false
  Tool Properties:
    collapsed: false
  Views:
    collapsed: false
  Width: 859
  X: 517
  Y: 361
```

The launch file uses the package `ros_gz_bridge`, which is used to create the interfaces, for instance topics, between **Gazebo** and **ROS2**.

We will see this package in more detail in the following section.

## 42.7 References

The official documentation for **Gazebo** is overall good. Here are some main topics where this tutorial borrowed from.

- https://gazebosim.org/docs/harmonic/sensors/
- https://gazebosim.org/docs/harmonic/ros2_launch_gazebo/
- https://gazebosim.org/docs/harmonic/ros2_integration/
- https://gazebosim.org/docs/harmonic/ros2_spawn_model/
- https://gazebosim.org/docs/harmonic/ros2_interop/
- https://gazebosim.org/docs/harmonic/ros_gz_project_template_guide/
- https://github.com/gazebosim/ros_gz/blob/jazzy/ros_gz_bridge/README.md

CHAPTER

# FORTYTHREE

# USING ROS_GZ_BRIDGE

Added in version Jazzy: This section.

**Warning**

This topic is under construction and this might not even be its final form. Please feel free to open an issue if you spot any typos or other problems.

**See also**

Official documentation: https://docs.ros.org/en/jazzy/p/ros_gz_bridge/

We have seen how to see internal **Gazebo** topics through the command line. This is not yet very useful to us, because we want to be able to integrate **Gazebo** with our own, custom **ROS2** nodes.

The idea is to map **Gazebo** topics into **ROS2** topics. This is where `ros_gz_bridge` comes in.

Each **Gazebo** message must be paired with a correct **ROS2** message if you want to access these outside **Gazebo**. The file ros_gz_bridge/README.md in the official documentation shows the mappings.

Here is the current table.

```
The following message types can be bridged for topics:

| ROS type                                      | Gazebo Transport Type              |
| --------------------------------------------- | :--------------------------:      |
| actuator_msgs/msg/Actuators                   | gz.msgs.Actuators                  |
| builtin_interfaces/msg/Time                   | gz.msgs.Time                       |
| geometry_msgs/msg/Point                       | gz.msgs.Vector3d                   |
| geometry_msgs/msg/Pose                        | gz.msgs.Pose                       |
| geometry_msgs/msg/PoseArray                   | gz.msgs.Pose_V                     |
| geometry_msgs/msg/PoseStamped                 | gz.msgs.Pose                       |
| geometry_msgs/msg/PoseWithCovariance          | gz.msgs.PoseWithCovariance         |
| geometry_msgs/msg/PoseWithCovarianceStamped   | gz.msgs.PoseWithCovariance         |
| geometry_msgs/msg/Quaternion                  | gz.msgs.Quaternion                 |
| geometry_msgs/msg/Transform                   | gz.msgs.Pose                       |
| geometry_msgs/msg/TransformStamped            | gz.msgs.Pose                       |
| geometry_msgs/msg/Twist                       | gz.msgs.Twist                      |
| geometry_msgs/msg/TwistStamped                | gz.msgs.Twist                      |
| geometry_msgs/msg/TwistWithCovariance         | gz.msgs.TwistWithCovariance        |
```

(continues on next page)

(continued from previous page)

| | |
|---|---|
| geometry_msgs/msg/TwistWithCovarianceStamped | gz.msgs.TwistWithCovariance |
| geometry_msgs/msg/Vector3 | gz.msgs.Vector3d |
| geometry_msgs/msg/Wrench | gz.msgs.Wrench |
| geometry_msgs/msg/WrenchStamped | gz.msgs.Wrench |
| gps_msgs/msg/GPSFix | gz.msgs.NavSat |
| marine_acoustic_msgs/msg/Dvl | gz.msgs.DVLVelocityTracking |
| nav_msgs/msg/Odometry | gz.msgs.Odometry |
| nav_msgs/msg/Odometry | gz.msgs.OdometryWithCovariance |
| rcl_interfaces/msg/ParameterValue | gz.msgs.Any |
| ros_gz_interfaces/msg/Altimeter | gz.msgs.Altimeter |
| ros_gz_interfaces/msg/Contact | gz.msgs.Contact |
| ros_gz_interfaces/msg/Contacts | gz.msgs.Contacts |
| ros_gz_interfaces/msg/Dataframe | gz.msgs.Dataframe |
| ros_gz_interfaces/msg/Entity | gz.msgs.Entity |
| ros_gz_interfaces/msg/EntityWrench | gz.msgs.EntityWrench |
| ros_gz_interfaces/msg/Float32Array | gz.msgs.Float_V |
| ros_gz_interfaces/msg/GuiCamera | gz.msgs.GUICamera |
| ros_gz_interfaces/msg/JointWrench | gz.msgs.JointWrench |
| ros_gz_interfaces/msg/Light | gz.msgs.Light |
| ros_gz_interfaces/msg/LogicalCameraImage | gz.msgs.LogicalCameraImage |
| ros_gz_interfaces/msg/LogPlaybackStatistics | gz.msgs.LogPlaybackStatistics |
| ros_gz_interfaces/msg/ParamVec | gz.msgs.Param |
| ros_gz_interfaces/msg/ParamVec | gz.msgs.Param_V |
| ros_gz_interfaces/msg/SensorNoise | gz.msgs.SensorNoise |
| ros_gz_interfaces/msg/StringVec | gz.msgs.StringMsg_V |
| ros_gz_interfaces/msg/TrackVisual | gz.msgs.TrackVisual |
| ros_gz_interfaces/msg/VideoRecord | gz.msgs.VideoRecord |
| ros_gz_interfaces/msg/WorldStatistics | gz.msgs.WorldStatistics |
| rosgraph_msgs/msg/Clock | gz.msgs.Clock |
| sensor_msgs/msg/BatteryState | gz.msgs.BatteryState |
| sensor_msgs/msg/CameraInfo | gz.msgs.CameraInfo |
| sensor_msgs/msg/FluidPressure | gz.msgs.FluidPressure |
| sensor_msgs/msg/Image | gz.msgs.Image |
| sensor_msgs/msg/Imu | gz.msgs.IMU |
| sensor_msgs/msg/JointState | gz.msgs.Model |
| sensor_msgs/msg/Joy | gz.msgs.Joy |
| sensor_msgs/msg/LaserScan | gz.msgs.LaserScan |
| sensor_msgs/msg/MagneticField | gz.msgs.Magnetometer |
| sensor_msgs/msg/NavSatFix | gz.msgs.NavSat |
| sensor_msgs/msg/PointCloud2 | gz.msgs.PointCloudPacked |
| sensor_msgs/msg/Range | gz.msgs.LaserScan |
| std_msgs/msg/Bool | gz.msgs.Boolean |
| std_msgs/msg/ColorRGBA | gz.msgs.Color |
| std_msgs/msg/Empty | gz.msgs.Empty |
| std_msgs/msg/Float32 | gz.msgs.Float |
| std_msgs/msg/Float64 | gz.msgs.Double |
| std_msgs/msg/Header | gz.msgs.Header |
| std_msgs/msg/Int32 | gz.msgs.Int32 |
| std_msgs/msg/String | gz.msgs.StringMsg |
| std_msgs/msg/UInt32 | gz.msgs.UInt32 |
| tf2_msgs/msg/TFMessage | gz.msgs.Pose_V |
| trajectory_msgs/msg/JointTrajectory | gz.msgs.JointTrajectory |

(continues on next page)

(continued from previous page)

```
| vision_msgs/msg/Detection2D                  | gz.msgs.AnnotatedAxisAligned2DBox   |
| vision_msgs/msg/Detection2DArray             | gz.msgs.AnnotatedAxisAligned2DBox_V |
| vision_msgs/msg/Detection3D                  | gz::msgs::AnnotatedOriented3DBox    |
| vision_msgs/msg/Detection3DArray             | gz::msgs::AnnotatedOriented3DBox_V  |

And the following for services:
```

**Important**

Some of these messages are part of the package `ros_gz_interfaces`. You can see more details about these in the official repository.

We will be using mostly **`parameter_bridge`**, which is part of `ros_gz_bridge` to make these connections. More information about the tool can be obtained with the help command.

```
ros2 run ros_gz_bridge parameter_bridge -h
```

To summarise the output, it is expected that we run

> ros2 run ros_gz_bridge parameter_bridge <Gazebo Topic>@<ROS Type>@<Gazebo Transport Type>

Using the logic above, we have that

- <Gazebo Topic> defined for the entity on Gazebo and can be obtained with `gz topic -l`.
- We can find <Gazebo Transport Type> with `gz topic -i --topic <Gazebo Topic>`.
- Lastly, we check the pairing table and see what <ROS Type> matches the <Gazebo Transport Type>.
- The second @ means a bidirectional bridge. It can also be **[** or **]** to indicate a single direction. Some official examples are relatively lax with this rule.

Let's see some examples.

## 43.1 Sensors

**See also**

Official documentation: https://gazebosim.org/docs/harmonic/sensors/

Simulated sensor information is becoming evermore useful. In simulators, one of the main aspects making them useful will be our ability to obtain their data through **`ROS2`**. Cameras are likely to be the most common, but other sensors, such as lidars, are frequently used.

For each of these subsections, suppose that we have the `sensors_demo.sdf` scene always open. Don't forget to start the simulation as well!

```
gz sim sensors_demo.sdf
```

**Note**

Sensor information will only start to be published after the simulation is started in **Gazebo**.

### 43.1.1 gz.msgs.Image

In a previous section, we have seen that the **Gazebo** topic `/rgbd_camera/image` exists and has transport type `gz.msgs.Image`. Looking at the pairing table, we notice that the pairing **ROS2** message type is `sensor_msgs/msg/Image`.

We can run, therefore in one terminal, the following command.

```
ros2 run ros_gz_bridge \
parameter_bridge \
/rgbd_camera/image@sensor_msgs/msg/Image[gz.msgs.Image
```

To show the images, we can use **`rqt_image_view`**. Notice that the internal **Gazebo** topic name will be replicated into a **ROS2** topic with the same name.

```
ros2 run rqt_image_view rqt_image_view /rgbd_camera/image
```

This will show the image obtained from **Gazebo** in **`rqt_image_view`**, effectively showing that these two are paired.

**Exercises**

Can you do the same for the `/depth_camera` or `/thermal_camera` internal **Gazebo** topic available in the same scene? What steps would you take to show the images available in **Gazebo** into **`rqt_image_view`**?

### 43.1.2 gz.msgs.LaserScan

We have already listed the **Gazebo** topics for this scene, so we know that there is a topic called `/lidar`. We can obtain the related information with the following command.

```
gz topic -i --topic /lidar
```

This will output the following.

```
Publishers [Address, Message Type]:
  tcp://172.16.191.128:40679, gz.msgs.LaserScan
No subscribers on topic [/lidar]
```

By looking at the pairing table, we find that `gz.msgs.LaserScan` should be paired with a `sensor_msgs/msg/LaserScan`. Therefore, we can run the bridge as follows.

```
ros2 run ros_gz_bridge \
parameter_bridge \
/lidar@sensor_msgs/msg/LaserScan[gz.msgs.LaserScan
```

One convenient way to visualise this messages is through **`rviz2`**. We can do so as follows.

```
ros2 run rviz2 rviz2 -f camera_with_lidar/link/gpu_lidar
```

> **Note**
>
> We are starting **rviz2** with the reference frame `camera_with_lidar/link/gpu_lidar` because that is the frame shown in the topic `/lidar`.

We can then define the proper **rviz2** view so be able to visualise the laser scan results.

1. *Add* → *rviz_default_plugins* → *LaserScan* → *OK*.
2. *Displays* → *LaserScan* → *topic* → *lidar* → *Press ENTER*.

## 43.2 Getting pose information

One important aspect of entities in any simulation is their pose. We must be able to obtain poses to define the behavior of our robots. For instance, if a box in a simulation moves, the controller might need this information to define evasive maneuvers.

The most basic manner to be able to obtain the pose of an entity in **Gazebo** from an external program is to add the following to each model, in the `.sdf` scene.

```
<plugin
        filename="gz-sim-pose-publisher-system"
        name="gz::sim::systems::PosePublisher">
    <publish_link_pose>false</publish_link_pose>
    <publish_collision_pose>false</publish_collision_pose>
    <publish_visual_pose>false</publish_visual_pose>
    <publish_nested_model_pose>true</publish_nested_model_pose>
</plugin>
```

> **Note**
>
> Settings will vary for more advanced use, but for basic shapes this will be enough.

To show how this works, let us create a folder for our modified `shapes.sdf` scene.

```
mkdir -p ~/gazebo_tutorial_workspace/scenes
cd ~/gazebo_tutorial_workspace/scenes
```

In this case, the file is quite large, so I suggest downloading `shapes_with_pose_publisher.sdf` and adding it to the folder above.

`shapes_with_pose_publisher.sdf`

### Contents of `shapes_with_pose_publisher.sdf`

You will note that this is simply the default `shapes.sdf` with the plugin added to each entity.

```
<?xml version="1.0" ?>
<sdf version="1.11">
    <!--
      Try moving a model using the command in the following CDATA block::
    -->
    <![CDATA[
```

(continues on next page)

(continued from previous page)

```
    gz service -s /world/shapes/set_pose \
               --reqtype gz.msgs.Pose --reptype gz.msgs.Boolean \
               --timeout 300 --req 'name: "box", position: {z: 5.0}'
  ]]>
    <world name="shapes_with_pose_publisher">

        <scene>
            <ambient>1.0 1.0 1.0</ambient>
            <background>0.8 0.8 0.8</background>
        </scene>

        <light type="directional" name="sun">
            <cast_shadows>true</cast_shadows>
            <pose>0 0 10 0 0 0</pose>
            <diffuse>0.8 0.8 0.8 1</diffuse>
            <specular>0.2 0.2 0.2 1</specular>
            <attenuation>
                <range>1000</range>
                <constant>0.9</constant>
                <linear>0.01</linear>
                <quadratic>0.001</quadratic>
            </attenuation>
            <direction>-0.5 0.1 -0.9</direction>
        </light>

        <model name="ground_plane">
            <static>true</static>
            <link name="link">
                <collision name="collision">
                    <geometry>
                        <plane>
                            <normal>0 0 1</normal>
                            <size>100 100</size>
                        </plane>
                    </geometry>
                </collision>
                <visual name="visual">
                    <geometry>
                        <plane>
                            <normal>0 0 1</normal>
                            <size>100 100</size>
                        </plane>
                    </geometry>
                    <material>
                        <ambient>0.8 0.8 0.8 1</ambient>
                        <diffuse>0.8 0.8 0.8 1</diffuse>
                        <specular>0.8 0.8 0.8 1</specular>
                    </material>
                </visual>
            </link>
        </model>

```

(continues on next page)

(continued from previous page)

```
        <model name="box">
            <pose>0 0 0.5 0 0 0</pose>
            <link name="box_link">
                <inertial>
                    <inertia>
                        <ixx>0.16666</ixx>
                        <ixy>0</ixy>
                        <ixz>0</ixz>
                        <iyy>0.16666</iyy>
                        <iyz>0</iyz>
                        <izz>0.16666</izz>
                    </inertia>
                    <mass>1.0</mass>
                </inertial>
                <collision name="box_collision">
                    <geometry>
                        <box>
                            <size>1 1 1</size>
                        </box>
                    </geometry>
                </collision>

                <visual name="box_visual">
                    <geometry>
                        <box>
                            <size>1 1 1</size>
                        </box>
                    </geometry>
                    <material>
                        <ambient>1 0 0 1</ambient>
                        <diffuse>1 0 0 1</diffuse>
                        <specular>1 0 0 1</specular>
                    </material>
                </visual>
            </link>

            <plugin
                    filename="gz-sim-pose-publisher-system"
                    name="gz::sim::systems::PosePublisher">
                <publish_link_pose>false</publish_link_pose>
                <publish_collision_pose>false</publish_collision_pose>
                <publish_visual_pose>false</publish_visual_pose>
                <publish_nested_model_pose>true</publish_nested_model_pose>
            </plugin>

        </model>

        <model name="cylinder">
            <pose>0 -1.5 0.5 0 0 0</pose>
            <link name="cylinder_link">
                <inertial>
                    <inertia>
```

(continues on next page)

(continued from previous page)

```
                    <ixx>0.1458</ixx>
                    <ixy>0</ixy>
                    <ixz>0</ixz>
                    <iyy>0.1458</iyy>
                    <iyz>0</iyz>
                    <izz>0.125</izz>
                </inertia>
                <mass>1.0</mass>
            </inertial>
            <collision name="cylinder_collision">
                <geometry>
                    <cylinder>
                        <radius>0.5</radius>
                        <length>1.0</length>
                    </cylinder>
                </geometry>
            </collision>

            <visual name="cylinder_visual">
                <geometry>
                    <cylinder>
                        <radius>0.5</radius>
                        <length>1.0</length>
                    </cylinder>
                </geometry>
                <material>
                    <ambient>0 1 0 1</ambient>
                    <diffuse>0 1 0 1</diffuse>
                    <specular>0 1 0 1</specular>
                </material>
            </visual>
        </link>

        <plugin
                filename="gz-sim-pose-publisher-system"
                name="gz::sim::systems::PosePublisher">
            <publish_link_pose>false</publish_link_pose>
            <publish_collision_pose>false</publish_collision_pose>
            <publish_visual_pose>false</publish_visual_pose>
            <publish_nested_model_pose>true</publish_nested_model_pose>
        </plugin>

    </model>

    <model name="sphere">
        <pose>0 1.5 0.5 0 0 0</pose>
        <link name="sphere_link">
            <inertial>
                <inertia>
                    <ixx>0.1</ixx>
                    <ixy>0</ixy>
                    <ixz>0</ixz>
```

(continues on next page)

(continued from previous page)

```
                    <iyy>0.1</iyy>
                    <iyz>0</iyz>
                    <izz>0.1</izz>
                </inertia>
                <mass>1.0</mass>
            </inertial>
            <collision name="sphere_collision">
                <geometry>
                    <sphere>
                        <radius>0.5</radius>
                    </sphere>
                </geometry>
            </collision>

            <visual name="sphere_visual">
                <geometry>
                    <sphere>
                        <radius>0.5</radius>
                    </sphere>
                </geometry>
                <material>
                    <ambient>0 0 1 1</ambient>
                    <diffuse>0 0 1 1</diffuse>
                    <specular>0 0 1 1</specular>
                </material>
            </visual>
        </link>

        <plugin
                filename="gz-sim-pose-publisher-system"
                name="gz::sim::systems::PosePublisher">
            <publish_link_pose>false</publish_link_pose>
            <publish_collision_pose>false</publish_collision_pose>
            <publish_visual_pose>false</publish_visual_pose>
            <publish_nested_model_pose>true</publish_nested_model_pose>
        </plugin>

    </model>

    <model name="capsule">
        <pose>0 -3.0 0.5 0 0 0</pose>
        <link name="capsule_link">
            <inertial>
                <inertia>
                    <ixx>0.074154</ixx>
                    <ixy>0</ixy>
                    <ixz>0</ixz>
                    <iyy>0.074154</iyy>
                    <iyz>0</iyz>
                    <izz>0.018769</izz>
                </inertia>
                <mass>1.0</mass>
```

(continues on next page)

(continued from previous page)

```
            </inertial>
            <collision name="capsule_collision">
                <geometry>
                    <capsule>
                        <radius>0.2</radius>
                        <length>0.6</length>
                    </capsule>
                </geometry>
            </collision>
            <visual name="capsule_visual">
                <geometry>
                    <capsule>
                        <radius>0.2</radius>
                        <length>0.6</length>
                    </capsule>
                </geometry>
                <material>
                    <ambient>1 1 0 1</ambient>
                    <diffuse>1 1 0 1</diffuse>
                    <specular>1 1 0 1</specular>
                </material>
            </visual>
        </link>

        <plugin
                filename="gz-sim-pose-publisher-system"
                name="gz::sim::systems::PosePublisher">
            <publish_link_pose>false</publish_link_pose>
            <publish_collision_pose>false</publish_collision_pose>
            <publish_visual_pose>false</publish_visual_pose>
            <publish_nested_model_pose>true</publish_nested_model_pose>
        </plugin>

    </model>

    <model name="ellipsoid">
        <pose>0 3.0 0.5 0 0 0</pose>
        <link name="ellipsoid_link">
            <inertial>
                <inertia>
                    <ixx>0.068</ixx>
                    <ixy>0</ixy>
                    <ixz>0</ixz>
                    <iyy>0.058</iyy>
                    <iyz>0</iyz>
                    <izz>0.026</izz>
                </inertia>
                <mass>1.0</mass>
            </inertial>
            <collision name="ellipsoid_collision">
                <geometry>
                    <ellipsoid>
```

(continues on next page)

(continued from previous page)

```
                        <radii>0.2 0.3 0.5</radii>
                    </ellipsoid>
                </geometry>
            </collision>
            <visual name="ellipsoid_visual">
                <geometry>
                    <ellipsoid>
                        <radii>0.2 0.3 0.5</radii>
                    </ellipsoid>
                </geometry>
                <material>
                    <ambient>1 0 1 1</ambient>
                    <diffuse>1 0 1 1</diffuse>
                    <specular>1 0 1 1</specular>
                </material>
            </visual>
        </link>

        <plugin
                filename="gz-sim-pose-publisher-system"
                name="gz::sim::systems::PosePublisher">
            <publish_link_pose>false</publish_link_pose>
            <publish_collision_pose>false</publish_collision_pose>
            <publish_visual_pose>false</publish_visual_pose>
            <publish_nested_model_pose>true</publish_nested_model_pose>
        </plugin>

    </model>

    <model name="cone">
        <pose>0 4.5 0.5 0 0 0</pose>
        <link name="cone_link">
            <inertial auto="true">
                <density>1</density>
            </inertial>
            <collision name="cone_collision">
                <geometry>
                    <cone>
                        <radius>0.5</radius>
                        <length>1.0</length>
                    </cone>
                </geometry>
            </collision>

            <visual name="cone_visual">
                <geometry>
                    <cone>
                        <radius>0.5</radius>
                        <length>1.0</length>
                    </cone>
                </geometry>
                <material>
```

(continues on next page)

(continued from previous page)

```
                        <ambient>1 0.47 0 1</ambient>
                        <diffuse>1 0.47 0 1</diffuse>
                        <specular>1 0.47 0 1</specular>
                    </material>
                </visual>
            </link>

            <plugin
                    filename="gz-sim-pose-publisher-system"
                    name="gz::sim::systems::PosePublisher">
                <publish_link_pose>false</publish_link_pose>
                <publish_collision_pose>false</publish_collision_pose>
                <publish_visual_pose>false</publish_visual_pose>
                <publish_nested_model_pose>true</publish_nested_model_pose>
            </plugin>

        </model>
    </world>
</sdf>
```

We open **Gazebo** with this scene, as follows, then run the simulation by clicking the run button.

```
gz sim ~/gazebo_tutorial_workspace/scenes/shapes_with_pose_publisher.sdf
```

> **Note**
>
> Don't forget to run the simulation otherwise most information will not be available.

The plugin for each entity will create a `pose` topic. We can verify that with the following command.

```
gz topic -l
```

The result will be as follows, where the relevant topics are highlighted.

```
/clock
/gazebo/resource_paths
/gui/camera/pose
/gui/currently_tracked
/gui/track
/model/box/pose
/model/capsule/pose
/model/cone/pose
/model/cylinder/pose
/model/ellipsoid/pose
/model/sphere/pose
/stats
/world/shapes_with_pose_publisher/clock
/world/shapes_with_pose_publisher/dynamic_pose/info
/world/shapes_with_pose_publisher/pose/info
/world/shapes_with_pose_publisher/scene/deletion
/world/shapes_with_pose_publisher/scene/info
```

(continues on next page)

(continued from previous page)

```
/world/shapes_with_pose_publisher/state
/world/shapes_with_pose_publisher/stats
/world/shapes_with_pose_publisher/light_config
/world/shapes_with_pose_publisher/material_color
```

As always, we can investigate the **Gazebo** type these topics, as we expect all of them to be the same as they come from the same plugin. Let's choose the *box* as it's the first entity on the list.

We use the following command.

```
gz topic -i --topic /model/box/pose
```

It will output the type.

```
Publishers [Address, Message Type]:
  tcp://172.16.191.128:41611, gz.msgs.Pose
No subscribers on topic [/model/box/pose]
```

Because this will be an unilateral bridge from **Gazebo** to **ROS2**, we use the `[` instead of `@` as follows.

```
ros2 run ros_gz_bridge \
parameter_bridge \
/model/box/pose@geometry_msgs/msg/Pose[gz.msgs.Pose
```

We can see the contents of the topic with the following command. It will show the real-time information of the entity, even if you move it around within Gazebo.

```
ros2 topic echo /model/box/pose
```

We can see how fast your bridge is running with the usual command below.

```
ros2 topic hz /model/box/pose
```

Depending on the quality of your machine, your results will very, but you should expect this frequency to be quite high.

```
average rate: 799.066
    min: 0.000s max: 0.009s std dev: 0.00142s window: 800
average rate: 824.706
    min: 0.000s max: 0.009s std dev: 0.00129s window: 1652
average rate: 811.552
    min: 0.000s max: 0.010s std dev: 0.00131s window: 2439
average rate: 815.633
    min: 0.000s max: 0.010s std dev: 0.00129s window: 3267
```

**Exercises**

I have shown how to obtain the pose of the `box` entity. This scene has other five topics, namely.

- /model/capsule/pose
- /model/cone/pose
- /model/cylinder/pose
- /model/ellipsoid/pose

- /model/sphere/pose

Can you run the bridge of each one of these and see the results of each of these entities?

## 43.3 Getting transforms

**Warning**

You'll notice that each scene creates a **Gazebo** topic called `/world/<SCENE>/pose/info` where `<SCENE>` is the name of the active scene. Although this publishes useful information when checking through **Gazebo** topics, it does not play well with **ros_gz_bridge**. You can get the transform information without the headers, which in **ROS2** is not useful.

**See also**

https://github.com/gazebosim/ros_gz/issues/172

Integration with other parts of **ROS2** is important, therefore it could be useful to have all poses of relevant objects available for `tf2`. In this case, we need to modify the message that is published by **Gazebo**, but otherwise there is no big difference.

Our use of `gz::sim::systems::PosePublisher` will be slightly modified to have `<use_pose_vector_msg>true</use_pose_vector_msg>`, highlighted below.

```
<plugin
        filename="gz-sim-pose-publisher-system"
        name="gz::sim::systems::PosePublisher">
    <use_pose_vector_msg>true</use_pose_vector_msg>
    <publish_link_pose>false</publish_link_pose>
    <publish_collision_pose>false</publish_collision_pose>
    <publish_visual_pose>false</publish_visual_pose>
    <publish_nested_model_pose>true</publish_nested_model_pose>
</plugin>
```

Add the following file to your `~/gazebo_tutorial_workspace/scenes` folder.

`shapes_with_tf2_publisher.sdf`

### Contents of `shapes_with_tf2_publisher.sdf`

This is `shapes_with_pose_publisher.sdf` with one additional line in each `gz::sim::systems::PosePublisher` plugin.

```
<?xml version="1.0" ?>
<sdf version="1.11">
    <!--
      Try moving a model using the command in the following CDATA block::
    -->
    <![CDATA[
    gz service -s /world/shapes/set_pose \
               --reqtype gz.msgs.Pose --reptype gz.msgs.Boolean \
```

(continues on next page)

(continued from previous page)

```
              --timeout 300 --req 'name: "box", position: {z: 5.0}'
]]>
  <world name="shapes_with_tf2_publisher">

      <scene>
          <ambient>1.0 1.0 1.0</ambient>
          <background>0.8 0.8 0.8</background>
      </scene>

      <light type="directional" name="sun">
          <cast_shadows>true</cast_shadows>
          <pose>0 0 10 0 0 0</pose>
          <diffuse>0.8 0.8 0.8 1</diffuse>
          <specular>0.2 0.2 0.2 1</specular>
          <attenuation>
              <range>1000</range>
              <constant>0.9</constant>
              <linear>0.01</linear>
              <quadratic>0.001</quadratic>
          </attenuation>
          <direction>-0.5 0.1 -0.9</direction>
      </light>

      <model name="ground_plane">
          <static>true</static>
          <link name="link">
              <collision name="collision">
                  <geometry>
                      <plane>
                          <normal>0 0 1</normal>
                          <size>100 100</size>
                      </plane>
                  </geometry>
              </collision>
              <visual name="visual">
                  <geometry>
                      <plane>
                          <normal>0 0 1</normal>
                          <size>100 100</size>
                      </plane>
                  </geometry>
                  <material>
                      <ambient>0.8 0.8 0.8 1</ambient>
                      <diffuse>0.8 0.8 0.8 1</diffuse>
                      <specular>0.8 0.8 0.8 1</specular>
                  </material>
              </visual>
          </link>
      </model>

      <model name="box">
          <pose>0 0 0.5 0 0 0</pose>
```

(continues on next page)

(continued from previous page)

```
            <link name="box_link">
                <inertial>
                    <inertia>
                        <ixx>0.16666</ixx>
                        <ixy>0</ixy>
                        <ixz>0</ixz>
                        <iyy>0.16666</iyy>
                        <iyz>0</iyz>
                        <izz>0.16666</izz>
                    </inertia>
                    <mass>1.0</mass>
                </inertial>
                <collision name="box_collision">
                    <geometry>
                        <box>
                            <size>1 1 1</size>
                        </box>
                    </geometry>
                </collision>

                <visual name="box_visual">
                    <geometry>
                        <box>
                            <size>1 1 1</size>
                        </box>
                    </geometry>
                    <material>
                        <ambient>1 0 0 1</ambient>
                        <diffuse>1 0 0 1</diffuse>
                        <specular>1 0 0 1</specular>
                    </material>
                </visual>
            </link>

            <plugin
                    filename="gz-sim-pose-publisher-system"
                    name="gz::sim::systems::PosePublisher">
                <use_pose_vector_msg>true</use_pose_vector_msg>
                <publish_link_pose>false</publish_link_pose>
                <publish_collision_pose>false</publish_collision_pose>
                <publish_visual_pose>false</publish_visual_pose>
                <publish_nested_model_pose>true</publish_nested_model_pose>
            </plugin>

        </model>

        <model name="cylinder">
            <pose>0 -1.5 0.5 0 0 0</pose>
            <link name="cylinder_link">
                <inertial>
                    <inertia>
                        <ixx>0.1458</ixx>
```

(continues on next page)

(continued from previous page)

```
                    <ixy>0</ixy>
                    <ixz>0</ixz>
                    <iyy>0.1458</iyy>
                    <iyz>0</iyz>
                    <izz>0.125</izz>
                </inertia>
                <mass>1.0</mass>
            </inertial>
            <collision name="cylinder_collision">
                <geometry>
                    <cylinder>
                        <radius>0.5</radius>
                        <length>1.0</length>
                    </cylinder>
                </geometry>
            </collision>

            <visual name="cylinder_visual">
                <geometry>
                    <cylinder>
                        <radius>0.5</radius>
                        <length>1.0</length>
                    </cylinder>
                </geometry>
                <material>
                    <ambient>0 1 0 1</ambient>
                    <diffuse>0 1 0 1</diffuse>
                    <specular>0 1 0 1</specular>
                </material>
            </visual>
        </link>

        <plugin
                filename="gz-sim-pose-publisher-system"
                name="gz::sim::systems::PosePublisher">
            <use_pose_vector_msg>true</use_pose_vector_msg>
            <publish_link_pose>false</publish_link_pose>
            <publish_collision_pose>false</publish_collision_pose>
            <publish_visual_pose>false</publish_visual_pose>
            <publish_nested_model_pose>true</publish_nested_model_pose>
        </plugin>

    </model>

    <model name="sphere">
        <pose>0 1.5 0.5 0 0 0</pose>
        <link name="sphere_link">
            <inertial>
                <inertia>
                    <ixx>0.1</ixx>
                    <ixy>0</ixy>
                    <ixz>0</ixz>
```

(continues on next page)

(continued from previous page)

```
                    <iyy>0.1</iyy>
                    <iyz>0</iyz>
                    <izz>0.1</izz>
                </inertia>
                <mass>1.0</mass>
            </inertial>
            <collision name="sphere_collision">
                <geometry>
                    <sphere>
                        <radius>0.5</radius>
                    </sphere>
                </geometry>
            </collision>

            <visual name="sphere_visual">
                <geometry>
                    <sphere>
                        <radius>0.5</radius>
                    </sphere>
                </geometry>
                <material>
                    <ambient>0 0 1 1</ambient>
                    <diffuse>0 0 1 1</diffuse>
                    <specular>0 0 1 1</specular>
                </material>
            </visual>
        </link>

        <plugin
                filename="gz-sim-pose-publisher-system"
                name="gz::sim::systems::PosePublisher">
            <use_pose_vector_msg>true</use_pose_vector_msg>
            <publish_link_pose>false</publish_link_pose>
            <publish_collision_pose>false</publish_collision_pose>
            <publish_visual_pose>false</publish_visual_pose>
            <publish_nested_model_pose>true</publish_nested_model_pose>
        </plugin>

    </model>

    <model name="capsule">
        <pose>0 -3.0 0.5 0 0 0</pose>
        <link name="capsule_link">
            <inertial>
                <inertia>
                    <ixx>0.074154</ixx>
                    <ixy>0</ixy>
                    <ixz>0</ixz>
                    <iyy>0.074154</iyy>
                    <iyz>0</iyz>
                    <izz>0.018769</izz>
                </inertia>
```

(continues on next page)

(continued from previous page)

```
                    <mass>1.0</mass>
                </inertial>
                <collision name="capsule_collision">
                    <geometry>
                        <capsule>
                            <radius>0.2</radius>
                            <length>0.6</length>
                        </capsule>
                    </geometry>
                </collision>
                <visual name="capsule_visual">
                    <geometry>
                        <capsule>
                            <radius>0.2</radius>
                            <length>0.6</length>
                        </capsule>
                    </geometry>
                    <material>
                        <ambient>1 1 0 1</ambient>
                        <diffuse>1 1 0 1</diffuse>
                        <specular>1 1 0 1</specular>
                    </material>
                </visual>
            </link>

            <plugin
                    filename="gz-sim-pose-publisher-system"
                    name="gz::sim::systems::PosePublisher">
                <use_pose_vector_msg>true</use_pose_vector_msg>
                <publish_link_pose>false</publish_link_pose>
                <publish_collision_pose>false</publish_collision_pose>
                <publish_visual_pose>false</publish_visual_pose>
                <publish_nested_model_pose>true</publish_nested_model_pose>
            </plugin>

        </model>

        <model name="ellipsoid">
            <pose>0 3.0 0.5 0 0 0</pose>
            <link name="ellipsoid_link">
                <inertial>
                    <inertia>
                        <ixx>0.068</ixx>
                        <ixy>0</ixy>
                        <ixz>0</ixz>
                        <iyy>0.058</iyy>
                        <iyz>0</iyz>
                        <izz>0.026</izz>
                    </inertia>
                    <mass>1.0</mass>
                </inertial>
                <collision name="ellipsoid_collision">
```

(continues on next page)

(continued from previous page)

```
                    <geometry>
                        <ellipsoid>
                            <radii>0.2 0.3 0.5</radii>
                        </ellipsoid>
                    </geometry>
                </collision>
                <visual name="ellipsoid_visual">
                    <geometry>
                        <ellipsoid>
                            <radii>0.2 0.3 0.5</radii>
                        </ellipsoid>
                    </geometry>
                    <material>
                        <ambient>1 0 1 1</ambient>
                        <diffuse>1 0 1 1</diffuse>
                        <specular>1 0 1 1</specular>
                    </material>
                </visual>
            </link>

            <plugin
                    filename="gz-sim-pose-publisher-system"
                    name="gz::sim::systems::PosePublisher">
                <use_pose_vector_msg>true</use_pose_vector_msg>
                <publish_link_pose>false</publish_link_pose>
                <publish_collision_pose>false</publish_collision_pose>
                <publish_visual_pose>false</publish_visual_pose>
                <publish_nested_model_pose>true</publish_nested_model_pose>
            </plugin>

        </model>

        <model name="cone">
            <pose>0 4.5 0.5 0 0 0</pose>
            <link name="cone_link">
                <inertial auto="true">
                    <density>1</density>
                </inertial>
                <collision name="cone_collision">
                    <geometry>
                        <cone>
                            <radius>0.5</radius>
                            <length>1.0</length>
                        </cone>
                    </geometry>
                </collision>

                <visual name="cone_visual">
                    <geometry>
                        <cone>
                            <radius>0.5</radius>
                            <length>1.0</length>
```

(continues on next page)

(continued from previous page)

```
                    </cone>
                </geometry>
                <material>
                    <ambient>1 0.47 0 1</ambient>
                    <diffuse>1 0.47 0 1</diffuse>
                    <specular>1 0.47 0 1</specular>
                </material>
            </visual>
        </link>

        <plugin
                filename="gz-sim-pose-publisher-system"
                name="gz::sim::systems::PosePublisher">
            <use_pose_vector_msg>true</use_pose_vector_msg>
            <publish_link_pose>false</publish_link_pose>
            <publish_collision_pose>false</publish_collision_pose>
            <publish_visual_pose>false</publish_visual_pose>
            <publish_nested_model_pose>true</publish_nested_model_pose>
        </plugin>

    </model>
  </world>
</sdf>
```

We open **Gazebo** with this scene, as follows, then run the simulation by clicking the run button.

```
gz sim ~/gazebo_tutorial_workspace/scenes/shapes_with_tf2_publisher.sdf
```

> **Note**
>
> Don't forget to run the simulation otherwise most information will not be available.

In another terminal, we can check the type of message being used to publish poses.

```
gz topic -i --topic /model/box/pose
```

The output will be similar to the one below. Our modification in the plugin makes the **Gazebo** topic use a different message type, `gz.msgs.Pose_V`.

```
Publishers [Address, Message Type]:
  tcp://172.16.191.128:34555, gz.msgs.Pose_V
Subscribers [Address, Message Type]:
  tcp://172.16.191.128:37389, gz.msgs.Pose_V
```

The idea now is to bridge that type of message with `tf2`. Beyond simply bridging the topics, we have to make sure that these two points are correct.

- All transforms are published to a topic named `/tf`, for compatibility with `tf2`.
- The clock used in **ROS2** matches the one used in **Gazebo**. This will make sure that when we look up transforms the timestamp will make sense.

Now, because we have seven topics to bridge, it would not be practical to have that done through the command line. In

addition we need topics that have different names in **Gazebo** and **ROS2**.

We can take advantage of `.yaml` configuration files that are supported by **ros_gz_bridge**.

To show how this works, let us create a folder for our `.yaml` bridge files.

```
mkdir -p ~/gazebo_tutorial_workspace/bridge_config
cd ~/gazebo_tutorial_workspace/bridge_config
```

We add the following file to the newly created folder. The `/clock` bridge and the `/tf` bridge elements that are new are highlighted.

`transforms.yaml`

```
- ros_topic_name: "/clock"
  gz_topic_name: "/clock"
  ros_type_name: "rosgraph_msgs/msg/Clock"
  gz_type_name: "gz.msgs.Clock"
  direction: GZ_TO_ROS

- ros_topic_name: "/tf"
  gz_topic_name: "/model/box/pose"
  ros_type_name: "tf2_msgs/msg/TFMessage"
  gz_type_name: "gz.msgs.Pose_V"
  direction: GZ_TO_ROS

- ros_topic_name: "/tf"
  gz_topic_name: "/model/capsule/pose"
  ros_type_name: "tf2_msgs/msg/TFMessage"
  gz_type_name: "gz.msgs.Pose_V"
  direction: GZ_TO_ROS

- ros_topic_name: "/tf"
  gz_topic_name: "/model/cone/pose"
  ros_type_name: "tf2_msgs/msg/TFMessage"
  gz_type_name: "gz.msgs.Pose_V"
  direction: GZ_TO_ROS

- ros_topic_name: "/tf"
  gz_topic_name: "/model/cylinder/pose"
  ros_type_name: "tf2_msgs/msg/TFMessage"
  gz_type_name: "gz.msgs.Pose_V"
  direction: GZ_TO_ROS

- ros_topic_name: "/tf"
  gz_topic_name: "/model/ellipsoid/pose"
  ros_type_name: "tf2_msgs/msg/TFMessage"
  gz_type_name: "gz.msgs.Pose_V"
  direction: GZ_TO_ROS

- ros_topic_name: "/tf"
  gz_topic_name: "/model/sphere/pose"
  ros_type_name: "tf2_msgs/msg/TFMessage"
  gz_type_name: "gz.msgs.Pose_V"
  direction: GZ_TO_ROS
```

**Note**

You will see that bridge files allow more flexibility. For instance, we are able to bridge topics in that have different names in **Gazebo** and **ROS2**.

We now have to call **ros_gz_bridge** using this newly created file. Note that some **bash** commands will not expand ~ into the home folder. We can replace those instances safely with `$HOME`. We send any configuration we want with the flag `--ros-args -p`, then set the parameter `config_file` to have a path to our newly created configuration file.

```
ros2 run ros_gz_bridge \
parameter_bridge \
--ros-args -p \
config_file:=$HOME/gazebo_tutorial_workspace/bridge_config/transforms.yaml
```

In the messages published to `tf2` by gazebo, you will notice that the frame of reference is the name of the scene. In this case, `shapes_with_tf2_publisher`. Therefore, to see that the frames are correctly published via **ROS2**, we can see them in **rviz2** with the following command.

```
ros2 run rviz2 rviz2 -f shapes_with_tf2_publisher
```

After adding the `TF` display, you'll be able to see all the relevant frames.

**Exercise**

Suppose that you added a new model to `shapes_with_tf2_publisher.sdf` and you wanted to publish that information to `/tf` as well. What files would you modify and what steps would you take to make that possible?

So far, we have seen the basics of **ros_gz_bridge**, mostly allowing us to expose **Gazebo** information in **ROS2**. We haven't used any of these in our own nodes, so we will do that in the following section.

CHAPTER

# FORTYFOUR

# INTERFACE GAZEBO WITH CUSTOM ROS2 NODES

Added in version Jazzy: This section.

**Warning**

This topic is under construction and this might not even be its final form. Please feel free to open an issue if you spot any typos or other problems.

**See also**

Official documentation: https://gazebosim.org/docs/harmonic/ros_gz_project_template_guide/

In this section, our intention is to control things in **Gazebo** from a **ROS2** package, using nodes and launch files as needed. For things that are well supported in **Gazebo**, this will work fine. For anything new, you might need to dive more deeply into **Gazebo** plugins, which is considerably outside the scope of a tutorial like this one.

There is a lot of sample code available online for **Gazebo** integration. You will also see frequent patterns of having multiple packages in a single project, to correctly organize the package according to **Gazebo** logic. This logic tends to be reflected in other packages, such as examples using `nav2`.

Most of the sample code I found relates to the use of **Gazebo** plugins with specific functionality, such as `gz::sim::systems::DiffDrive`, `gz::sim::systems::JointStatePublisher`, and `gz::sim::systems::OdometryPublisher`.

Instead of repeating any of that, the idea here is to show how a simple node can perform simple communication with **Gazebo**. Using this logic, you can find out how to communicate with any existing plugins, updated plugins, or new plugins when they become available.

**Note**

I haven't yet found a comprehensive list of **Gazebo** plugins and their pairing `xml` descriptions to add them to a `.sdf`. Most of the information is spread around examples and takes a bit a trial-and-error to find dependencies between plugins.

Some common patterns are:

- Information as comments of the `.hh` files, see this example.
- Information available when you run **Gazebo** with `gz sim -v4`.

Please feel free to correct me here.

## 44.1 Setting up the scene

We will use a slightly modified version of the `shapes_with_tf2_publisher.sdf` created previously. Let's add it to our **Gazebo** folder.

We will use two systems in this example. We will use `gz::sim::systems::UserCommands` to allow us to set the pose of entities and `gz::sim::systems::ApplyLinkWrench` to allow us to set wrenches to links. This is illustrative because poses use **Gazebo** services and wrenches use **Gazebo** topics.

We will modify our `shapes_with_tf2_publisher.sdf` to add the following lines inside the `<world>` tag.

```
      <physics type="ode">
        <max_step_size>0.004</max_step_size>
        <real_time_factor>1.0</real_time_factor>
        <real_time_update_rate>250</real_time_update_rate>
      </physics>
      <plugin name='gz::sim::systems::Physics' filename='gz-sim-physics-system'/>
      <plugin filename="gz-sim-scene-broadcaster-system"  name=
↪"gz::sim::systems::SceneBroadcaster"/>
      <plugin filename="gz-sim-apply-link-wrench-system"  name=
↪"gz::sim::systems::ApplyLinkWrench"/>
      <plugin filename="gz-sim-user-commands-system" name=
↪"gz::sim::systems::UserCommands"/>
```

Here's a brief explanation of why we need each one of these.

> **Note**
>
> More information is available in the `gz-sim/src/systems` folder at https://github.com/gazebosim/gz-sim/tree/gz-sim8/src/systems.

| | |
|---|---|
| `gz::sim::systems::Physics` | The physics behavior needed by the other systems. |
| `gz::sim::systems::SceneBroadcaster` | Needed by the other systems [doc]. |
| `gz::sim::systems::ApplyLinkWrench` | Creates the `/wrench` topic [doc]. |
| `gz::sim::systems::UserCommands` | Creates the `/set_pose` service [doc]. |

We start by adding the following file to your `~/gazebo_tutorial_workspace/scenes` folder.

`shapes_with_tf2_and_wrench.sdf`

### Contents of `shapes_with_tf2_and_wrench.sdf`

```
<?xml version="1.0" ?>
<sdf version="1.11">
    <!--
      Try moving a model using the command in the following CDATA block::
    -->
    <![CDATA[
    gz service -s /world/shapes/set_pose \
               --reqtype gz.msgs.Pose --reptype gz.msgs.Boolean \
               --timeout 300 --req 'name: "box", position: {z: 5.0}'
  ]]>
```

(continues on next page)

(continued from previous page)

```
    <world name="shapes_with_tf2_and_wrench">

        <physics type="ode">
          <max_step_size>0.004</max_step_size>
          <real_time_factor>1.0</real_time_factor>
          <real_time_update_rate>250</real_time_update_rate>
        </physics>
        <plugin name='gz::sim::systems::Physics' filename='gz-sim-physics-system'/>
        <plugin filename="gz-sim-scene-broadcaster-system"  name=
↪"gz::sim::systems::SceneBroadcaster"/>
        <plugin filename="gz-sim-apply-link-wrench-system"  name=
↪"gz::sim::systems::ApplyLinkWrench"/>
        <plugin filename="gz-sim-user-commands-system" name=
↪"gz::sim::systems::UserCommands"/>

        <scene>
            <ambient>1.0 1.0 1.0</ambient>
            <background>0.8 0.8 0.8</background>
        </scene>

        <light type="directional" name="sun">
            <cast_shadows>true</cast_shadows>
            <pose>0 0 10 0 0 0</pose>
            <diffuse>0.8 0.8 0.8 1</diffuse>
            <specular>0.2 0.2 0.2 1</specular>
            <attenuation>
                <range>1000</range>
                <constant>0.9</constant>
                <linear>0.01</linear>
                <quadratic>0.001</quadratic>
            </attenuation>
            <direction>-0.5 0.1 -0.9</direction>
        </light>

        <model name="ground_plane">
            <static>true</static>
            <link name="link">
                <collision name="collision">
                    <geometry>
                        <plane>
                            <normal>0 0 1</normal>
                            <size>100 100</size>
                        </plane>
                    </geometry>
                </collision>
                <visual name="visual">
                    <geometry>
                        <plane>
                            <normal>0 0 1</normal>
                            <size>100 100</size>
                        </plane>
                    </geometry>
```

(continues on next page)

(continued from previous page)

```
                    <material>
                        <ambient>0.8 0.8 0.8 1</ambient>
                        <diffuse>0.8 0.8 0.8 1</diffuse>
                        <specular>0.8 0.8 0.8 1</specular>
                    </material>
                </visual>
            </link>
        </model>

        <model name="box">
            <pose>0 0 0.5 0 0 0</pose>
            <link name="box_link">
                <inertial>
                    <inertia>
                        <ixx>0.16666</ixx>
                        <ixy>0</ixy>
                        <ixz>0</ixz>
                        <iyy>0.16666</iyy>
                        <iyz>0</iyz>
                        <izz>0.16666</izz>
                    </inertia>
                    <mass>1.0</mass>
                </inertial>
                <collision name="box_collision">
                    <geometry>
                        <box>
                            <size>1 1 1</size>
                        </box>
                    </geometry>
                </collision>

                <visual name="box_visual">
                    <geometry>
                        <box>
                            <size>1 1 1</size>
                        </box>
                    </geometry>
                    <material>
                        <ambient>1 0 0 1</ambient>
                        <diffuse>1 0 0 1</diffuse>
                        <specular>1 0 0 1</specular>
                    </material>
                </visual>
            </link>

            <plugin
                    filename="gz-sim-pose-publisher-system"
                    name="gz::sim::systems::PosePublisher">
                <use_pose_vector_msg>true</use_pose_vector_msg>
                <publish_link_pose>false</publish_link_pose>
                <publish_collision_pose>false</publish_collision_pose>
                <publish_visual_pose>false</publish_visual_pose>
```

(continues on next page)

(continued from previous page)

```
            <publish_nested_model_pose>true</publish_nested_model_pose>
        </plugin>

    </model>

    <model name="cylinder">
        <pose>0 -1.5 0.5 0 0 0</pose>
        <link name="cylinder_link">
            <inertial>
                <inertia>
                    <ixx>0.1458</ixx>
                    <ixy>0</ixy>
                    <ixz>0</ixz>
                    <iyy>0.1458</iyy>
                    <iyz>0</iyz>
                    <izz>0.125</izz>
                </inertia>
                <mass>1.0</mass>
            </inertial>
            <collision name="cylinder_collision">
                <geometry>
                    <cylinder>
                        <radius>0.5</radius>
                        <length>1.0</length>
                    </cylinder>
                </geometry>
            </collision>

            <visual name="cylinder_visual">
                <geometry>
                    <cylinder>
                        <radius>0.5</radius>
                        <length>1.0</length>
                    </cylinder>
                </geometry>
                <material>
                    <ambient>0 1 0 1</ambient>
                    <diffuse>0 1 0 1</diffuse>
                    <specular>0 1 0 1</specular>
                </material>
            </visual>
        </link>

        <plugin
                filename="gz-sim-pose-publisher-system"
                name="gz::sim::systems::PosePublisher">
            <use_pose_vector_msg>true</use_pose_vector_msg>
            <publish_link_pose>false</publish_link_pose>
            <publish_collision_pose>false</publish_collision_pose>
            <publish_visual_pose>false</publish_visual_pose>
            <publish_nested_model_pose>true</publish_nested_model_pose>
        </plugin>
```

(continues on next page)

(continued from previous page)

```
        </model>

        <model name="sphere">
            <pose>0 1.5 0.5 0 0 0</pose>
            <link name="sphere_link">
                <inertial>
                    <inertia>
                        <ixx>0.1</ixx>
                        <ixy>0</ixy>
                        <ixz>0</ixz>
                        <iyy>0.1</iyy>
                        <iyz>0</iyz>
                        <izz>0.1</izz>
                    </inertia>
                    <mass>1.0</mass>
                </inertial>
                <collision name="sphere_collision">
                    <geometry>
                        <sphere>
                            <radius>0.5</radius>
                        </sphere>
                    </geometry>
                </collision>

                <visual name="sphere_visual">
                    <geometry>
                        <sphere>
                            <radius>0.5</radius>
                        </sphere>
                    </geometry>
                    <material>
                        <ambient>0 0 1 1</ambient>
                        <diffuse>0 0 1 1</diffuse>
                        <specular>0 0 1 1</specular>
                    </material>
                </visual>
            </link>

            <plugin
                    filename="gz-sim-pose-publisher-system"
                    name="gz::sim::systems::PosePublisher">
                <use_pose_vector_msg>true</use_pose_vector_msg>
                <publish_link_pose>false</publish_link_pose>
                <publish_collision_pose>false</publish_collision_pose>
                <publish_visual_pose>false</publish_visual_pose>
                <publish_nested_model_pose>true</publish_nested_model_pose>
            </plugin>

        </model>

        <model name="capsule">
```

(continues on next page)

(continued from previous page)

```
            <pose>0 -3.0 0.5 0 0 0</pose>
            <link name="capsule_link">
                <inertial>
                    <inertia>
                        <ixx>0.074154</ixx>
                        <ixy>0</ixy>
                        <ixz>0</ixz>
                        <iyy>0.074154</iyy>
                        <iyz>0</iyz>
                        <izz>0.018769</izz>
                    </inertia>
                    <mass>1.0</mass>
                </inertial>
                <collision name="capsule_collision">
                    <geometry>
                        <capsule>
                            <radius>0.2</radius>
                            <length>0.6</length>
                        </capsule>
                    </geometry>
                </collision>
                <visual name="capsule_visual">
                    <geometry>
                        <capsule>
                            <radius>0.2</radius>
                            <length>0.6</length>
                        </capsule>
                    </geometry>
                    <material>
                        <ambient>1 1 0 1</ambient>
                        <diffuse>1 1 0 1</diffuse>
                        <specular>1 1 0 1</specular>
                    </material>
                </visual>
            </link>

            <plugin
                    filename="gz-sim-pose-publisher-system"
                    name="gz::sim::systems::PosePublisher">
                <use_pose_vector_msg>true</use_pose_vector_msg>
                <publish_link_pose>false</publish_link_pose>
                <publish_collision_pose>false</publish_collision_pose>
                <publish_visual_pose>false</publish_visual_pose>
                <publish_nested_model_pose>true</publish_nested_model_pose>
            </plugin>

        </model>

        <model name="ellipsoid">
            <pose>0 3.0 0.5 0 0 0</pose>
            <link name="ellipsoid_link">
                <inertial>
```

(continues on next page)

(continued from previous page)

```
                    <inertia>
                        <ixx>0.068</ixx>
                        <ixy>0</ixy>
                        <ixz>0</ixz>
                        <iyy>0.058</iyy>
                        <iyz>0</iyz>
                        <izz>0.026</izz>
                    </inertia>
                    <mass>1.0</mass>
                </inertial>
                <collision name="ellipsoid_collision">
                    <geometry>
                        <ellipsoid>
                            <radii>0.2 0.3 0.5</radii>
                        </ellipsoid>
                    </geometry>
                </collision>
                <visual name="ellipsoid_visual">
                    <geometry>
                        <ellipsoid>
                            <radii>0.2 0.3 0.5</radii>
                        </ellipsoid>
                    </geometry>
                    <material>
                        <ambient>1 0 1 1</ambient>
                        <diffuse>1 0 1 1</diffuse>
                        <specular>1 0 1 1</specular>
                    </material>
                </visual>
            </link>

            <plugin
                    filename="gz-sim-pose-publisher-system"
                    name="gz::sim::systems::PosePublisher">
                <use_pose_vector_msg>true</use_pose_vector_msg>
                <publish_link_pose>false</publish_link_pose>
                <publish_collision_pose>false</publish_collision_pose>
                <publish_visual_pose>false</publish_visual_pose>
                <publish_nested_model_pose>true</publish_nested_model_pose>
            </plugin>

        </model>

        <model name="cone">
            <pose>0 4.5 0.5 0 0 0</pose>
            <link name="cone_link">
                <inertial auto="true">
                    <density>1</density>
                </inertial>
                <collision name="cone_collision">
                    <geometry>
                        <cone>
```

(continues on next page)

(continued from previous page)

```
                        <radius>0.5</radius>
                        <length>1.0</length>
                    </cone>
                </geometry>
            </collision>

            <visual name="cone_visual">
                <geometry>
                    <cone>
                        <radius>0.5</radius>
                        <length>1.0</length>
                    </cone>
                </geometry>
                <material>
                    <ambient>1 0.47 0 1</ambient>
                    <diffuse>1 0.47 0 1</diffuse>
                    <specular>1 0.47 0 1</specular>
                </material>
            </visual>
        </link>

        <plugin
                filename="gz-sim-pose-publisher-system"
                name="gz::sim::systems::PosePublisher">
            <use_pose_vector_msg>true</use_pose_vector_msg>
            <publish_link_pose>false</publish_link_pose>
            <publish_collision_pose>false</publish_collision_pose>
            <publish_visual_pose>false</publish_visual_pose>
            <publish_nested_model_pose>true</publish_nested_model_pose>
        </plugin>

    </model>

</world>


</sdf>
```

We can open this scene in **Gazebo** with the following command.

```
gz sim $HOME/gazebo_tutorial_workspace/scenes/shapes_with_tf2_and_wrench.sdf
```

Using the following command will show the wrench-related topics.

```
gz topic -l | grep /wrench
```

It should result in the following output.

```
/world/shapes_with_tf2_and_wrench/wrench
/world/shapes_with_tf2_and_wrench/wrench/clear
/world/shapes_with_tf2_and_wrench/wrench/persistent
```

The message used in this topic can be found with the following command.

```
gz topic -i -t /world/shapes_with_tf2_and_wrench/wrench
```

It should result in the following output.

```
No publishers on topic [/world/shapes_with_tf2_and_wrench/wrench]
Subscribers [Address, Message Type]:
  tcp://172.16.191.128:37021, gz.msgs.EntityWrench
```

Using the following command will show the pose-related services.

```
gz service -l | grep set_pose
```

It should result in the following output.

```
/world/shapes_with_tf2_and_wrench/set_pose
/world/shapes_with_tf2_and_wrench/set_pose/blocking
/world/shapes_with_tf2_and_wrench/set_pose_vector
/world/shapes_with_tf2_and_wrench/set_pose_vector/blocking
```

The request and response of this service can be found with the following command.

```
gz service -i -s /world/shapes_with_tf2_and_wrench/set_pose
```

It should result in the following output.

```
Service providers [Address, Request Message Type, Response Message Type]:
  tcp://172.16.191.128:35577, gz.msgs.Pose, gz.msgs.Boolean
```

## 44.2 Objective

Our objectives in this section will be as follows.

1. Send **ROS2** messages to the `[...]/wrench` **Gazebo** topic. It uses `gz.msgs.EntityWrench`, therefore the pairing message is `ros_gz_interfaces/msg/EntityWrench`.
2. Use a **ROS2** service to call the `[...]/set_pose` **Gazebo** service. The interface is not listed in the official list, but we will use `ros_gz_interfaces/srv/SetEntityPose`. See the official repository: https://github.com/gazebosim/ros_gz/blob/jazzy/ros_gz_interfaces/srv/SetEntityPose.srv.

The pose messages will be processed by `tf2`.

## 44.3 Create the package

We start by creating a package that depends on the interface packages mentioned above, namely `ros_gz_interfaces` and `tf2_ros`.

```
cd ~/ros2_tutorial_workspace/src
ros2 pkg create python_package_that_uses_gazebo \
--build-type ament_python \
--dependencies rclpy ros_gz_interfaces tf2_ros
```

#### ros2 pkg create output

```
going to create a new package
package name: python_package_that_uses_gazebo
destination directory: /IdeaProjects/ROS2_Tutorial/ros2_tutorial_workspace/src
package format: 3
version: 0.0.0
description: TODO: Package description
maintainer: ['root <murilo.marinho@manchester.ac.uk>']
licenses: ['TODO: License declaration']
build type: ament_python
dependencies: ['rclpy', 'ros_gz_interfaces', 'geometry_msgs', 'std_msgs']
creating folder ./python_package_that_uses_gazebo
creating ./python_package_that_uses_gazebo/package.xml
creating source folder
creating folder ./python_package_that_uses_gazebo/python_package_that_uses_gazebo
creating ./python_package_that_uses_gazebo/setup.py
creating ./python_package_that_uses_gazebo/setup.cfg
creating folder ./python_package_that_uses_gazebo/resource
creating ./python_package_that_uses_gazebo/resource/python_package_that_uses_gazebo
creating ./python_package_that_uses_gazebo/python_package_that_uses_gazebo/__init__.py
creating folder ./python_package_that_uses_gazebo/test
creating ./python_package_that_uses_gazebo/test/test_copyright.py
creating ./python_package_that_uses_gazebo/test/test_flake8.py
creating ./python_package_that_uses_gazebo/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
BSL-1.0
BSD-2.0
BSD-2-Clause
BSD-3-Clause
GPL-3.0-only
LGPL-3.0-only
MIT
MIT-0
```

## 44.4 Files

We will be working on the following files.

```
python_package_that_uses_gazebo/
|-- config_bridge
|   `-- control_shape_thrust.yaml
|-- launch
|   |-- control_shape_thrust_launch.py
|   |-- send_poses_to_gazebo_launch.py
|   `-- send_wrenches_to_gazebo_launch.py
|-- package.xml
|-- python_package_that_uses_gazebo
```

(continues on next page)

(continued from previous page)

```
|   |-- __init__.py
|   |-- control_shape_thrust_node.py
|   |-- send_poses_to_gazebo_node.py
|   `-- send_wrenches_to_gazebo_node.py
|-- resource
|   `-- python_package_that_uses_gazebo
|-- setup.cfg
|-- setup.py
`-- test
    |-- test_copyright.py
    |-- test_flake8.py
    `-- test_pep257.py
```

## 44.5 Sending poses to Gazebo

**Summary**

We will make a node that does the following for us through **ROS2**.

```
gz service -s \
/world/shapes_with_tf2_and_wrench/set_pose \
--reqtype gz.msgs.Pose \
--reptype gz.msgs.Boolean \
--req 'name: "box", position: {x: 0.0, y: 0.0, z: 50}'
```

We'll be able to send poses to **Gazebo** with the following node. It is a relatively simple node with a service client.

`send_poses_to_gazebo_node.py`

```python
import time
from ros_gz_interfaces.srv import SetEntityPose

import rclpy
from rclpy.node import Node

class SendPosesToGazeboNode(Node):
    """A ROS2 Node that sends poses to Gazebo"""

    def __init__(self):
        super().__init__('send_poses_to_gazebo_node')

        self.service_client = self.create_client(
            srv_type=SetEntityPose,
            srv_name='/world/shapes_with_tf2_and_wrench/set_pose')

        while not self.service_client.wait_for_service(timeout_sec=1.0):
            self.get_logger().info(f'service {self.service_client.srv_name} not␣
↪available, waiting...')


```

(continues on next page)

(continued from previous page)

```
    def send_pose_to_gazebo(self):
        """Sample method sending a pose to and entity in Gazebo."""

        request = SetEntityPose.Request()

        # Add the entity name. For this one, Gazebo accepts the name. :)
        request.entity.name = "box"

        # Set the position
        request.pose.position.x = 0.0
        request.pose.position.y = 0.0
        request.pose.position.z = 50.0

        # Set the orientation
        request.pose.orientation.x = 0.0
        request.pose.orientation.y = 0.0
        request.pose.orientation.z = 0.0
        request.pose.orientation.w = 1.0

        return self.service_client.call_async(request)


def main(args=None):
    """
    The main function.
    :param args: Not used directly by the user, but used by ROS2 to configure certain␣
↪aspects of the Node.
    """
    try:
        rclpy.init(args=args)

        node = SendPosesToGazeboNode()
        future = node.send_pose_to_gazebo()
        rclpy.spin_until_future_complete(node, future)

    except KeyboardInterrupt:
        pass
    except Exception as e:
        print(e)



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

There are perhaps only two unfamiliar aspects of this node by now. First, that we need to send the correct entity name for **Gazebo** to know which entity to set the pose. This name can be obtained in **Gazebo**`s GUI.

```
    def send_pose_to_gazebo(self):
        """Sample method sending a pose to and entity in Gazebo."""

        request = SetEntityPose.Request()
```

(continues on next page)

(continued from previous page)

```
        # Add the entity name. For this one, Gazebo accepts the name. :)
        request.entity.name = "box"

        # Set the position
        request.pose.position.x = 0.0
        request.pose.position.y = 0.0
        request.pose.position.z = 50.0

        # Set the orientation
        request.pose.orientation.x = 0.0
        request.pose.orientation.y = 0.0
        request.pose.orientation.z = 0.0
        request.pose.orientation.w = 1.0

        return self.service_client.call_async(request)
```

The second possibly unfamiliar is the choice of `rclpy.spin_until_future_complete` to illustrate calling the service client only once.

```
def main(args=None):
    """
    The main function.
    :param args: Not used directly by the user, but used by ROS2 to configure certain␣
↪aspects of the Node.
    """
    try:
        rclpy.init(args=args)

        node = SendPosesToGazeboNode()
        future = node.send_pose_to_gazebo()
        rclpy.spin_until_future_complete(node, future)

    except KeyboardInterrupt:
        pass
    except Exception as e:
        print(e)
```

**Exercises**

How would you approach this node if you had to, for instance.

- Accept the entity information as a parameter for the node.
- Accept the pose as a parameter from the node.
- Set the pose of multiple entities.

### 44.5.1 The launch file

This node will not work without a pairing `parameter_bridge`. We add the following launch file to the `launch` folder.

`send_poses_to_gazebo_launch.py`

```
from launch import LaunchDescription
from launch_ros.actions import Node


def generate_launch_description():

    node = Node(
            output='screen',
            emulate_tty=True,
            package='python_package_that_uses_gazebo',
            executable='send_poses_to_gazebo_node',
            name='send_poses_to_gazebo_node'
        )

    bridge = Node(
        package='ros_gz_bridge',
        executable='parameter_bridge',
        arguments=['/world/shapes_with_tf2_and_wrench/set_pose@ros_gz_interfaces/srv/SetEntityPose'],
        output='screen'
    )

    return LaunchDescription([
        node,
        bridge
    ])
```

## 44.6 Sending wrenches to Gazebo

**Summary**

We will make a node that does the following for us through **ROS2**.

```
gz topic -t \
/world/shapes_with_tf2_and_wrench/wrench \
-m gz.msgs.EntityWrench \
-p  'entity: {id: 9}, wrench: {force: {x: 1000.0, y: 0.0, z: 0.0}, torque: {x: 0.0, y: 0.0, z: 0.0}}'
```

We'll be able to send poses to **Gazebo** with the following node. It is a relatively simple node with a publisher.

send_wrenches_to_gazebo_node.py

```python
import time
from ros_gz_interfaces.msg import EntityWrench

import rclpy
from rclpy.node import Node

class SendWrenchesToGazeboNode(Node):
    """A ROS2 Node that sends wrenches to Gazebo."""

    def __init__(self):
        super().__init__('send_wrenches_to_gazebo_node')

        self.publisher = self.create_publisher(
            msg_type=EntityWrench,
            topic='/world/shapes_with_tf2_and_wrench/wrench',
            qos_profile=1)

        while not self.count_subscribers('/world/shapes_with_tf2_and_wrench/wrench'):
            print(f"Waiting for subscriber to be connected...")
            time.sleep(1)

    def send_wrench_to_gazebo(self):
        """Basic method publishing wrenches to Gazebo."""

        ew = EntityWrench()

        # Add the entity id. The entity.name is currently ignored by Gazebo.
        ew.entity.id = 9

        # Set the force
        ew.wrench.force.x = 1000.0
        ew.wrench.force.y = 0.0
        ew.wrench.force.z = 0.0

        # Set the torque
        ew.wrench.torque.x = 0.0
        ew.wrench.torque.y = 0.0
        ew.wrench.torque.z = 0.0

        self.get_logger().info(f"This sent entity {ew.entity.name} a wrench with force:"
                               f" {ew.wrench.force} "
                               f"and torque:"
                               f" {ew.wrench.torque}.")


        self.publisher.publish(ew)

```

(continues on next page)

(continued from previous page)

```
48
49  def main(args=None):
50      """
51      The main function.
52      :param args: Not used directly by the user, but used by ROS2 to configure certain␣
    ↪aspects of the Node.
53      """
54      try:
55          rclpy.init(args=args)
56
57          node = SendWrenchesToGazeboNode()
58          node.send_wrench_to_gazebo()
59          rclpy.spin(node)
60
61      except KeyboardInterrupt:
62          pass
63      except Exception as e:
64          print(e)
65
66
67
68  if __name__ == '__main__':
69      main()
```

The only unfamiliar aspect is shown below. The `id` of the **Gazebo** entity must be sent. You can obtain that information from **Gazebo**'s GUI. Note that here, trying to send a `name` instead is inconsequential and **Gazebo** won't recognise it.

```
    def send_wrench_to_gazebo(self):
        """Basic method publishing wrenches to Gazebo."""

        ew = EntityWrench()

        # Add the entity id. The entity.name is currently ignored by Gazebo.
        ew.entity.id = 9

        # Set the force
        ew.wrench.force.x = 1000.0
        ew.wrench.force.y = 0.0
        ew.wrench.force.z = 0.0

        # Set the torque
        ew.wrench.torque.x = 0.0
        ew.wrench.torque.y = 0.0
        ew.wrench.torque.z = 0.0

        self.get_logger().info(f"This sent entity {ew.entity.name} a wrench with force:"
                               f" {ew.wrench.force} "
                               f"and torque:"
                               f" {ew.wrench.torque}.")


        self.publisher.publish(ew)
```

**Wait… what?**

Yes, for wrenches, we need the `id`. For the pose, we use the `name`. Such is life.

**Exercises**

How would you approach this node if you had to, for instance.

- Accept the entity information as a parameter for the node.
- Accept the wrench as a parameter from the node.
- Publish the wrench of multiple entities.
- **Define the wrench based on other information?**
  - The next section shows one example of it.

### 44.6.1 The launch file

This node will not work without a pairing `parameter_bridge`. We add the following launch file to the `launch` folder.

`send_wrenches_to_gazebo_launch.py`

```
from launch import LaunchDescription
from launch_ros.actions import Node


def generate_launch_description():

    node = Node(
            output='screen',
            emulate_tty=True,
            package='python_package_that_uses_gazebo',
            executable='send_wrenches_to_gazebo_node',
            name='send_wrenches_to_gazebo_node'
        )

    bridge = Node(
        package='ros_gz_bridge',
        executable='parameter_bridge',
        arguments=['/world/shapes_with_tf2_and_wrench/wrench@ros_gz_interfaces/msg/
↪EntityWrench]gz.msgs.EntityWrench'],
        output='screen'
    )

    return LaunchDescription([
        node,
        bridge
    ])
```

## 44.7 Controlling thrust of shapes

Lastly, we can have a slightly more complex example. In this example, we read the poses through `tf2`. Then, using that pose information, we publish a wrench. This way, we can attempt to move a shape to a given place in the scene.

This example highlights one possible way of interacting with **Gazebo**.

`control_shape_thrust_node.py`

```
import time
from ros_gz_interfaces.msg import EntityWrench

import rclpy
from rclpy.node import Node

import tf2_ros

class ControlShapeThrustNode(Node):
    """A ROS2 Node that controls the thrust of a shape in Gazebo.
       It will apply force based on the position of an entity."""

    def __init__(self):
        super().__init__('control_shape_thurst_node')

        # Structured easily to be changed into configurable parameters
        self._gazebo_world_name = "shapes_with_tf2_and_wrench"
        self._wrench_topic = f'/world/{self._gazebo_world_name}/wrench'
        self._gazebo_entity_name = "box" # Find this in Gazebo
        self._gazebo_entity_id = 9 # Find this in Gazebo

        # Set up the wrench publisher
        self.wrench_publisher = self.create_publisher(
            msg_type=EntityWrench,
            topic=self._wrench_topic,
            qos_profile=1)

        # This is one way to prevent published messages from being lost, but we don't know what is subscribing.
        while not self.count_subscribers(self._wrench_topic):
            print(f"Waiting for subscriber to be connected to {self._wrench_topic}...")
            time.sleep(1)

        # Setting up the TransformListener.
        self.transform_listener_buffer = tf2_ros.Buffer()
        self.transform_listener = tf2_ros.TransformListener(self.transform_listener_buffer, self)

        # Information about the transform we want to listen to
        self.parent_name = self._gazebo_world_name # It will be populated this way by Gazebo
        self.child_name = self._gazebo_entity_name # It will be populated this way by Gazebo

        self.timer_period: float = 0.001
```

(continues on next page)

(continued from previous page)

```
        self.timer = self.create_timer(self.timer_period, self.timer_callback)

    def compute_control_action(self, current, target, proportional_gain:float = 5.0) ->␣
↪float:
        """A simple proportional controller"""
        error = current - target
        return -proportional_gain * error

    def send_force_to_gazebo(self, force: tuple[float, float, float] = (0, 0, 0)) ->␣
↪None:
        """Basic method to send force to an entity."""

        ew = EntityWrench()

        # Add the entity id. The entity.name is currently ignored by Gazebo.
        ew.entity.id = self._gazebo_entity_id

        # Set the force
        ew.wrench.force.x = force[0]
        ew.wrench.force.y = force[1]
        ew.wrench.force.z = force[2]

        # Set the torque
        ew.wrench.torque.x = 0.0
        ew.wrench.torque.y = 0.0
        ew.wrench.torque.z = 0.0

        self.wrench_publisher.publish(ew)

    def timer_callback(self):
        """The timer callback will look up the transforms and send the next control␣
↪action."""

        try:
            tfs =   self.transform_listener_buffer.lookup_transform(
                    self.parent_name,
                    self.child_name,
                    rclpy.time.Time())

            # A simple proportional controller for the x-axis force
            u = self.compute_control_action(tfs.transform.translation.x, -3.0)

            self.send_force_to_gazebo(
                (u,
                 0.0,
                 0.0)
            )

        except tf2_ros.TransformException as e:

            self.get_logger().warn(
                f'Could not get transform from `{self.parent_name}` to `{self.child_name}
```

(continues on next page)

(continued from previous page)

```
↪`: {e}')


def main(args=None):
    """
    The main function.
    :param args: Not used directly by the user, but used by ROS2 to configure certain␣
↪aspects of the Node.
    """
    try:
        rclpy.init(args=args)

        node = ControlShapeThrustNode()
        rclpy.spin(node)

    except KeyboardInterrupt:
        pass
    except Exception as e:
        print(e)



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

We start by having multiple hard-coded values to configure the node. This is illustrative and because of how this is organised it would be easier to turn them into configurable parameters.

As you have seen in previous examples, `_gazebo_world_name` is used to define the topic name for the wrench and also the parent reference frame for `tf2`. The `_gazebo_entity_name` is used to get the child frame for `tf2`. Lastly, *_gazebo_entity_id* is used to send the wrench to the correct entity.

```
        # Structured easily to be changed into configurable parameters
        self._gazebo_world_name = "shapes_with_tf2_and_wrench"
        self._wrench_topic = f'/world/{self._gazebo_world_name}/wrench'
        self._gazebo_entity_name = "box" # Find this in Gazebo
        self._gazebo_entity_id = 9 # Find this in Gazebo
```

There is also a simple proportional controller in one dimension.

```
    def compute_control_action(self, current, target, proportional_gain:float = 5.0) ->␣
↪float:
        """A simple proportional controller"""
        error = current - target
        return -proportional_gain * error
```

Lastly, in the `timer_callback`, we use the current coordinate value to compute the control action and send it to **Gazebo**.

```
    def timer_callback(self):
        """The timer callback will look up the transforms and send the next control␣
↪action."""
```

(continues on next page)

(continued from previous page)

```
        try:
            tfs =   self.transform_listener_buffer.lookup_transform(
                    self.parent_name,
                    self.child_name,
                    rclpy.time.Time())

            # A simple proportional controller for the x-axis force
            u = self.compute_control_action(tfs.transform.translation.x, -3.0)

            self.send_force_to_gazebo(
                (u,
                 0.0,
                 0.0)
            )

        except tf2_ros.TransformException as e:

            self.get_logger().warn(
                f'Could not get transform from `{self.parent_name}` to `{self.child_name}
↪`: {e}')
```

**Exercises**

How would you approach this node if you had to, for instance.

- Accept relevant information as a parameter for the node.
- Control the shape via thrusts in the 2D plane.
- Add derivative and integral control actions.
- **Control the shape in 3D.**
  - How would the gravity be counterbalanced?

### 44.7.1 The bridge file

The bridge file, added to the folder `config_bridge`, will manage bridging the topics with `tf2` and the wrench.

`control_shape_thrust.yaml`

```
- ros_topic_name: "/clock"
  gz_topic_name: "/clock"
  ros_type_name: "rosgraph_msgs/msg/Clock"
  gz_type_name: "gz.msgs.Clock"
  direction: GZ_TO_ROS

- ros_topic_name: "/tf"
  gz_topic_name: "/model/box/pose"
  ros_type_name: "tf2_msgs/msg/TFMessage"
  gz_type_name: "gz.msgs.Pose_V"
  direction: GZ_TO_ROS

- ros_topic_name: "/world/shapes_with_tf2_and_wrench/wrench"
```

(continues on next page)

(continued from previous page)

```
  gz_topic_name: "/world/shapes_with_tf2_and_wrench/wrench"
  ros_type_name: "ros_gz_interfaces/msg/EntityWrench"
  gz_type_name: "gz.msgs.EntityWrench"
  direction: ROS_TO_GZ
```

### 44.7.2 The launch file

This node will not work without a pairing `parameter_bridge`. We add the following launch file to the `launch` folder.

`control_shape_thrust_launch.py`

```
import os
from ament_index_python.packages import get_package_share_directory
from launch import LaunchDescription
from launch_ros.actions import Node


def generate_launch_description():

    this_package_share_directory = get_package_share_directory('python_package_that_uses_gazebo')

    node = Node(
            output='screen',
            emulate_tty=True,
            package='python_package_that_uses_gazebo',
            executable='control_shape_thrust_node',
            name='control_shape_thrust_node'
        )

    bridge = Node(
        package='ros_gz_bridge',
        executable='parameter_bridge',
        parameters=[{
            'config_file': os.path.join(this_package_share_directory, 'config_bridge', 'control_shape_thrust.yaml')
        }],
        output='screen'
    )

    return LaunchDescription([
        node,
        bridge
    ])
```

## 44.8 Adjusting the `setup.py`

This file will include the directives for all the three nodes, added in `entry_points`. We also have on L15 the directive for the launch files. Lastly, on L16, we have a directive to install the bridge configuration file which is used by `control_shape_thrust_launch.py`.

`setup.py`

```
import os
from glob import glob
from setuptools import find_packages, setup

package_name = 'python_package_that_uses_gazebo'

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']),
        (os.path.join('share', package_name, 'launch'), glob(os.path.join('launch', '*launch.[pxy][yma]*'))),
        (os.path.join('share', package_name, 'config_bridge'), glob(os.path.join('config_bridge', '*.yaml')))
    ],
    install_requires=['setuptools'],
    zip_safe=True,
    maintainer='root',
    maintainer_email='murilo.marinho@manchester.ac.uk',
    description='TODO: Package description',
    license='TODO: License declaration',
    tests_require=['pytest'],
    entry_points={
        'console_scripts': [
            "send_wrenches_to_gazebo_node = python_package_that_uses_gazebo.send_wrenches_to_gazebo_node:main",
            "send_poses_to_gazebo_node = python_package_that_uses_gazebo.send_poses_to_gazebo_node:main",
            "control_shape_thrust_node = python_package_that_uses_gazebo.control_shape_thrust_node:main"
        ],
    },
)
```

## 44.9 Build and source

Before we proceed, let us build and source once.

```
cd ~/ros2_tutorial_workspace
colcon build
source install/setup.bash
```

> **Note**
>
> For additional explanation and troubleshooting tips, see *Always source after you build*.

> **Warning**
>
> **colcon** will *not* work properly if your terminal has an active **venv**.

## 44.10 Testing

We first open and run the simulation.

```
gz sim $HOME/gazebo_tutorial_workspace/scenes/shapes_with_tf2_and_wrench.sdf
```

It is possible to run each of these examples in this particular order and see their effects.

We can run the first example with the following command. Then, stop it with CTRL+C before moving to the following one.

```
ros2 launch python_package_that_uses_gazebo send_poses_to_gazebo_launch.py
```

### Output for send_poses_to_gazebo_launch.py

```
[INFO] [launch]: All log files can be found below /home/murilo/.ros/log/2025-11-12-10-29-
↪03-415524-murilo-VMware20-1-6420
[INFO] [launch]: Default logging verbosity is set to INFO
[INFO] [send_poses_to_gazebo_node-1]: process started with pid [6423]
[INFO] [parameter_bridge-2]: process started with pid [6424]
[parameter_bridge-2] [INFO] [1762943343.652578097] [ros_gz_bridge]: Creating ROS->GZ␣
↪service bridge [/world/shapes_with_tf2_and_wrench/set_pose (ros_gz_interfaces/srv/
↪SetEntityPose -> /)]
[INFO] [send_poses_to_gazebo_node-1]: process has finished cleanly [pid 6423]
^C[WARNING] [launch]: user interrupted with ctrl-c (SIGINT)
[parameter_bridge-2] [INFO] [1762943347.930139121] [rclcpp]: signal_handler(signum=2)
[INFO] [parameter_bridge-2]: process has finished cleanly [pid 6424]
```

We can run the second example with the following command. Then, stop it with CTRL+C before moving to the following one.

```
ros2 launch python_package_that_uses_gazebo send_wrenches_to_gazebo_launch.py
```

### Output for send_wrenches_to_gazebo_launch.py

```
[INFO] [launch]: All log files can be found below /home/murilo/.ros/log/2025-11-12-10-29-
↪35-514265-murilo-VMware20-1-6504
[INFO] [launch]: Default logging verbosity is set to INFO
[INFO] [send_wrenches_to_gazebo_node-1]: process started with pid [6507]
[INFO] [parameter_bridge-2]: process started with pid [6508]
[parameter_bridge-2] [INFO] [1762943375.599973888] [ros_gz_bridge]: Creating ROS->GZ␣
↪Bridge: [/world/shapes_with_tf2_and_wrench/wrench (ros_gz_interfaces/msg/EntityWrench)␣
↪-> /world/shapes_with_tf2_and_wrench/wrench (gz.msgs.EntityWrench)] (Lazy 0)
[send_wrenches_to_gazebo_node-1] Waiting for subscriber to be connected...
[send_wrenches_to_gazebo_node-1] [INFO] [1762943376.818068638] [send_wrenches_to_gazebo_
↪node]: This sent entity  a wrench with force: geometry_msgs.msg.Vector3(x=1000.0, y=0.
↪0, z=0.0) and torque: geometry_msgs.msg.Vector3(x=0.0, y=0.0, z=0.0).
```

(continues on next page)

(continued from previous page)

```
[parameter_bridge-2] [INFO] [1762943376.824228506] [ros_gz_bridge]: Passing message from␣
↪ROS ros_gz_interfaces/msg/EntityWrench to Gazebo gz.msgs.EntityWrench (showing msg␣
↪only once per type)
^C[WARNING] [launch]: user interrupted with ctrl-c (SIGINT)
[parameter_bridge-2] [INFO] [1762943378.394190781] [rclcpp]: signal_handler(signum=2)
[INFO] [parameter_bridge-2]: process has finished cleanly [pid 6508]
[INFO] [send_wrenches_to_gazebo_node-1]: process has finished cleanly [pid 6507]
```

We can run the third example with the following command. Then, stop it with CTRL+C.

```
ros2 launch python_package_that_uses_gazebo control_shape_thrust_launch.py
```

### Output for `control_shape_thrust_launch.py`

```
[INFO] [launch]: All log files can be found below /home/murilo/.ros/log/2025-11-12-10-30-
↪17-199931-murilo-VMware20-1-6604
[INFO] [launch]: Default logging verbosity is set to INFO
/home/murilo/git_another/ROS2_Tutorial/ros2_tutorial_workspace/install/python_package_
↪that_uses_gazebo/share/python_package_that_uses_gazebo/config_bridge/control_shape_
↪thrust.yaml
[INFO] [control_shape_thrust_node-1]: process started with pid [6607]
[INFO] [parameter_bridge-2]: process started with pid [6608]
[control_shape_thrust_node-1] Waiting for subscriber to be connected to /world/shapes_
↪with_tf2_and_wrench/wrench...
[parameter_bridge-2] [INFO] [1762943418.299653851] [ros_gz_bridge]: Creating GZ->ROS␣
↪Bridge: [/clock (gz.msgs.Clock) -> /clock (rosgraph_msgs/msg/Clock)] (Lazy 0)
[parameter_bridge-2] [INFO] [1762943418.304154168] [ros_gz_bridge]: Creating GZ->ROS␣
↪Bridge: [/model/box/pose (gz.msgs.Pose_V) -> /tf (tf2_msgs/msg/TFMessage)] (Lazy 0)
[parameter_bridge-2] [INFO] [1762943418.316157126] [ros_gz_bridge]: Creating ROS->GZ␣
↪Bridge: [/world/shapes_with_tf2_and_wrench/wrench (ros_gz_interfaces/msg/EntityWrench)␣
↪-> /world/shapes_with_tf2_and_wrench/wrench (gz.msgs.EntityWrench)] (Lazy 0)
[parameter_bridge-2] [INFO] [1762943418.596583200] [ros_gz_bridge]: Passing message from␣
↪ROS ros_gz_interfaces/msg/EntityWrench to Gazebo gz.msgs.EntityWrench (showing msg␣
↪only once per type)
^C[WARNING] [launch]: user interrupted with ctrl-c (SIGINT)
[parameter_bridge-2] [INFO] [1762943424.457110149] [rclcpp]: signal_handler(signum=2)
[INFO] [parameter_bridge-2]: process has finished cleanly [pid 6608]
[INFO] [control_shape_thrust_node-1]: process has finished cleanly [pid 6607]
```

This is illustrated in the video below.

CHAPTER

# FORTYFIVE

# ABOUT NAVIGATION

Added in version Jazzy: This section.

**Warning**

This topic is under construction and this might not even be its final form. Please feel free to open an issue if you spot any typos or other problems.

**See also**

Official documentation: https://docs.nav2.org

The concept of navigation is part of a wide range of robotics applications. It can usually be simply put as moving a robot from one pose to another. Although in our human experience such task can be trivially understood, formally transforming that into mathematical descriptions and their suitable implementation is far from trivial.

This section will cover a few navigation concepts, but it is not meant to be complete. Our interest is to use the existing navigation packages in **ROS2**, namely `nav2`, through custom-made nodes. The tools we have seen so far in **ROS2** will allow us to understand and interact with `nav2` packages.

Nonetheless, there are sizable tutorials dedicated exclusively to navigation. This will not be the case here. The focus will be on linking the concepts learned so far and showcasing how they can be combined to interact with existing packages from the community. Conceptually, interacting with existing packages should be easy. However, going beyond using existing examples will require, possibly, all the knowledge acquired in previous sections.

For illustrative purposes, we will also use the `slam_toolbox`, meant for simultaneous localisation and mapping (SLAM). Whenever a concept from SLAM coincides with a navigation topic, this will be highlighted. However, going into detail about SLAM is out of the scope of this tutorial.

CHAPTER

# FORTYSIX

# INSTALLATION

Added in version Jazzy: This section.

**Warning**

This topic is under construction and this might not even be its final form. Please feel free to open an issue if you spot any typos or other problems.

**See also**

Official documentation: https://docs.nav2.org/getting_started/index.html

We will install `nav2`, a few example packages, and the `slam-toolbox` with the following command.

The package `ros-jazzy-navigation2` contains most of `nav2`. However, we have to install `ros-jazzy-nav2-bringup` separately owing to "recursive dependencies". See below all the packages that are installed with `ros-jazzy-navigation2`.

### Contents of `ros-jazzy-navigation2`

```
<!-- nav2_bringup doesn't belong for recursive dependencies -->
<exec_depend>nav2_amcl</exec_depend>
<exec_depend>nav2_behavior_tree</exec_depend>
<exec_depend>nav2_bt_navigator</exec_depend>
<exec_depend>nav2_collision_monitor</exec_depend>
<exec_depend>nav2_constrained_smoother</exec_depend>
<exec_depend>nav2_controller</exec_depend>
<exec_depend>nav2_core</exec_depend>
<exec_depend>nav2_costmap_2d</exec_depend>
<exec_depend>nav2_dwb_controller</exec_depend>
<exec_depend>nav2_graceful_controller</exec_depend>
<exec_depend>nav2_lifecycle_manager</exec_depend>
<exec_depend>nav2_map_server</exec_depend>
<exec_depend>nav2_msgs</exec_depend>
<exec_depend>nav2_mppi_controller</exec_depend>
<exec_depend>nav2_navfn_planner</exec_depend>
<exec_depend>nav2_planner</exec_depend>
<exec_depend>nav2_behaviors</exec_depend>
<exec_depend>nav2_smoother</exec_depend>
```

(continues on next page)

(continued from previous page)

```
<exec_depend>nav2_regulated_pure_pursuit_controller</exec_depend>
<exec_depend>nav2_route</exec_depend>
<exec_depend>nav2_rotation_shim_controller</exec_depend>
<exec_depend>nav2_rviz_plugins</exec_depend>
<exec_depend>nav2_simple_commander</exec_depend>
<exec_depend>nav2_smac_planner</exec_depend>
<exec_depend>nav2_smoother</exec_depend>
<exec_depend>nav2_theta_star_planner</exec_depend>
<exec_depend>nav2_util</exec_depend>
<exec_depend>nav2_velocity_smoother</exec_depend>
<exec_depend>nav2_voxel_grid</exec_depend>
<exec_depend>nav2_waypoint_follower</exec_depend>
<exec_depend>opennav_docking</exec_depend>
<exec_depend>opennav_docking_bt</exec_depend>
<exec_depend>opennav_docking_core</exec_depend>
```

The last package related to `nav2`, `ros-jazzy-nav2-minimal-tb*` has a wildcard that will expand, currently, to install the following packages. These are optional packages that will give us some infrastructure to work on the tutorial.

- ros-jazzy-nav2-minimal-tb3-sim - Nav2 Minimum TurtleBot3 Simulation
- ros-jazzy-nav2-minimal-tb4-description - Nav2's minimum Turtlebot4 Description package
- ros-jazzy-nav2-minimal-tb4-sim - Nav2 Minimum TurtleBot4 Simulation

For illustrative purposes, we also add `ros-jazzy-slam-toolbox`. Please note that navigation is complementary to localisation and mapping. Both can be done in isolation too.

**References**

- https://github.com/ros-navigation/navigation2/tree/jazzy
- https://github.com/ros-navigation/navigation2/tree/jazzy/navigation2
- https://github.com/ros-navigation/nav2_minimal_turtlebot_simulation/tree/jazzy
- https://github.com/ros-navigation/navigation2/tree/jazzy/nav2_bringup
- https://github.com/SteveMacenski/slam_toolbox/tree/jazzy
- https://docs.nav2.org/concepts/index.html
- https://docs.nav2.org/setup_guides/transformation/setup_transforms.html
- https://docs.nav2.org/setup_guides/sdf/setup_sdf.html
- https://docs.nav2.org/setup_guides/odom/setup_odom_gz.html

CHAPTER

# FORTYSEVEN

# USING NAV2

Added in version Jazzy: This section.

**Warning**

This topic is under construction and this might not even be its final form. Please feel free to open an issue if you spot any typos or other problems.

There is a plethora of online examples of `nav2`, with varying levels of detail and quality. My approach will top-down, where we will see how illustrative examples and then walk through their contents.

## 47.1 Navigation on a known map

**See also**

Official documentation: https://docs.nav2.org/tutorials/docs/navigation2_on_real_turtlebot3.html

In this example, our interest is looking at how navigation can be done with built-in tools, using a known *map*. A map will have many definitions and the literature about it is extensive. For now, we can think of the map as something usual. It points out where things are, for instance, where passable regions (such as roads) are, and possibly hazards, obstacles, and other relevant objects.

In this example, a TurtleBot3 will be used. As part of `nav2_bringup`, there is a rather complete example that we can utilize, namely `tb3_simulation_launch.py`. The example can be executed with the following command.

```
ros2 launch nav2_bringup \
tb3_simulation_launch.py \
use_sim_time:=True \
headless:=False \
sigterm_timeout:=120
```

**Important**

Differently from everything else used in this tutorial, this `tb3_simulation_launch.py` demo does not always finishes cleanly on my machines. It has hanged sometimes, sometimes **`Gazebo`** does not show, and sometimes it does not shutdown properly.

It is highly recommended to use a docker container, such as https://github.com/UoMMScRobotics/sfr_gazebo_nav2, to minimise disappointment.

**Warning**

The `sigterm_timeout` flag is particularly important. Given that some devices we use might not be powerful enough for all processes to finish cleanly within 5 or 10 seconds, we should add this to prevent a `SIGTERM` from being sent to the nodes. If nodes are not terminated correctly they can leave connections open, linger indefinitely, and cause extremely difficult-to-debug situations. This can also be dangerous when real robots are used.

We use `headless:=False` for the launch file to spawn **`Gazebo`**. We use `use_sim_time:=True` to make sure that **`Gazebo`**'s clock will be used an prevent timing issues with */tf*.

This example will create a large number of nodes and open two screens. One of these will be for **`Gazebo`** and another for **`rviz2`**. In this example, there are two actions expected from the user.

1. Set the initial *2D Pose Estimate*. This can be done through **`rviz2`**, or **`ROS2`** interfaces.
2. Send one or more *Nav2 Goal* s.

An example of how to do so is shown in the video below. Please note that although I adjust **`Gazebo`** to line up with the **`rviz2`** view, this is for my convenience. As long as your initial estimate is fairly accurate the navigation should work fine.

**Warning**

The `jazzy` version of this example shows a number of errors when shutting down. This might cause issues when the example is run repeatedly.

### 47.1.1 Unpacking the example

Locally, you can navigate to the package using the following command.

```
cd $(ros2 pkg prefix nav2_bringup --share)
```

The installed folder will have the following structure.

```
nav2_bringup/
├── cmake
│   ├── nav2_bringupConfig.cmake
│   └── nav2_bringupConfig-version.cmake
├── environment
│   ├── ament_prefix_path.dsv
│   ├── ament_prefix_path.sh
│   ├── path.dsv
│   └── path.sh
├── launch
│   ├── bringup_launch.py
│   ├── cloned_multi_tb3_simulation_launch.py
│   ├── localization_launch.py
│   ├── navigation_launch.py
│   ├── rviz_launch.py
│   ├── slam_launch.py
│   ├── tb3_loopback_simulation.launch.py
│   ├── tb3_simulation_launch.py
│   ├── tb4_loopback_simulation.launch.py
```

(continues on next page)

(continued from previous page)

```
    ├── tb4_simulation_launch.py
    └── unique_multi_tb3_simulation_launch.py
├── local_setup.bash
├── local_setup.dsv
├── local_setup.sh
├── local_setup.zsh
├── maps
    ├── depot.pgm
    ├── depot.yaml
    ├── tb3_sandbox.pgm
    ├── tb3_sandbox.yaml
    ├── warehouse.pgm
    └── warehouse.yaml
├── package.dsv
├── package.xml
├── params
    ├── nav2_multirobot_params_1.yaml
    ├── nav2_multirobot_params_2.yaml
    ├── nav2_multirobot_params_all.yaml
    └── nav2_params.yaml
└── rviz
    ├── nav2_default_view.rviz
    └── nav2_namespaced_view.rviz
```

The launch file we executed in this example is shown below. A launch file that is general will tend to have a proportional level of complexity. We are not currently interested in dissecting all elements of this file. Rather, our interest is to take a look at the relevant files used and what they mean. By understanding these, we would be a step closer to be able to modify the example or interact with it for your own purposes.

### Contents of `tb3_simulation_launch.py`

```
# Copyright (C) 2023 Open Source Robotics Foundation
#
# Licensed under the Apache License, Version 2.0 (the "License");
# you may not use this file except in compliance with the License.
# You may obtain a copy of the License at
#
#     http://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
# See the License for the specific language governing permissions and
# limitations under the License.

"""This is all-in-one launch script intended for use by nav2 developers."""

import os
import tempfile

from ament_index_python.packages import get_package_share_directory

```

(continues on next page)

(continued from previous page)

```
from launch import LaunchDescription
from launch.actions import (
    DeclareLaunchArgument,
    ExecuteProcess,
    IncludeLaunchDescription,
    OpaqueFunction,
    RegisterEventHandler,
)
from launch.conditions import IfCondition
from launch.event_handlers import OnShutdown
from launch.launch_description_sources import PythonLaunchDescriptionSource
from launch.substitutions import LaunchConfiguration, PythonExpression

from launch_ros.actions import Node


def generate_launch_description():
    # Get the launch directory
    bringup_dir = get_package_share_directory('nav2_bringup')
    launch_dir = os.path.join(bringup_dir, 'launch')
    sim_dir = get_package_share_directory('nav2_minimal_tb3_sim')

    # Create the launch configuration variables
    slam = LaunchConfiguration('slam')
    namespace = LaunchConfiguration('namespace')
    use_namespace = LaunchConfiguration('use_namespace')
    map_yaml_file = LaunchConfiguration('map')
    use_sim_time = LaunchConfiguration('use_sim_time')
    params_file = LaunchConfiguration('params_file')
    autostart = LaunchConfiguration('autostart')
    use_composition = LaunchConfiguration('use_composition')
    use_respawn = LaunchConfiguration('use_respawn')

    # Launch configuration variables specific to simulation
    rviz_config_file = LaunchConfiguration('rviz_config_file')
    use_simulator = LaunchConfiguration('use_simulator')
    use_robot_state_pub = LaunchConfiguration('use_robot_state_pub')
    use_rviz = LaunchConfiguration('use_rviz')
    headless = LaunchConfiguration('headless')
    world = LaunchConfiguration('world')
    pose = {
        'x': LaunchConfiguration('x_pose', default='-2.00'),
        'y': LaunchConfiguration('y_pose', default='-0.50'),
        'z': LaunchConfiguration('z_pose', default='0.01'),
        'R': LaunchConfiguration('roll', default='0.00'),
        'P': LaunchConfiguration('pitch', default='0.00'),
        'Y': LaunchConfiguration('yaw', default='0.00'),
    }
    robot_name = LaunchConfiguration('robot_name')
    robot_sdf = LaunchConfiguration('robot_sdf')

    remappings = [('/tf', 'tf'), ('/tf_static', 'tf_static')]
```

(continues on next page)

(continued from previous page)

```
    # Declare the launch arguments
    declare_namespace_cmd = DeclareLaunchArgument(
        'namespace', default_value='', description='Top-level namespace'
    )

    declare_use_namespace_cmd = DeclareLaunchArgument(
        'use_namespace',
        default_value='false',
        description='Whether to apply a namespace to the navigation stack',
    )

    declare_slam_cmd = DeclareLaunchArgument(
        'slam', default_value='False', description='Whether run a SLAM'
    )

    declare_map_yaml_cmd = DeclareLaunchArgument(
        'map',
        default_value=os.path.join(bringup_dir, 'maps', 'tb3_sandbox.yaml'),
    )

    declare_use_sim_time_cmd = DeclareLaunchArgument(
        'use_sim_time',
        default_value='true',
        description='Use simulation (Gazebo) clock if true',
    )

    declare_params_file_cmd = DeclareLaunchArgument(
        'params_file',
        default_value=os.path.join(bringup_dir, 'params', 'nav2_params.yaml'),
        description='Full path to the ROS2 parameters file to use for all launched nodes
↪',
    )

    declare_autostart_cmd = DeclareLaunchArgument(
        'autostart',
        default_value='true',
        description='Automatically startup the nav2 stack',
    )

    declare_use_composition_cmd = DeclareLaunchArgument(
        'use_composition',
        default_value='True',
        description='Whether to use composed bringup',
    )

    declare_use_respawn_cmd = DeclareLaunchArgument(
        'use_respawn',
        default_value='False',
        description='Whether to respawn if a node crashes. Applied when composition is␣
↪disabled.',
    )
```

(continues on next page)

(continued from previous page)

```
    declare_rviz_config_file_cmd = DeclareLaunchArgument(
        'rviz_config_file',
        default_value=os.path.join(bringup_dir, 'rviz', 'nav2_default_view.rviz'),
        description='Full path to the RVIZ config file to use',
    )

    declare_use_simulator_cmd = DeclareLaunchArgument(
        'use_simulator',
        default_value='True',
        description='Whether to start the simulator',
    )

    declare_use_robot_state_pub_cmd = DeclareLaunchArgument(
        'use_robot_state_pub',
        default_value='True',
        description='Whether to start the robot state publisher',
    )

    declare_use_rviz_cmd = DeclareLaunchArgument(
        'use_rviz', default_value='True', description='Whether to start RVIZ'
    )

    declare_simulator_cmd = DeclareLaunchArgument(
        'headless', default_value='True', description='Whether to execute gzclient)'
    )

    declare_world_cmd = DeclareLaunchArgument(
        'world',
        default_value=os.path.join(sim_dir, 'worlds', 'tb3_sandbox.sdf.xacro'),
        description='Full path to world model file to load',
    )

    declare_robot_name_cmd = DeclareLaunchArgument(
        'robot_name', default_value='turtlebot3_waffle', description='name of the robot'
    )

    declare_robot_sdf_cmd = DeclareLaunchArgument(
        'robot_sdf',
        default_value=os.path.join(sim_dir, 'urdf', 'gz_waffle.sdf.xacro'),
        description='Full path to robot sdf file to spawn the robot in gazebo',
    )

    urdf = os.path.join(sim_dir, 'urdf', 'turtlebot3_waffle.urdf')
    with open(urdf, 'r') as infp:
        robot_description = infp.read()

    start_robot_state_publisher_cmd = Node(
        condition=IfCondition(use_robot_state_pub),
        package='robot_state_publisher',
        executable='robot_state_publisher',
        name='robot_state_publisher',
```

(continues on next page)

(continued from previous page)

```
        namespace=namespace,
        output='screen',
        parameters=[
            {'use_sim_time': use_sim_time, 'robot_description': robot_description}
        ],
        remappings=remappings,
    )

    rviz_cmd = IncludeLaunchDescription(
        PythonLaunchDescriptionSource(os.path.join(launch_dir, 'rviz_launch.py')),
        condition=IfCondition(use_rviz),
        launch_arguments={
            'namespace': namespace,
            'use_namespace': use_namespace,
            'use_sim_time': use_sim_time,
            'rviz_config': rviz_config_file,
        }.items(),
    )

    bringup_cmd = IncludeLaunchDescription(
        PythonLaunchDescriptionSource(os.path.join(launch_dir, 'bringup_launch.py')),
        launch_arguments={
            'namespace': namespace,
            'use_namespace': use_namespace,
            'slam': slam,
            'map': map_yaml_file,
            'use_sim_time': use_sim_time,
            'params_file': params_file,
            'autostart': autostart,
            'use_composition': use_composition,
            'use_respawn': use_respawn,
        }.items(),
    )
    # The SDF file for the world is a xacro file because we wanted to
    # conditionally load the SceneBroadcaster plugin based on wheter we're
    # running in headless mode. But currently, the Gazebo command line doesn't
    # take SDF strings for worlds, so the output of xacro needs to be saved into
    # a temporary file and passed to Gazebo.
    world_sdf = tempfile.mktemp(prefix='nav2_', suffix='.sdf')
    world_sdf_xacro = ExecuteProcess(
        cmd=['xacro', '-o', world_sdf, ['headless:=', headless], world])
    gazebo_server = ExecuteProcess(
        cmd=['gz', 'sim', '-r', '-s', world_sdf],
        output='screen',
        condition=IfCondition(use_simulator)
    )

    remove_temp_sdf_file = RegisterEventHandler(event_handler=OnShutdown(
        on_shutdown=[
            OpaqueFunction(function=lambda _: os.remove(world_sdf))
        ]))

```

(continues on next page)

(continued from previous page)

```
    gazebo_client = IncludeLaunchDescription(
        PythonLaunchDescriptionSource(
            os.path.join(get_package_share_directory('ros_gz_sim'),
                         'launch',
                         'gz_sim.launch.py')
        ),
        condition=IfCondition(PythonExpression(
            [use_simulator, ' and not ', headless])),
        launch_arguments={'gz_args': ['-v4 -g ']}.items(),
    )

    gz_robot = IncludeLaunchDescription(
        PythonLaunchDescriptionSource(
            os.path.join(sim_dir, 'launch', 'spawn_tb3.launch.py')),
        launch_arguments={'namespace': namespace,
                          'use_sim_time': use_sim_time,
                          'robot_name': robot_name,
                          'robot_sdf': robot_sdf,
                          'x_pose': pose['x'],
                          'y_pose': pose['y'],
                          'z_pose': pose['z'],
                          'roll': pose['R'],
                          'pitch': pose['P'],
                          'yaw': pose['Y']}.items())

    # Create the launch description and populate
    ld = LaunchDescription()

    # Declare the launch options
    ld.add_action(declare_namespace_cmd)
    ld.add_action(declare_use_namespace_cmd)
    ld.add_action(declare_slam_cmd)
    ld.add_action(declare_map_yaml_cmd)
    ld.add_action(declare_use_sim_time_cmd)
    ld.add_action(declare_params_file_cmd)
    ld.add_action(declare_autostart_cmd)
    ld.add_action(declare_use_composition_cmd)

    ld.add_action(declare_rviz_config_file_cmd)
    ld.add_action(declare_use_simulator_cmd)
    ld.add_action(declare_use_robot_state_pub_cmd)
    ld.add_action(declare_use_rviz_cmd)
    ld.add_action(declare_simulator_cmd)
    ld.add_action(declare_world_cmd)
    ld.add_action(declare_robot_name_cmd)
    ld.add_action(declare_robot_sdf_cmd)
    ld.add_action(declare_use_respawn_cmd)

    ld.add_action(world_sdf_xacro)
    ld.add_action(remove_temp_sdf_file)
    ld.add_action(gz_robot)
    ld.add_action(gazebo_server)
```

(continues on next page)

(continued from previous page)

```
    ld.add_action(gazebo_client)

    # Add the actions to launch all of the navigation nodes
    ld.add_action(start_robot_state_publisher_cmd)
    ld.add_action(rviz_cmd)
    ld.add_action(bringup_cmd)

    return ld
```

The highlighted lines point out to these important files. These will guide the upcoming discussion.

- `params/nav2_params.yaml`
- `maps/tb3_sandbox.yaml`
- `worlds/tb3_sandbox.sdf.xacro`
- `urdf/gz_waffle.sdf.xacro`
- `urdf/turtlebot3_waffle.urdf`

### 47.1.2 Navigation

**See also**

Official documentation: https://docs.nav2.org/configuration/index.html

In the example, the navigation is defined in the following configuration file.

#### Contents of `nav2_params.yaml`

```yaml
amcl:
  ros__parameters:
    alpha1: 0.2
    alpha2: 0.2
    alpha3: 0.2
    alpha4: 0.2
    alpha5: 0.2
    base_frame_id: "base_footprint"
    beam_skip_distance: 0.5
    beam_skip_error_threshold: 0.9
    beam_skip_threshold: 0.3
    do_beamskip: false
    global_frame_id: "map"
    lambda_short: 0.1
    laser_likelihood_max_dist: 2.0
    laser_max_range: 100.0
    laser_min_range: -1.0
    laser_model_type: "likelihood_field"
    max_beams: 60
    max_particles: 2000
    min_particles: 500
    odom_frame_id: "odom"
```

(continues on next page)

(continued from previous page)

```
    pf_err: 0.05
    pf_z: 0.99
    recovery_alpha_fast: 0.0
    recovery_alpha_slow: 0.0
    resample_interval: 1
    robot_model_type: "nav2_amcl::DifferentialMotionModel"
    save_pose_rate: 0.5
    sigma_hit: 0.2
    tf_broadcast: true
    transform_tolerance: 1.0
    update_min_a: 0.2
    update_min_d: 0.25
    z_hit: 0.5
    z_max: 0.05
    z_rand: 0.5
    z_short: 0.05
    scan_topic: scan

bt_navigator:
  ros__parameters:
    global_frame: map
    robot_base_frame: base_link
    odom_topic: /odom
    bt_loop_duration: 10
    default_server_timeout: 20
    wait_for_service_timeout: 1000
    action_server_result_timeout: 900.0
    navigators: ["navigate_to_pose", "navigate_through_poses"]
    navigate_to_pose:
      plugin: "nav2_bt_navigator::NavigateToPoseNavigator"
    navigate_through_poses:
      plugin: "nav2_bt_navigator::NavigateThroughPosesNavigator"
    # 'default_nav_through_poses_bt_xml' and 'default_nav_to_pose_bt_xml' are use defaults:
    # nav2_bt_navigator/navigate_to_pose_w_replanning_and_recovery.xml
    # nav2_bt_navigator/navigate_through_poses_w_replanning_and_recovery.xml
    # They can be set here or via a RewrittenYaml remap from a parent launch file to␣
↪Nav2.

    # plugin_lib_names is used to add custom BT plugins to the executor (vector of␣
↪strings).
    # Built-in plugins are added automatically
    # plugin_lib_names: []

    error_code_names:
      - compute_path_error_code
      - follow_path_error_code

controller_server:
  ros__parameters:
    controller_frequency: 20.0
    costmap_update_timeout: 0.30
    min_x_velocity_threshold: 0.001
```

(continues on next page)

(continued from previous page)

```
    min_y_velocity_threshold: 0.5
    min_theta_velocity_threshold: 0.001
    failure_tolerance: 0.3
    progress_checker_plugins: ["progress_checker"]
    goal_checker_plugins: ["general_goal_checker"] # "precise_goal_checker"
    controller_plugins: ["FollowPath"]
    use_realtime_priority: false

    # Progress checker parameters
    progress_checker:
      plugin: "nav2_controller::SimpleProgressChecker"
      required_movement_radius: 0.5
      movement_time_allowance: 10.0
    # Goal checker parameters
    #precise_goal_checker:
    #  plugin: "nav2_controller::SimpleGoalChecker"
    #  xy_goal_tolerance: 0.25
    #  yaw_goal_tolerance: 0.25
    #  stateful: True
    general_goal_checker:
      stateful: True
      plugin: "nav2_controller::SimpleGoalChecker"
      xy_goal_tolerance: 0.25
      yaw_goal_tolerance: 0.25
    FollowPath:
      plugin: "nav2_mppi_controller::MPPIController"
      time_steps: 56
      model_dt: 0.05
      batch_size: 2000
      ax_max: 3.0
      ax_min: -3.0
      ay_max: 3.0
      ay_min: -3.0
      az_max: 3.5
      vx_std: 0.2
      vy_std: 0.2
      wz_std: 0.4
      vx_max: 0.5
      vx_min: -0.35
      vy_max: 0.5
      wz_max: 1.9
      iteration_count: 1
      prune_distance: 1.7
      transform_tolerance: 0.1
      temperature: 0.3
      gamma: 0.015
      motion_model: "DiffDrive"
      visualize: true
      regenerate_noises: true
      TrajectoryVisualizer:
        trajectory_step: 5
        time_step: 3
```

(continues on next page)

(continued from previous page)

```
      AckermannConstraints:
        min_turning_r: 0.2
      critics: [
        "ConstraintCritic", "CostCritic", "GoalCritic",
        "GoalAngleCritic", "PathAlignCritic", "PathFollowCritic",
        "PathAngleCritic", "PreferForwardCritic"]
      ConstraintCritic:
        enabled: true
        cost_power: 1
        cost_weight: 4.0
      GoalCritic:
        enabled: true
        cost_power: 1
        cost_weight: 5.0
        threshold_to_consider: 1.4
      GoalAngleCritic:
        enabled: true
        cost_power: 1
        cost_weight: 3.0
        threshold_to_consider: 0.5
      PreferForwardCritic:
        enabled: true
        cost_power: 1
        cost_weight: 5.0
        threshold_to_consider: 0.5
      CostCritic:
        enabled: true
        cost_power: 1
        cost_weight: 3.81
        near_collision_cost: 253
        critical_cost: 300.0
        consider_footprint: false
        collision_cost: 1000000.0
        near_goal_distance: 1.0
        trajectory_point_step: 2
      PathAlignCritic:
        enabled: true
        cost_power: 1
        cost_weight: 14.0
        max_path_occupancy_ratio: 0.05
        trajectory_point_step: 4
        threshold_to_consider: 0.5
        offset_from_furthest: 20
        use_path_orientations: false
      PathFollowCritic:
        enabled: true
        cost_power: 1
        cost_weight: 5.0
        offset_from_furthest: 5
        threshold_to_consider: 1.4
      PathAngleCritic:
        enabled: true
```

(continues on next page)

(continued from previous page)

```
        cost_power: 1
        cost_weight: 2.0
        offset_from_furthest: 4
        threshold_to_consider: 0.5
        max_angle_to_furthest: 1.0
        mode: 0
      # TwirlingCritic:
      #   enabled: true
      #   twirling_cost_power: 1
      #   twirling_cost_weight: 10.0

local_costmap:
  local_costmap:
    ros__parameters:
      update_frequency: 5.0
      publish_frequency: 2.0
      global_frame: odom
      robot_base_frame: base_link
      rolling_window: true
      width: 3
      height: 3
      resolution: 0.05
      robot_radius: 0.22
      plugins: ["voxel_layer", "inflation_layer"]
      inflation_layer:
        plugin: "nav2_costmap_2d::InflationLayer"
        cost_scaling_factor: 3.0
        inflation_radius: 0.70
      voxel_layer:
        plugin: "nav2_costmap_2d::VoxelLayer"
        enabled: True
        publish_voxel_map: True
        origin_z: 0.0
        z_resolution: 0.05
        z_voxels: 16
        max_obstacle_height: 2.0
        mark_threshold: 0
        observation_sources: scan
        scan:
          topic: /scan
          max_obstacle_height: 2.0
          clearing: True
          marking: True
          data_type: "LaserScan"
          raytrace_max_range: 3.0
          raytrace_min_range: 0.0
          obstacle_max_range: 2.5
          obstacle_min_range: 0.0
      static_layer:
        plugin: "nav2_costmap_2d::StaticLayer"
        map_subscribe_transient_local: True
      always_send_full_costmap: True
```

(continues on next page)

(continued from previous page)

```
global_costmap:
  global_costmap:
    ros__parameters:
      update_frequency: 1.0
      publish_frequency: 1.0
      global_frame: map
      robot_base_frame: base_link
      robot_radius: 0.22
      resolution: 0.05
      track_unknown_space: true
      plugins: ["static_layer", "obstacle_layer", "inflation_layer"]
      obstacle_layer:
        plugin: "nav2_costmap_2d::ObstacleLayer"
        enabled: True
        observation_sources: scan
        scan:
          topic: /scan
          max_obstacle_height: 2.0
          clearing: True
          marking: True
          data_type: "LaserScan"
          raytrace_max_range: 3.0
          raytrace_min_range: 0.0
          obstacle_max_range: 2.5
          obstacle_min_range: 0.0
      static_layer:
        plugin: "nav2_costmap_2d::StaticLayer"
        map_subscribe_transient_local: True
      inflation_layer:
        plugin: "nav2_costmap_2d::InflationLayer"
        cost_scaling_factor: 3.0
        inflation_radius: 0.7
      always_send_full_costmap: True

# The yaml_filename does not need to be specified since it going to be set by defaults␣
↪in launch.
# If you'd rather set it in the yaml, remove the default "map" value in the tb3_
↪simulation_launch.py
# file & provide full path to map below. If CLI map configuration or launch default is␣
↪provided, that will be used.
# map_server:
#   ros__parameters:
#     yaml_filename: ""

map_saver:
  ros__parameters:
    save_map_timeout: 5.0
    free_thresh_default: 0.25
    occupied_thresh_default: 0.65
    map_subscribe_transient_local: True

```

(continues on next page)

(continued from previous page)

```yaml
planner_server:
  ros__parameters:
    expected_planner_frequency: 20.0
    planner_plugins: ["GridBased"]
    costmap_update_timeout: 1.0
    GridBased:
      plugin: "nav2_navfn_planner::NavfnPlanner"
      tolerance: 0.5
      use_astar: false
      allow_unknown: true

smoother_server:
  ros__parameters:
    smoother_plugins: ["simple_smoother"]
    simple_smoother:
      plugin: "nav2_smoother::SimpleSmoother"
      tolerance: 1.0e-10
      max_its: 1000
      do_refinement: True

behavior_server:
  ros__parameters:
    local_costmap_topic: local_costmap/costmap_raw
    global_costmap_topic: global_costmap/costmap_raw
    local_footprint_topic: local_costmap/published_footprint
    global_footprint_topic: global_costmap/published_footprint
    cycle_frequency: 10.0
    behavior_plugins: ["spin", "backup", "drive_on_heading", "assisted_teleop", "wait"]
    spin:
      plugin: "nav2_behaviors::Spin"
    backup:
      plugin: "nav2_behaviors::BackUp"
    drive_on_heading:
      plugin: "nav2_behaviors::DriveOnHeading"
    wait:
      plugin: "nav2_behaviors::Wait"
    assisted_teleop:
      plugin: "nav2_behaviors::AssistedTeleop"
    local_frame: odom
    global_frame: map
    robot_base_frame: base_link
    transform_tolerance: 0.1
    simulate_ahead_time: 2.0
    max_rotational_vel: 1.0
    min_rotational_vel: 0.4
    rotational_acc_lim: 3.2

waypoint_follower:
  ros__parameters:
    loop_rate: 20
    stop_on_failure: false
    action_server_result_timeout: 900.0
```

(continues on next page)

(continued from previous page)

```
    waypoint_task_executor_plugin: "wait_at_waypoint"
    wait_at_waypoint:
      plugin: "nav2_waypoint_follower::WaitAtWaypoint"
      enabled: True
      waypoint_pause_duration: 200

route_server:
  ros__parameters:
    # The graph_filepath does not need to be specified since it going to be set by defaults in launch.
    # If you'd rather set it in the yaml, remove the default "graph" value in the launch file(s).
    # file & provide full path to map below. If graph config or launch default is provided, it is used
    # graph_filepath: $(find-pkg-share nav2_route)/graphs/aws_graph.geojson
    boundary_radius_to_achieve_node: 1.0
    radius_to_achieve_node: 2.0
    smooth_corners: true
    operations: ["AdjustSpeedLimit", "ReroutingService", "CollisionMonitor"]
    ReroutingService:
      plugin: "nav2_route::ReroutingService"
    AdjustSpeedLimit:
      plugin: "nav2_route::AdjustSpeedLimit"
    CollisionMonitor:
      plugin: "nav2_route::CollisionMonitor"
      max_collision_dist: 3.0
    edge_cost_functions: ["DistanceScorer", "CostmapScorer"]
    DistanceScorer:
      plugin: "nav2_route::DistanceScorer"
    CostmapScorer:
      plugin: "nav2_route::CostmapScorer"

velocity_smoother:
  ros__parameters:
    smoothing_frequency: 20.0
    stamp_smoothed_velocity_with_smoothing_time: False
    scale_velocities: False
    feedback: "OPEN_LOOP"
    max_velocity: [0.5, 0.0, 2.0]
    min_velocity: [-0.5, 0.0, -2.0]
    max_accel: [2.5, 0.0, 3.2]
    max_decel: [-2.5, 0.0, -3.2]
    odom_topic: "odom"
    odom_duration: 0.1
    deadband_velocity: [0.0, 0.0, 0.0]
    velocity_timeout: 1.0

collision_monitor:
  ros__parameters:
    base_frame_id: "base_footprint"
    odom_frame_id: "odom"
    cmd_vel_in_topic: "cmd_vel_smoothed"
```

(continues on next page)

(continued from previous page)

```
    cmd_vel_out_topic: "cmd_vel"
    state_topic: "collision_monitor_state"
    transform_tolerance: 0.2
    source_timeout: 1.0
    base_shift_correction: True
    stop_pub_timeout: 2.0
    # Polygons represent zone around the robot for "stop", "slowdown" and "limit" action types,
    # and robot footprint for "approach" action type.
    polygons: ["FootprintApproach"]
    FootprintApproach:
      type: "polygon"
      action_type: "approach"
      footprint_topic: "/local_costmap/published_footprint"
      time_before_collision: 1.2
      simulation_time_step: 0.1
      min_points: 6
      visualize: False
      enabled: True
    observation_sources: ["scan"]
    scan:
      type: "scan"
      topic: "scan"
      min_height: 0.15
      max_height: 2.0
      enabled: True

docking_server:
  ros__parameters:
    controller_frequency: 50.0
    initial_perception_timeout: 5.0
    wait_charge_timeout: 5.0
    dock_approach_timeout: 30.0
    undock_linear_tolerance: 0.05
    undock_angular_tolerance: 0.1
    max_retries: 3
    base_frame: "base_link"
    fixed_frame: "odom"
    dock_backwards: false
    dock_prestaging_tolerance: 0.5

    # Types of docks
    dock_plugins: ['simple_charging_dock']
    simple_charging_dock:
      plugin: 'opennav_docking::SimpleChargingDock'
      docking_threshold: 0.05
      staging_x_offset: -0.7
      use_external_detection_pose: true
      use_battery_status: false # true
      use_stall_detection: false # true

      external_detection_timeout: 1.0
```

(continues on next page)

(continued from previous page)

```
      external_detection_translation_x: -0.18
      external_detection_translation_y: 0.0
      external_detection_rotation_roll: -1.57
      external_detection_rotation_pitch: -1.57
      external_detection_rotation_yaw: 0.0
      filter_coef: 0.1

    # Dock instances
    # The following example illustrates configuring dock instances.
    # docks: ['home_dock']  # Input your docks here
    # home_dock:
    #   type: 'simple_charging_dock'
    #   frame: map
    #   pose: [0.0, 0.0, 0.0]

    controller:
      k_phi: 3.0
      k_delta: 2.0
      v_linear_min: 0.15
      v_linear_max: 0.15
      use_collision_detection: true
      costmap_topic: "local_costmap/costmap_raw"
      footprint_topic: "local_costmap/published_footprint"
      transform_tolerance: 0.1
      projection_time: 5.0
      simulation_step: 0.1
      dock_collision_threshold: 0.3

loopback_simulator:
  ros__parameters:
    base_frame_id: "base_footprint"
    odom_frame_id: "odom"
    map_frame_id: "map"
    scan_frame_id: "base_scan"  # tb4_loopback_simulator.launch.py remaps to 'rplidar_link
↪'
    update_duration: 0.02
    scan_range_min: 0.05
    scan_range_max: 30.0
    scan_angle_min: -3.1415
    scan_angle_max: 3.1415
    scan_angle_increment: 0.02617
    scan_use_inf: true
```

Once more, there is a trade-off between generality and the complexity of the configuration file. We can unpack the major elements of the configuration in the table below.

In nav2, each functionality is divided into a so-called *server*. Each server will have its own set of parameters and will communicate with other servers. In this topology, ideally one functionality can be modified without affecting other servers. These are not all servers available in nav2, but all the servers used in this example.

| Server | TL;DR | Link |
|---|---|---|
| `amcl` | Adaptive Monte-Carlo Localiser | docs |
| `bt_navigator` | Behavior tree navigator | docs |
| `controller_server` | Controller server | docs |
| `local_costmap` | A 2D costmap | docs |
| `global_costmap` | A 2D costmap | docs |
| `planner_server` | Calculates path to goal | docs |
| `smoother_server` | Keeps path smooth | docs |
| `behavior_server` | Defines basic robot behaviors | docs |
| `route_server` | Handles pre-defined routes | docs |
| `velocity_smoother` | Smooth velocities sent to robots | docs |
| `collision_monitor` | Extra layer of safety | docs |

Each server can be attached to one or more `nav2` *plugins*. The plugins will define important aspects of the server and plugins will have their own parameters.

**See also**

Official documentation: https://docs.nav2.org/plugins/index.html

In `nav2`, mainly, there will be a *global* planner (in this case `nav2_navfn_planner::NavfnPlanner`) and a *local* controller (in this case `nav2_mppi_controller::MPPIController`). In general terms, the global planner can find solutions when the goal is far and create a trajectory. The controller will be responsible for dealing with following the trajectory in the short term. This split is common in robotics. Both will use costmaps, described in a following section.

The trajectories generated by the global planner might not be smooth owing to several factors, including the parameters used to adjust execution speed. This possible jaggedness can be countered with a `smoother_server`. Analogously, the velocity calculated by the controller might not be smooth and, if applied directly, cause the robot to vibrate, destabilise, wear, or break. The `velocity_smoother` should help to address this.

Lastly, the `behavior_server` will describe basic robot behaviours such as *Wait*, or *Move Straight* for a given robot. The `bt_navigator` will utilise these, among other things, to decide higher level behaviour beyond the planner. A behaviour tree can be seen as a more general state machine. Thence, as the name implies, each behaviour can branch into different outcomes defined for each behaviour. For instance, if the robot fails reaching a goal, it can be set to do another task or attempt to replan to another location.

### 47.1.3 map

The contents of `tb3_sandbox.yaml` are shown below.

```yaml
image: tb3_sandbox.pgm
resolution: 0.050000
origin: [-10.000000, -10.000000, 0.000000]
negate: 0
occupied_thresh: 0.65
free_thresh: 0.196
```

The map will have several parameters. The global costmap is defined in `tb3_sandbox.pgm`.

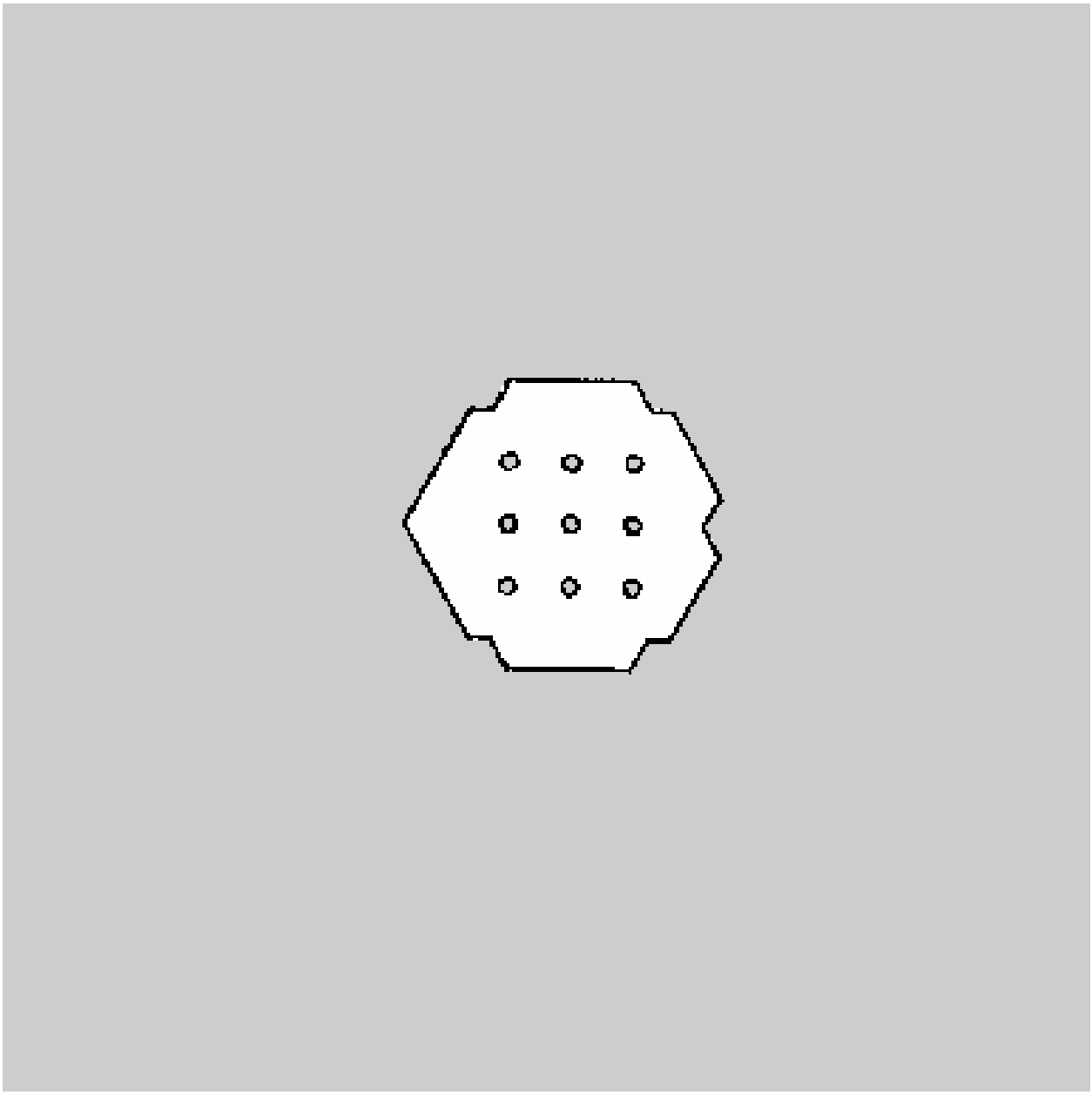
A costmap in `nav2` is a 2D grid in which each cell is assigned a value, similarly to an image. Using the image analogy, each *costmap* (image) will be made of *costs* (pixels). The value in each pixel will define how *costly* it is to move through that pixel.

Suppose that we have the small costmap below, for illustrative purposes. Suppose that we use F for free space, U for unknown, and O for occupied. We can see this as a top-view image of the path that the robot can traverse. The planner and controller will use this information to define how and where to navigate.

| | | | | |
|---|---|---|---|---|
| O | O | O | O | U |
| O | F | F | O | U |
| O | O | F | O | U |
| O | F | F | O | U |
| O | O | O | O | U |

For larger maps, and to consolidate the image analogy, this is one possible representation of `tb3_sandbox.pgm`. In this representation, the dark pixels represent occupied regions, such as walls. The white pixel represents free movement. The grey pixels represent unknown regions. This will be part of the so-called `nav2_costmap_2d::StaticLayer`, representing these three states.

Below is a representation of `tb3_sandbox.pgm` when opened in an image viewer.



Given that we are unable to fully entrust the safety of the robot to our carefully-drawn maps, `nav2` also allows the use of a `nav2_costmap_2d::ObstacleLayer`. This will be tied to one of the robot sensors, for instance, a laser scanner. This will allow the robot to avoid obstacles when the map itself was inaccurate, for instance, if an obstacle has not been accounted for, or account for inaccuracies in the localisation itself. The `nav2_costmap_2d::VoxelLayer` is used in this example for a similar purpose, with the difference that the sensor relays 3D information that is then transformed into relevant 2D information.

Lastly, there will be the `nav2_costmap_2d::InflationLayer`. Given that the maps hold only three possible states for each pixel, the changes are sudden. The inflation layer will behave similarly to a smoother. The region around each occupied cell in the grid will be modified to show that, although that is indeed free space, the robot should not be going through there. By assigning these costs when approaching an obstacle, planners and controllers can devise better trajectories.

The navigation file also had a local costmap, which works similarly, but can have different parameters and accept data from different sources. The reason for this is that it is more convenient to have individual parameters for the planner

and the controller.

### 47.1.4 World and robot definition files

As you might remember from the `.sdf` contents, they are XML files. These files, by default, do not take advantage of programming concepts, such as macros and conditionals. This becomes an issue when you want to make a robot description file that is slightly more general and can be easily modified with specific parameters. Therefore you are likely to see xacro being used in many ROS examples.

For instance, the `tb3_sandbox.sdf.xacro` file's contents can be modified by the parameter `headless`. Where the parameter is defined and where it's used are highlighted in the file below. In this example, this might be a quick patch given that **Gazebo** does not seem to support it and **xacro** will pre-process the file to output a compliant `.sdf` based on the parameter value.

#### The contents of `tb3_sandbox.sdf.xacro`.

```
<?xml version="1.0"?>
<sdf version="1.6" xmlns:xacro="http://www.ros.org/wiki/xacro">
  <xacro:arg name="headless" default="true"/>

  <world name="default">
    <plugin
      filename="gz-sim-physics-system"
      name="gz::sim::systems::Physics">
    </plugin>
    <plugin
      filename="gz-sim-user-commands-system"
      name="gz::sim::systems::UserCommands">
    </plugin>
  <xacro:unless value="$(arg headless)">
    <plugin
      filename="gz-sim-scene-broadcaster-system"
      name="gz::sim::systems::SceneBroadcaster">
    </plugin>
  </xacro:unless>
    <plugin
      filename="gz-sim-sensors-system"
      name="gz::sim::systems::Sensors">
      <render_engine>ogre2</render_engine>
    </plugin>
    <plugin
      filename="gz-sim-imu-system"
      name="gz::sim::systems::Imu">
    </plugin>

    <light name='sun' type='directional'>
        <cast_shadows>0</cast_shadows>
        <pose>0 0 10 0 0 0</pose>
        <diffuse>0.8 0.8 0.8 1</diffuse>
        <specular>0.8 0.8 0.8 1</specular>
        <attenuation>
            <range>1000</range>
            <constant>0.9</constant>
            <linear>0.01</linear>
```

(continues on next page)

(continued from previous page)

```
            <quadratic>0.001</quadratic>
        </attenuation>
        <direction>-0.5 0.1 -0.9</direction>
    </light>
    <model name='ground_plane'>
        <static>1</static>
        <link name='link'>
            <collision name='collision'>
                <geometry>
                    <plane>
                        <normal>0 0 1</normal>
                        <size>100 100</size>
                    </plane>
                </geometry>
                <max_contacts>10</max_contacts>
            </collision>
            <visual name='visual'>
                <geometry>
                    <plane>
                        <normal>0 0 1</normal>
                        <size>100 100</size>
                    </plane>
                </geometry>
                <material>
                    <ambient>0.8 0.8 0.8 1</ambient>
                    <diffuse>0.8 0.8 0.8 1</diffuse>
                    <specular>0.8 0.8 0.8 1</specular>
                </material>
            </visual>
        </link>
    </model>

    <scene>
      <shadows>0</shadows>
    </scene>

    <physics name='3ms' type='ode'>
        <max_step_size>0.003</max_step_size>
        <real_time_factor>1</real_time_factor>
    </physics>

    <model name="turtlebot3_world">
      <static>1</static>
      <include>
        <uri>model://turtlebot3_world</uri>
      </include>
    </model>

  </world>
</sdf>
```

Something similar is done to allow **Gazebo** topics to have a different namespace. This is helpful when multiple robots are added to the same **Gazebo** world.

**The contents of `gz_waffle.sdf.xacro`.**

```
<?xml version="1.0"?>
<sdf version="1.6" xmlns:xacro="http://www.ros.org/wiki/xacro">
    <xacro:arg name="namespace" default=""/>

    <model name="turtlebot3_waffle">
      <pose>0.0 0.0 0.0 0.0 0.0 0.0</pose>

      <link name="base_footprint"/>

      <link name="base_link">

        <inertial>
          <pose>-0.064 0 0.048 0 0 0</pose>
          <inertia>
            <ixx>0.001</ixx>
            <ixy>0.000</ixy>
            <ixz>0.000</ixz>
            <iyy>0.001</iyy>
            <iyz>0.000</iyz>
            <izz>0.001</izz>
          </inertia>
          <mass>1.0</mass>
        </inertial>

        <collision name="base_collision">
          <pose>-0.064 0 0.048 0 0 0</pose>
          <geometry>
            <box>
              <size>0.265 0.265 0.089</size>
            </box>
          </geometry>
        </collision>

        <visual name="base_visual">
          <pose>-0.064 0 0 0 0 0</pose>
          <geometry>
            <mesh>
              <uri>package://nav2_minimal_tb3_sim/models/turtlebot3_model/meshes/waffle_
↪base.dae</uri>
              <scale>0.001 0.001 0.001</scale>
            </mesh>
          </geometry>
          <material>
            <diffuse>1 1 1</diffuse>
          </material>
        </visual>
      </link>

      <link name="imu_link">
        <sensor name="tb3_imu" type="imu">
          <always_on>true</always_on>
```

(continues on next page)

(continued from previous page)

```
        <update_rate>200</update_rate>
        <topic>$(arg namespace)/imu</topic>
        <gz_frame_id>imu_link</gz_frame_id>
        <imu>
          <angular_velocity>
            <x>
              <noise type="gaussian">
                <mean>0.0</mean>
                <stddev>2e-4</stddev>
              </noise>
            </x>
            <y>
              <noise type="gaussian">
                <mean>0.0</mean>
                <stddev>2e-4</stddev>
              </noise>
            </y>
            <z>
              <noise type="gaussian">
                <mean>0.0</mean>
                <stddev>2e-4</stddev>
              </noise>
            </z>
          </angular_velocity>
          <linear_acceleration>
            <x>
              <noise type="gaussian">
                <mean>0.0</mean>
                <stddev>1.7e-2</stddev>
              </noise>
            </x>
            <y>
              <noise type="gaussian">
                <mean>0.0</mean>
                <stddev>1.7e-2</stddev>
              </noise>
            </y>
            <z>
              <noise type="gaussian">
                <mean>0.0</mean>
                <stddev>1.7e-2</stddev>
              </noise>
            </z>
          </linear_acceleration>
        </imu>
      </sensor>
    </link>

    <link name="base_scan">
      <inertial>
        <pose>-0.064 0 0.121 0 0 0</pose>
        <inertia>
```

(continues on next page)

(continued from previous page)

```
            <ixx>0.001</ixx>
            <ixy>0.000</ixy>
            <ixz>0.000</ixz>
            <iyy>0.001</iyy>
            <iyz>0.000</iyz>
            <izz>0.001</izz>
          </inertia>
          <mass>0.125</mass>
        </inertial>

        <collision name="lidar_sensor_collision">
          <pose>-0.052 0 0.111 0 0 0</pose>
          <geometry>
            <cylinder>
              <radius>0.0508</radius>
              <length>0.055</length>
            </cylinder>
          </geometry>
        </collision>

        <visual name="lidar_sensor_visual">
          <pose>-0.064 0 0.121 0 0 0</pose>
          <geometry>
            <mesh>
              <uri>package://nav2_minimal_tb3_sim/models/turtlebot3_model/meshes/lds.dae
↪</uri>
              <scale>0.001 0.001 0.001</scale>
            </mesh>
          </geometry>
          <material>
            <diffuse>1 1 1</diffuse>
          </material>
        </visual>

        <sensor name="hls_lfcd_lds" type="gpu_lidar">
          <always_on>true</always_on>
          <visualize>true</visualize>
          <pose>-0.064 0 0.15 0 0 0</pose>
          <update_rate>5</update_rate>
          <topic>$(arg namespace)/scan</topic>
          <gz_frame_id>base_scan</gz_frame_id>
          <ray>
            <scan>
              <horizontal>
                <samples>360</samples>
                <resolution>1.000000</resolution>
                <min_angle>0.000000</min_angle>
                <max_angle>6.280000</max_angle>
              </horizontal>
            </scan>
            <range>
              <min>0.00001</min>
```

(continues on next page)

(continued from previous page)

```
            <max>20.0</max>
            <resolution>0.015000</resolution>
          </range>
          <noise>
            <type>gaussian</type>
            <mean>0.0</mean>
            <stddev>0.01</stddev>
          </noise>
        </ray>
      </sensor>
    </link>

    <link name="wheel_left_link">

      <inertial>
        <pose>0.0 0.144 0.023 -1.57 0 0</pose>
        <inertia>
          <ixx>0.001</ixx>
          <ixy>0.000</ixy>
          <ixz>0.000</ixz>
          <iyy>0.001</iyy>
          <iyz>0.000</iyz>
          <izz>0.001</izz>
        </inertia>
        <mass>0.1</mass>
      </inertial>

      <collision name="wheel_left_collision">
        <pose>0.0 0.144 0.023 -1.57 0 0</pose>
        <geometry>
          <cylinder>
            <radius>0.033</radius>
            <length>0.018</length>
          </cylinder>
        </geometry>
        <surface>
          <friction>
            <ode>
              <mu>1</mu>
              <mu2>1</mu2>
              <slip1>0.035</slip1>
              <slip2>0</slip2>
              <fdir1>0 0 1</fdir1>
            </ode>
          </friction>
          <contact>
            <ode>
              <soft_cfm>0</soft_cfm>
              <soft_erp>0.2</soft_erp>
              <kp>1e+5</kp>
              <kd>1</kd>
              <max_vel>0.01</max_vel>
```

(continues on next page)

(continued from previous page)

```
              <min_depth>0.001</min_depth>
            </ode>
          </contact>
        </surface>
      </collision>

      <visual name="wheel_left_visual">
        <pose>0.0 0.144 0.023 0 0 0</pose>
        <geometry>
          <mesh>
            <uri>package://nav2_minimal_tb3_sim/models/turtlebot3_model/meshes/tire.dae
↪</uri>
            <scale>0.001 0.001 0.001</scale>
          </mesh>
        </geometry>
        <material>
          <diffuse>1 1 1</diffuse>
        </material>
      </visual>
    </link>

    <link name="wheel_right_link">

      <inertial>
        <pose>0.0 -0.144 0.023 -1.57 0 0</pose>
        <inertia>
          <ixx>0.001</ixx>
          <ixy>0.000</ixy>
          <ixz>0.000</ixz>
          <iyy>0.001</iyy>
          <iyz>0.000</iyz>
          <izz>0.001</izz>
        </inertia>
        <mass>0.1</mass>
      </inertial>

      <collision name="wheel_right_collision">
        <pose>0.0 -0.144 0.023 -1.57 0 0</pose>
        <geometry>
          <cylinder>
            <radius>0.033</radius>
            <length>0.018</length>
          </cylinder>
        </geometry>
        <surface>
          <friction>
            <ode>
              <mu>1</mu>
              <mu2>1</mu2>
              <slip1>0.035</slip1>
              <slip2>0</slip2>
              <fdir1>0 0 1</fdir1>
```

(continues on next page)

(continued from previous page)

```
            </ode>
          </friction>
          <contact>
            <ode>
              <soft_cfm>0</soft_cfm>
              <soft_erp>0.2</soft_erp>
              <kp>1e+5</kp>
              <kd>1</kd>
              <max_vel>0.01</max_vel>
              <min_depth>0.001</min_depth>
            </ode>
          </contact>
        </surface>
      </collision>

      <visual name="wheel_right_visual">
        <pose>0.0 -0.144 0.023 0 0 0</pose>
        <geometry>
          <mesh>
            <uri>package://nav2_minimal_tb3_sim/models/turtlebot3_model/meshes/tire.dae
↪</uri>
            <scale>0.001 0.001 0.001</scale>
          </mesh>
        </geometry>
        <material>
          <diffuse>1 1 1</diffuse>
        </material>
      </visual>
    </link>

    <link name='caster_back_right_link'>
      <pose>-0.177 -0.064 -0.004 0 0 0</pose>
      <inertial>
        <mass>0.001</mass>
        <inertia>
          <ixx>0.00001</ixx>
          <ixy>0.000</ixy>
          <ixz>0.000</ixz>
          <iyy>0.00001</iyy>
          <iyz>0.000</iyz>
          <izz>0.00001</izz>
        </inertia>
      </inertial>
      <collision name='collision'>
        <geometry>
          <sphere>
            <radius>0.006000</radius>
          </sphere>
        </geometry>
        <surface>
          <contact>
            <ode>
```

(continues on next page)

(continued from previous page)

```
              <soft_cfm>0</soft_cfm>
              <soft_erp>0.2</soft_erp>
              <kp>1e+5</kp>
              <kd>1</kd>
              <max_vel>0.01</max_vel>
              <min_depth>0.001</min_depth>
            </ode>
          </contact>
        </surface>
      </collision>
    </link>

    <link name='caster_back_left_link'>
      <pose>-0.177 0.064 -0.004 0 0 0</pose>
      <inertial>
        <mass>0.001</mass>
        <inertia>
          <ixx>0.00001</ixx>
          <ixy>0.000</ixy>
          <ixz>0.000</ixz>
          <iyy>0.00001</iyy>
          <iyz>0.000</iyz>
          <izz>0.00001</izz>
        </inertia>
      </inertial>
      <collision name='collision'>
        <geometry>
          <sphere>
            <radius>0.006000</radius>
          </sphere>
        </geometry>
        <surface>
          <contact>
            <ode>
              <soft_cfm>0</soft_cfm>
              <soft_erp>0.2</soft_erp>
              <kp>1e+5</kp>
              <kd>1</kd>
              <max_vel>0.01</max_vel>
              <min_depth>0.001</min_depth>
            </ode>
          </contact>
        </surface>
      </collision>
    </link>

    <link name="camera_link">
      <inertial>
        <pose>0.069 -0.047 0.107 0 0 0</pose>
        <inertia>
          <ixx>0.001</ixx>
          <ixy>0.000</ixy>
```

(continues on next page)

(continued from previous page)

```
            <ixz>0.000</ixz>
            <iyy>0.001</iyy>
            <iyz>0.000</iyz>
            <izz>0.001</izz>
          </inertia>
          <mass>0.035</mass>
        </inertial>
        <collision name="collision">
          <pose>0 0.047 -0.005 0 0 0</pose>
          <geometry>
            <box>
              <size>0.008 0.130 0.022</size>
            </box>
          </geometry>
        </collision>

        <pose>0.069 -0.047 0.107 0 0 0</pose>

        <sensor name="intel_realsense_r200_depth" type="depth">
          <always_on>1</always_on>
          <update_rate>5</update_rate>
          <pose>0.064 -0.047 0.107 0 0 0</pose>
          <gz_frame_id>camera_depth_frame</gz_frame_id>
          <camera name="realsense_depth_camera">
            <horizontal_fov>1.047</horizontal_fov>
            <image>
              <width>320</width>
              <height>240</height>
            </image>
            <lens>
              <projection>
                <!-- focal_length = fx = fy = width / ( 2 * tan (hfov / 2 ) ) -->
                <!-- tx = hackBaseline * focal_length -->
                <tx>19.4</tx>
              </projection>
            </lens>
            <depth_camera>
              <clip>
                <near>0.001</near>
                <far>5.0</far>
              </clip>
            </depth_camera>
          </camera>
        </sensor>
      </link>

      <joint name="base_joint" type="fixed">
        <parent>base_footprint</parent>
        <child>base_link</child>
        <pose>0.0 0.0 0.010 0 0 0</pose>
      </joint>

```

(continues on next page)

(continued from previous page)

```
    <joint name="wheel_left_joint" type="revolute">
      <parent>base_link</parent>
      <child>wheel_left_link</child>
      <pose>0.0 0.144 0.023 -1.57 0 0</pose>
      <axis>
        <xyz>0 0 1</xyz>
      </axis>
    </joint>

    <joint name="wheel_right_joint" type="revolute">
      <parent>base_link</parent>
      <child>wheel_right_link</child>
      <pose>0.0 -0.144 0.023 -1.57 0 0</pose>
      <axis>
        <xyz>0 0 1</xyz>
      </axis>
    </joint>

    <joint name='caster_back_right_joint' type='ball'>
      <parent>base_link</parent>
      <child>caster_back_right_link</child>
    </joint>

    <joint name='caster_back_left_joint' type='ball'>
      <parent>base_link</parent>
      <child>caster_back_left_link</child>
    </joint>

    <joint name="lidar_joint" type="fixed">
      <parent>base_link</parent>
      <child>base_scan</child>
      <pose>-0.064 0 0.121 0 0 0</pose>
      <axis>
        <xyz>0 0 1</xyz>
      </axis>
    </joint>

    <joint name="camera_joint" type="fixed">
      <parent>base_link</parent>
      <child>camera_link</child>
      <pose>0.064 -0.065 0.094 0 0 0</pose>
      <axis>
        <xyz>0 0 1</xyz>
      </axis>
    </joint>

    <joint name="imu_joint" type="fixed">
      <parent>base_link</parent>
      <child>imu_link</child>
      <pose>0.0 0 0.068 0 0 0</pose>
    </joint>

```

(continues on next page)

(continued from previous page)

```
      <plugin
        filename="gz-sim-diff-drive-system"
        name="gz::sim::systems::DiffDrive">
        <left_joint>wheel_left_joint</left_joint>
        <right_joint>wheel_right_joint</right_joint>
        <wheel_separation>0.287</wheel_separation>
        <wheel_radius>0.033</wheel_radius>
        <max_linear_acceleration>1</max_linear_acceleration>
        <min_linear_acceleration>-1</min_linear_acceleration>
        <max_angular_acceleration>2</max_angular_acceleration>
        <min_angular_acceleration>-2</min_angular_acceleration>
        <max_linear_velocity>0.46</max_linear_velocity>
        <min_linear_velocity>-0.46</min_linear_velocity>
        <max_angular_velocity>1.9</max_angular_velocity>
        <min_angular_velocity>-1.9</min_angular_velocity>
        <topic>$(arg namespace)/cmd_vel</topic>
        <odom_topic>$(arg namespace)/odom</odom_topic>
        <tf_topic>$(arg namespace)/tf</tf_topic>
        <frame_id>odom</frame_id>
        <child_frame_id>base_footprint</child_frame_id>
        <odom_publish_frequency>30</odom_publish_frequency>
      </plugin>

      <plugin
        filename="gz-sim-joint-state-publisher-system"
        name="gz::sim::systems::JointStatePublisher">
        <joint_name>wheel_left_joint</joint_name>
        <joint_name>wheel_right_joint</joint_name>
        <topic>$(arg namespace)/joint_states</topic>
        <update_rate>30</update_rate>
      </plugin>

    </model>
</sdf>
```

Although `.sdf` files are meant to replace, or supersede, `.urdf` files in the long term, the process is still ongoing or under discussion. It is likely that you will see `.urdf` files heavily used in many **ROS2** packages. Nonetheless, `.urdf` are close in syntax to `.sdf` files and should be no surprise.

### The contents of `turtlebot3_waffle.urdf`.

```
<?xml version="1.0" ?>
<robot name="turtlebot3_waffle" xmlns:xacro="http://ros.org/wiki/xacro">
  <!-- <xacro:include filename="$(find turtlebot3_description)/urdf/common_properties.
↪urdf"/>

  <xacro:property name="r200_cam_rgb_px" value="0.005"/>
  <xacro:property name="r200_cam_rgb_py" value="0.018"/>
  <xacro:property name="r200_cam_rgb_pz" value="0.013"/>
  <xacro:property name="r200_cam_depth_offset" value="0.01"/> -->

```

(continues on next page)

(continued from previous page)

```
  <!-- Init colour -->
  <material name="black">
      <color rgba="0.0 0.0 0.0 1.0"/>
  </material>

  <material name="dark">
    <color rgba="0.3 0.3 0.3 1.0"/>
  </material>

  <material name="light_black">
    <color rgba="0.4 0.4 0.4 1.0"/>
  </material>

  <material name="blue">
    <color rgba="0.0 0.0 0.8 1.0"/>
  </material>

  <material name="green">
    <color rgba="0.0 0.8 0.0 1.0"/>
  </material>

  <material name="grey">
    <color rgba="0.5 0.5 0.5 1.0"/>
  </material>

  <material name="orange">
    <color rgba="1.0 0.4235 0.0392 1.0"/>
  </material>

  <material name="brown">
    <color rgba="0.8706 0.8118 0.7647 1.0"/>
  </material>

  <material name="red">
    <color rgba="0.8 0.0 0.0 1.0"/>
  </material>

  <material name="white">
    <color rgba="1.0 1.0 1.0 1.0"/>
  </material>

  <link name="base_footprint"/>

  <joint name="base_joint" type="fixed">
    <parent link="base_footprint"/>
    <child link="base_link" />
    <origin xyz="0 0 0.010" rpy="0 0 0"/>
  </joint>

  <link name="base_link">
    <visual>
      <origin xyz="-0.064 0 0.0" rpy="0 0 0"/>
```

(continues on next page)

(continued from previous page)

```
      <geometry>
        <mesh filename="package://nav2_minimal_tb3_sim/models/turtlebot3_model/meshes/
↪waffle_base.dae" scale="0.001 0.001 0.001"/>
      </geometry>
      <material name="light_black"/>
    </visual>

    <collision>
      <origin xyz="-0.064 0 0.047" rpy="0 0 0"/>
      <geometry>
        <box size="0.266 0.266 0.094"/>
      </geometry>
    </collision>

    <inertial>
      <origin xyz="0 0 0" rpy="0 0 0"/>
      <mass value="1.3729096e+00"/>
      <inertia ixx="8.7002718e-03" ixy="-4.7576583e-05" ixz="1.1160499e-04"
               iyy="8.6195418e-03" iyz="-3.5422299e-06"
               izz="1.4612727e-02" />
    </inertial>
  </link>

  <joint name="wheel_left_joint" type="continuous">
    <parent link="base_link"/>
    <child link="wheel_left_link"/>
    <origin xyz="0.0 0.144 0.023" rpy="-1.57 0 0"/>
    <axis xyz="0 0 1"/>
  </joint>

  <link name="wheel_left_link">
    <visual>
      <origin xyz="0 0 0" rpy="1.57 0 0"/>
      <geometry>
        <mesh filename="package://nav2_minimal_tb3_sim/models/turtlebot3_model/meshes/
↪tire.dae" scale="0.001 0.001 0.001"/>
      </geometry>
      <material name="dark"/>
    </visual>

    <collision>
      <origin xyz="0 0 0" rpy="0 0 0"/>
      <geometry>
        <cylinder length="0.018" radius="0.033"/>
      </geometry>
    </collision>

    <inertial>
      <origin xyz="0 0 0" />
      <mass value="2.8498940e-02" />
      <inertia ixx="1.1175580e-05" ixy="-4.2369783e-11" ixz="-5.9381719e-09"
               iyy="1.1192413e-05" iyz="-1.4400107e-11"
```

(continues on next page)

(continued from previous page)

```
              izz="2.0712558e-05" />
    </inertial>
  </link>

  <joint name="wheel_right_joint" type="continuous">
    <parent link="base_link"/>
    <child link="wheel_right_link"/>
    <origin xyz="0.0 -0.144 0.023" rpy="-1.57 0 0"/>
    <axis xyz="0 0 1"/>
  </joint>

  <link name="wheel_right_link">
    <visual>
      <origin xyz="0 0 0" rpy="1.57 0 0"/>
      <geometry>
        <mesh filename="package://nav2_minimal_tb3_sim/models/turtlebot3_model/meshes/
↪tire.dae" scale="0.001 0.001 0.001"/>
      </geometry>
      <material name="dark"/>
    </visual>

    <collision>
      <origin xyz="0 0 0" rpy="0 0 0"/>
      <geometry>
        <cylinder length="0.018" radius="0.033"/>
      </geometry>
    </collision>

    <inertial>
      <origin xyz="0 0 0" />
      <mass value="2.8498940e-02" />
      <inertia ixx="1.1175580e-05" ixy="-4.2369783e-11" ixz="-5.9381719e-09"
               iyy="1.1192413e-05" iyz="-1.4400107e-11"
               izz="2.0712558e-05" />
      </inertial>
  </link>

  <joint name="caster_back_right_joint" type="fixed">
    <parent link="base_link"/>
    <child link="caster_back_right_link"/>
    <origin xyz="-0.177 -0.064 -0.004" rpy="-1.57 0 0"/>
  </joint>

  <link name="caster_back_right_link">
    <collision>
      <origin xyz="0 0.001 0" rpy="0 0 0"/>
      <geometry>
        <box size="0.030 0.009 0.020"/>
      </geometry>
    </collision>

    <inertial>
```

(continues on next page)

(continued from previous page)

```
      <origin xyz="0 0 0" />
      <mass value="0.005" />
      <inertia ixx="0.001" ixy="0.0" ixz="0.0"
               iyy="0.001" iyz="0.0"
               izz="0.001" />
    </inertial>
  </link>

  <joint name="caster_back_left_joint" type="fixed">
    <parent link="base_link"/>
    <child link="caster_back_left_link"/>
    <origin xyz="-0.177 0.064 -0.004" rpy="-1.57 0 0"/>
  </joint>

  <link name="caster_back_left_link">
    <collision>
      <origin xyz="0 0.001 0" rpy="0 0 0"/>
      <geometry>
        <box size="0.030 0.009 0.020"/>
      </geometry>
    </collision>

    <inertial>
      <origin xyz="0 0 0" />
      <mass value="0.005" />
      <inertia ixx="0.001" ixy="0.0" ixz="0.0"
               iyy="0.001" iyz="0.0"
               izz="0.001" />
    </inertial>
  </link>

  <joint name="imu_joint" type="fixed">
    <parent link="base_link"/>
    <child link="imu_link"/>
    <origin xyz="0.0 0 0.068" rpy="0 0 0"/>
  </joint>

  <link name="imu_link"/>

  <joint name="scan_joint" type="fixed">
    <parent link="base_link"/>
    <child link="base_scan"/>
    <origin xyz="-0.064 0 0.122" rpy="0 0 0"/>
  </joint>

  <link name="base_scan">
    <visual>
      <origin xyz="0 0 0" rpy="0 0 0"/>
      <geometry>
        <mesh filename="package://nav2_minimal_tb3_sim/models/turtlebot3_model/meshes/
↪lds.dae" scale="0.001 0.001 0.001"/>
      </geometry>
```

(continues on next page)

(continued from previous page)

```
      <material name="dark"/>
    </visual>

    <collision>
      <origin xyz="0.015 0 -0.0065" rpy="0 0 0"/>
      <geometry>
        <cylinder length="0.0315" radius="0.055"/>
      </geometry>
    </collision>

    <inertial>
      <mass value="0.114" />
      <origin xyz="0 0 0" />
      <inertia ixx="0.001" ixy="0.0" ixz="0.0"
               iyy="0.001" iyz="0.0"
               izz="0.001" />
    </inertial>
  </link>

  <joint name="camera_joint" type="fixed">
    <origin xyz="0.064 -0.065 0.094" rpy="0 0 0"/>
    <parent link="base_link"/>
    <child link="camera_link"/>
  </joint>

  <link name="camera_link">
    <visual>
     <origin xyz="0 0 0" rpy="1.57 0 1.57"/>
      <geometry>
        <mesh filename="package://nav2_minimal_tb3_sim/models/turtlebot3_model/meshes/
↪r200.dae" />
      </geometry>
    </visual>
    <collision>
      <origin xyz="0.003 0.065 0.007" rpy="0 0 0"/>
      <geometry>
        <box size="0.012 0.132 0.020"/>
      </geometry>
    </collision>

    <!-- This inertial field needs doesn't contain reliable data!! -->
<!--   <inertial>
      <mass value="0.564" />
      <origin xyz="0 0 0" />
      <inertia ixx="0.003881243" ixy="0.0" ixz="0.0"
               iyy="0.000498940" iyz="0.0"
               izz="0.003879257" />
    </inertial>-->
  </link>

  <joint name="camera_rgb_joint" type="fixed">
    <origin xyz="0.005 0.018 0.013" rpy="0 0 0"/>
```

(continues on next page)

(continued from previous page)

```
    <!-- <origin xyz="${r200_cam_rgb_px} ${r200_cam_rgb_py} ${r200_cam_rgb_pz}" rpy="0 0 0"/> -->
    <parent link="camera_link"/>
    <child link="camera_rgb_frame"/>
  </joint>
  <link name="camera_rgb_frame"/>

  <joint name="camera_rgb_optical_joint" type="fixed">
    <origin xyz="0 0 0" rpy="-1.57 0 -1.57"/>
    <parent link="camera_rgb_frame"/>
    <child link="camera_rgb_optical_frame"/>
  </joint>
  <link name="camera_rgb_optical_frame"/>

  <joint name="camera_depth_joint" type="fixed">
    <origin xyz="0.005 0.028 0.013" rpy="-1.57 0 -1.57"/>
    <!-- <origin xyz="${r200_cam_rgb_px} ${r200_cam_rgb_py + r200_cam_depth_offset} ${r200_cam_rgb_pz}" rpy="0 0 0"/> -->
    <parent link="camera_link"/>
    <child link="camera_depth_frame"/>
  </joint>
  <link name="camera_depth_frame"/>

  <joint name="camera_depth_optical_joint" type="fixed">
    <origin xyz="0 0 0" rpy="0 0 0"/>
    <parent link="camera_depth_frame"/>
    <child link="camera_depth_optical_frame"/>
  </joint>
  <link name="camera_depth_optical_frame"/>

</robot>
```

### 47.1.5 So what?

After looking through this example you have probably noticed the complexity involved into setting up `nav2` for even relatively simple projects. Having a highly configurable system increases the number of possibilities, which is great when applying it to other settings, but might be daunting at first.

Here are some examples of what you could modify in this example and what would be the estimated difficulty of doing so.

- Changing the map. Not as trivial as it might sound, because the map needs to match the **Gazebo** simulation scene. Some community tools might help with that.
- Changing the **Gazebo** scene. Same problem as above, although you might benefit from SLAM, it is not likely to create a perfect map.
- Changing the robot. Online tutorials and official repositories might say that it's a matter of "changing the `urdf` file". Unfortunately, this might mean lots of other things will change, including the navigation parameters depending, for instance, on sensors and footprint of your robot.
- Changing planners or controllers in `nav2`. That is going to be easier generally, but the parameters of one planner or controller might not map exactly to another one. It might take some tinkering to get it to work.

In conclusion, using this section, you have learned the basics of `nav2` from an official example. We went through the

importance of each file and now you should have a conceptual knowledge beyond the basics for this navigation stack.

## 47.2 Navigation with SLAM

**See also**

Official documentation: https://docs.nav2.org/tutorials/docs/navigation2_with_slam.html#navigation2-with-slam

In the previous example, we looked into navigating through a completely known map. This is a common problem to solve in robotics, but definitely not the only issue. Commonly, we have partial or no knowledge of the map through which a robot has to traverse. This is where SLAM kicks in. It becomes a problem of navigating through an incomplete and dynamic map.

In this example, they utilise the slam toolbox. Going into the details of SLAM is beyond the scope of this tutorial, but there are some interesting aspects to learn from this example even with the basic concepts of navigation.

We can run the demo with SLAM using the code below. Please note that the only difference is the flag `slam:=True`.

```
ros2 launch nav2_bringup \
tb3_simulation_launch.py \
use_sim_time:=True \
headless:=False \
sigterm_timeout:=120 \
slam:=True
```

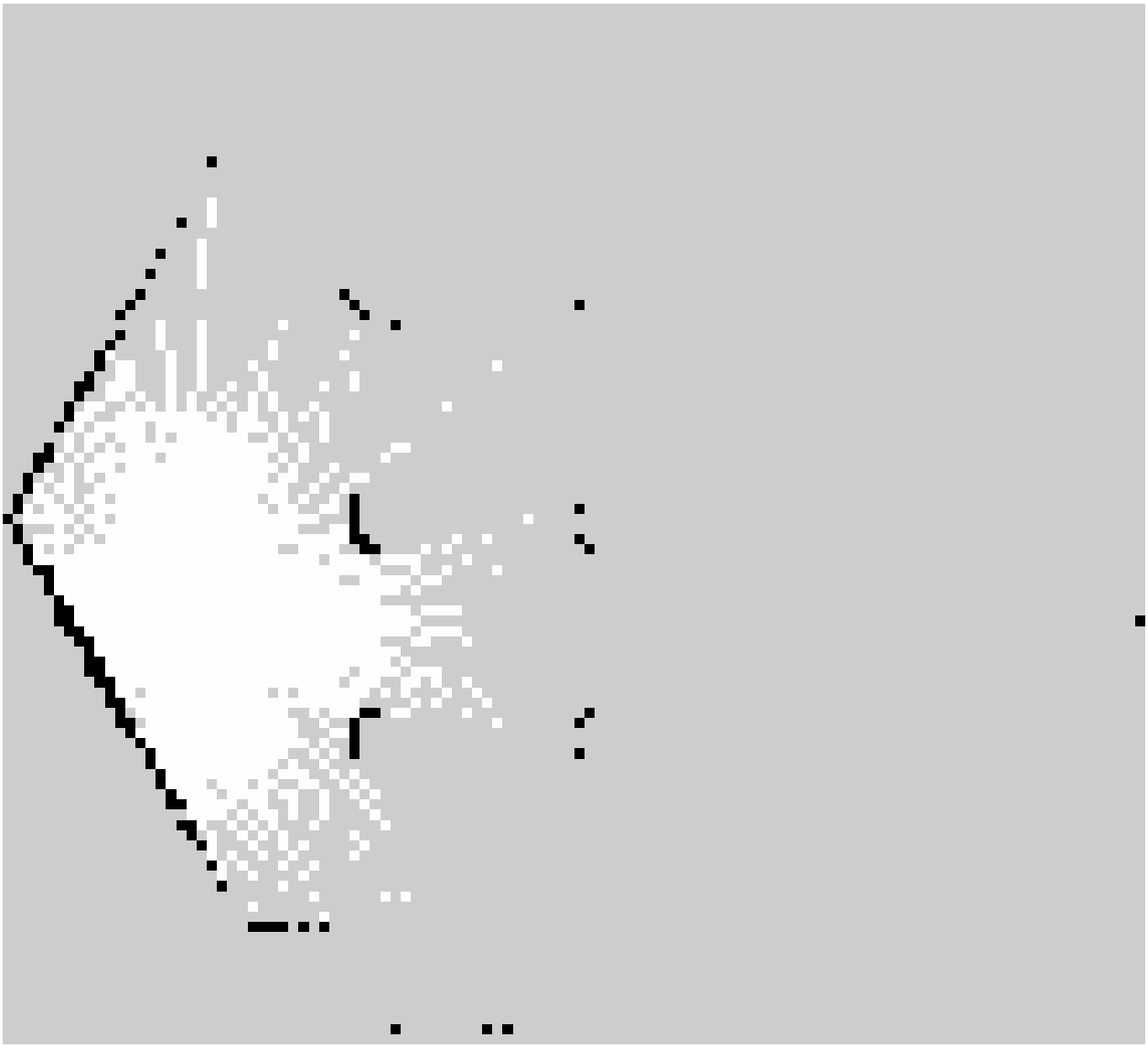
You can behold my incredible map completion skills in the video below. As you will notice it will start empty and be completed as we move around it. Using our knowledge of the environment we can know where to go to complete the map and have good coverage. In practice, this can be challenging when the environment is completely unknown and has more complex topology.

Running the same demonstration a second time, we can see how to save an incomplete map. The map will be saved in the folder in which you executed `ros2 launch`.

The maps will be saved into two files, compliant with `nav2` use. The first one we saved will be the `.yaml` file.

```
image: incomplete.pgm
mode: trinary
resolution: 0.050
origin: [-0.934, -2.059, 0]
negate: 0
occupied_thresh: 0.65
free_thresh: 0.196
```

The second one will be the map. When opened in a image viewer, this is how it looks like, showing its incomplete state.



This example is illustrative of the capacity to create maps using SLAM. This might not look like much, at first glance. However, please take this time to reflect that this mapping is happening with simulated data coming from **Gazebo**. Therefore, it is as representative of reality as the programs we have used so far allow us to be at this point.

CHAPTER

FORTYEIGHT

# INTERFACE NAV2 WITH CUSTOM ROS2 NODES

Added in version Jazzy: This section.

**Warning**

This topic is under construction and this might not even be its final form. Please feel free to open an issue if you spot any typos or other problems.

## 48.1 Objective

We are going to create custom code to interact with the navigation stack, `nav2`. This will be based on the demo `tb3_simulation_launch.py`, but it is generally applicable. In the example, `rviz2` is used for interactivity. In this example, we will be able to operate `nav2` automatically, without relying on the visualizer.

1. Create a publisher to send the initial pose.
2. Create an action client to send navigation goals.

**Important**

Differently from everything else used in this tutorial, this `tb3_simulation_launch.py` demo does not always finishes cleanly on my machines. It has hanged sometimes, sometimes **Gazebo** does not show, and sometimes it does not shutdown properly.

It is highly recommended to use a docker container, such as https://github.com/UoMMScRobotics/sfr_gazebo_nav2, to minimise disappointment.

When dealing with a new stack you might not be familiar with, the first step is to try to make sense of the interfaces available.

### 48.1.1 Initial pose topic

When the `tb3_simulation_launch.py` example is running, if we run `ros2 topic list` you will see a long list of topics. Let us filter them out with the keyword *pose*, giving that we're trying to find things related to pose. We can do so with the command below.

```
ros2 topic list | grep pose
```

The output of that will be the following topics.

```
/amcl_pose
/detected_dock_pose
/dock_pose
/filtered_dock_pose
/goal_pose
/initialpose
/staging_pose
```

This is convenient for us because `/initialpose` describes exactly what we want to do in the first step. If you are annoyed by the fact that the words *initial* and *pose* are not separated by a _, so am I.

We can check if this is in fact the correct topic used by **rviz2**. With the demo running, we run, in another terminal, the following command.

```
ros2 topic echo /initialpose
```

Then, we send the initial position through **rviz2**. We can see that in fact **rviz2** used this topic to set the initial pose, therefore our guess was correct. Below is not representative output.

### Output of ros2 topic echo

```
header:
  stamp:
    sec: 247
    nanosec: 791000000
  frame_id: map
pose:
  pose:
    position:
      x: -2.1046903133392334
      y: -0.3082020580768585
      z: 0.0
    orientation:
      x: 0.0
      y: 0.0
      z: 0.0
      w: 1.0
  covariance:
  - 0.25
  - 0.0
  - 0.0
  - 0.0
  - 0.0
  - 0.0
  - 0.0
  - 0.25
  - 0.0
  - 0.0
  - 0.0
  - 0.0
  - 0.0
  - 0.0
  - 0.0
```

(continues on next page)

(continued from previous page)

```
  - 0.0
  - 0.0
  - 0.0
  - 0.0
  - 0.0
  - 0.0
  - 0.0
  - 0.0
  - 0.0
  - 0.0
  - 0.0
  - 0.0
  - 0.0
  - 0.0
  - 0.0
  - 0.0
  - 0.0
  - 0.0
  - 0.06853891909122467
---
```

We can understand more about the topic with the following command.

```
ros2 topic info /initialpose
```

This will show us that the topic uses `geometry_msgs/msg/PoseWithCovarianceStamped`. Thence, all we have to do is create a subscriber with that message type to the topic `/initialpose`.

```
Type: geometry_msgs/msg/PoseWithCovarianceStamped
Publisher count: 1
Subscription count: 1
```

Although it might look somewhat trivial at this stage, there are many **ROS2** concepts at play so that these connections can be understood.

### 48.1.2 Navigate to pose action

When operating the demo through **rviz2**, you might have noticed that sending navigation goals behaves as an action. It has a goal, feedback, and a result. With that in mind, we look through the active action, once more filtering for *pose*. We can do so with the following command.

```
ros2 action list | grep pose
```

This results in the following actions.

```
/compute_path_through_poses
/compute_path_to_pose
/navigate_through_poses
/navigate_to_pose
```

Good sense should indicate that, given that we want to navigate to a pose, the correct topic in question is `/navigate_to_pose`. We can check more information about it with the following command.

```
ros2 action info /navigate_to_pose
```

The information clearly states that **rviz2** is connected to it, which is a good indicator that we are on the right path.

```
Action: /navigate_to_pose
Action clients: 5
    /rviz
    /rviz_navigation_dialog_action_client
    /bt_navigator
    /waypoint_follower
    /docking_server
Action servers: 1
    /bt_navigator
```

With the following command we see what type of action we have to support.

```
ros2 action type /navigate_to_pose
```

The output of the command will be the following action type.

```
nav2_msgs/action/NavigateToPose
```

Thence, our next step is to see what are the fields that we need to support in our action client. The long way to do so is with the command we are already aware of.

```
ros2 interface show nav2_msgs/action/NavigateToPose
```

### Output of ros2 interface show

```
#goal definition
geometry_msgs/PoseStamped pose
    std_msgs/Header header
        builtin_interfaces/Time stamp
            int32 sec
            uint32 nanosec
        string frame_id
    Pose pose
        Point position
            float64 x
            float64 y
            float64 z
        Quaternion orientation
            float64 x 0
            float64 y 0
            float64 z 0
            float64 w 1
string behavior_tree
---
#result definition

# Error codes
# Note: The expected priority order of the errors should match the message order
uint16 NONE=0
```

(continues on next page)

(continued from previous page)

```
uint16 error_code
string error_msg
---
#feedback definition
geometry_msgs/PoseStamped current_pose
    std_msgs/Header header
        builtin_interfaces/Time stamp
            int32 sec
            uint32 nanosec
        string frame_id
    Pose pose
        Point position
            float64 x
            float64 y
            float64 z
        Quaternion orientation
            float64 x 0
            float64 y 0
            float64 z 0
            float64 w 1
builtin_interfaces/Duration navigation_time
    int32 sec
    uint32 nanosec
builtin_interfaces/Duration estimated_time_remaining
    int32 sec
    uint32 nanosec
int16 number_of_recoveries
float32 distance_remaining
```

A more condensed version of the action can be obtained with the following commands.

```
cd $(ros2 pkg prefix nav2_msgs --share)
cat action/NavigateToPose.action
```

The output of this command will be the following.

```
#goal definition
geometry_msgs/PoseStamped pose
string behavior_tree
---
#result definition

# Error codes
# Note: The expected priority order of the errors should match the message order
uint16 NONE=0

uint16 error_code
string error_msg
---
#feedback definition
geometry_msgs/PoseStamped current_pose
```

(continues on next page)

(continued from previous page)

```
builtin_interfaces/Duration navigation_time
builtin_interfaces/Duration estimated_time_remaining
int16 number_of_recoveries
float32 distance_remaining
```

### 48.1.3 So what?

After going through this investigative process looking through topics and actions, we reached a conclusion of what must be done in a more concrete sense.

1. **Create a publisher to send the initial pose.**
    - The message type is `geometry_msgs/msg/PoseWithCovarianceStamped`.
2. **Create an action client to send navigation goals.**
    - The action type is `nav2_msgs/action/NavigateToPose`.

That's it!

## 48.2 Create the package

We'll start by creating a suitable **ROS2** package. We already now, from the previous investigation, what package we need to depend on.

```
cd ~/ros2_tutorial_workspace/src
ros2 pkg create python_package_that_uses_nav2 \
--build-type ament_python \
--dependencies rclpy geometry_msgs nav2_msgs
```

#### ros2 pkg create output

```
going to create a new package
package name: python_package_that_uses_nav2
destination directory: ~/ros2_tutorial_workspace/src
package format: 3
version: 0.0.0
description: TODO: Package description
maintainer: ['root <murilo.marinho@manchester.ac.uk>']
licenses: ['TODO: License declaration']
build type: ament_python
dependencies: ['rclpy', 'geometry_msgs', 'nav2_msgs']
creating folder ./python_package_that_uses_nav2
creating ./python_package_that_uses_nav2/package.xml
creating source folder
creating folder ./python_package_that_uses_nav2/python_package_that_uses_nav2
creating ./python_package_that_uses_nav2/setup.py
creating ./python_package_that_uses_nav2/setup.cfg
creating folder ./python_package_that_uses_nav2/resource
creating ./python_package_that_uses_nav2/resource/python_package_that_uses_nav2
creating ./python_package_that_uses_nav2/python_package_that_uses_nav2/__init__.py
creating folder ./python_package_that_uses_nav2/test
creating ./python_package_that_uses_nav2/test/test_copyright.py
```

(continues on next page)

(continued from previous page)

```
creating ./python_package_that_uses_nav2/test/test_flake8.py
creating ./python_package_that_uses_nav2/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
BSL-1.0
BSD-2.0
BSD-2-Clause
BSD-3-Clause
GPL-3.0-only
LGPL-3.0-only
MIT
MIT-0
```

## 48.3 Files

In this section, we will be creating or modifying the following highlighted files.

```
python_package_that_uses_nav2/
|-- package.xml
|-- python_package_that_uses_nav2
|   |-- __init__.py
|   |-- nav2_initial_pose_publisher_node.py
|   `-- nav2_navigate_to_pose_action_client_node.py
|-- resource
|   `-- python_package_that_uses_nav2
|-- setup.cfg
|-- setup.py
`-- test
    |-- test_copyright.py
    |-- test_flake8.py
    `-- test_pep257.py
```

## 48.4 Sending the initial pose to nav2

Given that this node is a simple publisher, we can jump straight to it.

nav2_initial_pose_publisher_node.py

```
1  import time
2
3  import rclpy
4  from rclpy.node import Node
5  from geometry_msgs.msg import PoseWithCovarianceStamped
6
7  class Nav2InitialPosePublisherNode(Node):
8      """A ROS2 Node that publishes the initial pose for nav2."""
9
10     def __init__(self):
```

(continues on next page)

(continued from previous page)

```
        super().__init__('nav2_initial_pose_publisher_node')

        self._topic = '/initialpose'

        self.publisher = self.create_publisher(
            msg_type=PoseWithCovarianceStamped,
            topic=self._topic,
            qos_profile=1)

        publisher_count = 0
        while publisher_count < 2:
            publisher_count = self.count_publishers(self._topic)
            print(f"Waiting for publisher to be connected to {self._topic}.")
            print(f"Publisher count is {publisher_count}.")
            time.sleep(1)

    def send_initial_pose_with_covariance(self):
        """Method to create the PoseWithCovarianceStamped."""

        print(f"Publishing pose to topic: {self._topic}.")

        pwcs = PoseWithCovarianceStamped()
        pwcs.header.stamp = self.get_clock().now().to_msg()
        pwcs.header.frame_id = 'map'

        pwcs.pose.pose.position.x = -2.1
        pwcs.pose.pose.position.y = -0.3
        pwcs.pose.pose.position.z = 0.0

        pwcs.pose.pose.orientation.w = 1.0
        pwcs.pose.pose.orientation.x = 0.0
        pwcs.pose.pose.orientation.y = 0.0
        pwcs.pose.pose.orientation.z = 0.0

        pwcs.pose.covariance[0] = 0.25
        pwcs.pose.covariance[7] = 0.25
        pwcs.pose.covariance[35] = 0.06853891909122467

        self.publisher.publish(pwcs)

def main(args=None):
    """
    The main function.
    :param args: Not used directly by the user, but used by ROS2 to configure
    certain aspects of the Node.
    """
    try:
        rclpy.init(args=args)
        node = Nav2InitialPosePublisherNode()
        node.send_initial_pose_with_covariance()
        rclpy.spin(node)
    except KeyboardInterrupt:
```

(continues on next page)

(continued from previous page)

```
        pass
    except Exception as e:
        print(e)

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

There might be two minor novelties in this example. The first one is similar to what we did in `tf2` when guaranteeing that the publisher was connected before attempting to publish for the first time. In this case, because in this demo **rviz2** is already connected to that topic, we check that there are two subscribers connected before proceeding with the initialisation.

```
        publisher_count = 0
        while publisher_count < 2:
            publisher_count = self.count_publishers(self._topic)
            print(f"Waiting for publisher to be connected to {self._topic}.")
            print(f"Publisher count is {publisher_count}.")
            time.sleep(1)
```

The second part that might be unfamiliar is the covariance used in the message. The covariance matrix holds the variance in the diagonal and off the diagonal any pair-wise variances between different states. All of this to say that the covariance will be used to say how confident we are in a given field of the pose and if their variation is correlated. In this case, we choose the same, or similar, values that were sent by **rviz2**. This is enough for our purposes. The covariance is highlighted below.

```
    def send_initial_pose_with_covariance(self):
        """Method to create the PoseWithCovarianceStamped."""

        print(f"Publishing pose to topic: {self._topic}.")

        pwcs = PoseWithCovarianceStamped()
        pwcs.header.stamp = self.get_clock().now().to_msg()
        pwcs.header.frame_id = 'map'

        pwcs.pose.pose.position.x = -2.1
        pwcs.pose.pose.position.y = -0.3
        pwcs.pose.pose.position.z = 0.0

        pwcs.pose.pose.orientation.w = 1.0
        pwcs.pose.pose.orientation.x = 0.0
        pwcs.pose.pose.orientation.y = 0.0
        pwcs.pose.pose.orientation.z = 0.0

        pwcs.pose.covariance[0] = 0.25
        pwcs.pose.covariance[7] = 0.25
        pwcs.pose.covariance[35] = 0.06853891909122467

        self.publisher.publish(pwcs)
```

It is important not to be distracted by the nested `pose`. The first one os the `PoseWithCovariance`, part of the `PoseWithCovarianceStamped`. The second one is the `Pose` part of the `PoseWithCovariance`.

That's it. This node is relatively simple: a publisher that publishes a single message. There will be other ways to achieve this. The one shown herein is the most standard one according to what has been shown in the tutorial.

## 48.5 Handling nav2 pose navigation actions

I'm conflicted whether this one is even easier. I suppose it's more complex, being an action client. However, it follows the same formula as we did in the action tutorial.

nav2_navigate_to_pose_action_client_node.py

```python
import rclpy
from rclpy.action import ActionClient
from rclpy.action.client import ClientGoalHandle
from rclpy.node import Node
from rclpy.task import Future

from geometry_msgs.msg import Pose, PoseStamped
from nav2_msgs.action import NavigateToPose

class Nav2NavigateToPoseActionClient(Node):
    """A ROS2 Node with an Action Client for Nav2NavigateToPoseActionClient."""

    def __init__(self):
        super().__init__('nav2_navigate_to_pose_action_client')

        self.action_client = ActionClient(self, NavigateToPose, '/navigate_to_pose')

        self.send_goal_future = None # This will be used in `send_goal`
        self.get_result_future = None # This will be used in 'goal_response_callback'

    def send_goal_async(self, desired_pose: Pose, behaviour_tree: str) -> None:
        goal_msg = NavigateToPose.Goal()
        goal_msg.pose.header.stamp = self.get_clock().now().to_msg()
        goal_msg.pose.header.frame_id = 'map'
        goal_msg.pose.pose = desired_pose
        goal_msg.behavior_tree = behaviour_tree

        while not self.action_client.wait_for_server(timeout_sec=1.0):
            self.get_logger().info(f'action {self.action_client} not available, waiting..
↪.')

        self.get_logger().info(f'Sending goal: {goal_msg}.')

        self.send_goal_future = self.action_client.send_goal_async(goal_msg, feedback_
↪callback=self.action_feedback_callback)
        self.send_goal_future.add_done_callback(self.goal_response_callback)

    def goal_response_callback(self, future: Future) -> None:
        goal: ClientGoalHandle = future.result()

        if not goal.accepted:
            self.get_logger().info('Goal was rejected by the server.')
            return
        self.get_logger().info('Goal was accepted by the server.')

        self.get_result_future = goal.get_result_async()
        self.get_result_future.add_done_callback(self.action_result_callback)
```

(continues on next page)

(continued from previous page)

```
    def action_result_callback(self, future: Future) -> None:
        result: NavigateToPose.Result = future.result()
        self.get_logger().info(f'Final position was: {result.result}.')

    def action_feedback_callback(self, feedback_msg: NavigateToPose.Feedback) -> None:
        feedback = feedback_msg.feedback
        self.get_logger().info(f'Received feedback distance: {feedback.distance_
↪remaining}.')


def main(args=None):
    """
    The main function.
    :param args: Not used directly by the user, but used by ROS2 to configure certain␣
↪aspects of the Node.
    """
    try:
        rclpy.init(args=args)

        node = Nav2NavigateToPoseActionClient()

        desired_pose = Pose()
        desired_pose.position.x = 1.0
        desired_pose.position.y = -1.0
        desired_pose.orientation.w = 1.0

        node.send_goal_async(desired_pose, "")

        rclpy.spin(node)
    except KeyboardInterrupt:
        pass
    except Exception as e:
        print(e)


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

We can highlight two possible pinch points. Starting by the `main()` function, we instantiate the node and create a desired pose, highlighted below. Indeed, we are going to use a stamped pose. However, it is simpler to grab the stamp from a node, so we do that in a method. We send an empty behaviour tree, because otherwise would be out of the scope of this tutorial.

```
        node = Nav2NavigateToPoseActionClient()

        desired_pose = Pose()
        desired_pose.position.x = 1.0
        desired_pose.position.y = -1.0
        desired_pose.orientation.w = 1.0

        node.send_goal_async(desired_pose, "")
```

Then, in our implementation we populate the fields of the goal with the pose we just created and the stamp obtained from the node. We receive an empty behaviour tree and that is also added to the goal.

```
    def send_goal_async(self, desired_pose: Pose, behaviour_tree: str) -> None:
        goal_msg = NavigateToPose.Goal()
        goal_msg.pose.header.stamp = self.get_clock().now().to_msg()
        goal_msg.pose.header.frame_id = 'map'
        goal_msg.pose.pose = desired_pose
        goal_msg.behavior_tree = behaviour_tree
```

Besides the slightly different action type, the process to make an action client is mostly unchanged.

## 48.6 Adjusting the `setup.py`

`setup.py`

```
from setuptools import find_packages, setup

package_name = 'python_package_that_uses_nav2'

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']),
    ],
    install_requires=['setuptools'],
    zip_safe=True,
    maintainer='root',
    maintainer_email='murilo.marinho@manchester.ac.uk',
    description='TODO: Package description',
    license='TODO: License declaration',
    extras_require={
        'test': [
            'pytest',
        ],
    },
    entry_points={
        'console_scripts': [
            'nav2_initial_pose_publisher_node = python_package_that_uses_nav2.nav2_initial_pose_publisher_node:main',
            'nav2_navigate_to_pose_action_client_node = python_package_that_uses_nav2.nav2_navigate_to_pose_action_client_node:main'
        ],
    },
)
```

## 48.7 Build and source

Before we proceed, let us build and source once.

```
cd ~/ros2_tutorial_workspace
colcon build
source install/setup.bash
```

**Note**

For additional explanation and troubleshooting tips, see *Always source after you build*.

**Warning**

**colcon** will *not* work properly if your terminal has an active **venv**.

## 48.8 Testing

The first step is to run the same demo as before, with `tb3_simulation_launch.py`. We can do that with the command below.

```
ros2 launch nav2_bringup \
tb3_simulation_launch.py \
use_sim_time:=True \
headless:=False \
sigterm_timeout:=120
```

Then, in another terminal, we can run the following command. This one will be rather tolerant of what it's started because it will wait for the correct number of publishers before publishing.

```
ros2 run python_package_that_uses_nav2 nav2_initial_pose_publisher_node
```

The following command should only be attempted after the initial pose was set. The navigation stack will not accept goals unless an initial estimate is given first.

```
ros2 run python_package_that_uses_nav2 nav2_navigate_to_pose_action_client_node
```

The qualitative behaviour can be seen in the following video.

**Exercises**

Other possible interactions with these interfaces could be as follows.

- What would change if the same node was supposed to both send the initial pose and, after that is done, send the goal?
- What would have to be modified if you had three goals, goal `a`, goal `b`, and goal `c`. Suppose that we initially send goal `a`. If the goal is reached, then go to goal `b`. If not reached, go to goal `c`. This is a very basic state machine. An embryo, but somewhat the motivation for behaviour tress.

CHAPTER

# FORTYNINE

# CYBERSECURITY

**Warning**

This topic is under construction and this might not even be its final form. Please feel free to open an issue if you spot any typos or other problems.

Let's start with a quick definition and a main reference for cybersecurity terminology.

> "Cybersecurity is the protection of information assets from harm."[1]

Anything related to robotics is based on my own experience, contextualised with the terminology of references.

Robotic systems are particularly susceptible to cybersecurity issues. In my experience, they tend to have relatively many *vulnerabilities*. For instance, robotic systems

- frequently have access to the internet through institutional/company networks. This is an increasing vulnerability given the popularity of cloud-based LLMs.
- might have an outdated mainstream operating system. Robotic systems might run older Linux variants and embedded Windows systems. The onboard software is rarely ever updated. Sometimes it can't be updated owing to specific device drivers.
- frequently are not (or cannot) managed by the central IT infrastructure as this could reduce their effectiveness or make them unpractical. Examples are firewalls and anti-viruses.
- could have manufacture's software with unauditable behaviour the is infrequently (or never) patched or documented.

This means that a *threat actor* can find many *attack vectors* to interfere with robotics infrastructure. The usual *impacts* we usually think of in terms of cybersecurity might be loss of data or data leakage. In a robotics context, loss of data could be the loss of photos of power lines stored in a vulnerable inspection drone. Data leakage might be, for example, videos of your house streamed via your cleaning robot or inspection videos taken by the latest robotic dog of a classified facility.

One additional impact of compromised robotic systems can be physical harm. Past are the days in which software was the thing you cursed and hardware the thing you punched. Now hardware can punch right back. Or even unprovoked.

Cybersecurity in robotic systems are (or should be) a real, and increasing, *risk*. Differently from a WiFi-connected refrigerator that might be more a display of wealth than actual need, our dear robots tend to benefit from (or need) some level of connectivity.

[1] Audun Jøsang. *Cybersecurity*. Springer International Publishing, Cham, Switzerland, 2024 edition, November 2024.

## 49.1 Basic terminology

The terminology can be slightly different between sources, see for instance[Page 379, 1] and[2]. Behold my incredible story-telling skills materialised into my own amalgamation among existing terminology.

Think of your robot's computer system as your house. The one in which you have a precious ring. In this context, each of the cyber-security terminology becomes as follows.

- Attacker: the bad person wearing a balaclava that wants to steal your precious.
- Vulnerability: issues with the security. For instance, leaving the front-door key under a potted plant.
- Attack vector: the way in which a vulnerability is exploited by a threat actor to steal your precious. For instance, the balaclava-person lifting up the potted plant, picking up the front-door key that was under the potted plant, and proceeding to open the door with it.
- Impact: the problem you'll have if you're attacked. In this case, your precious will be stolen. You'll cry. Your contents insurance payments will rise.
- Risk: impact put into perspective of likelihood. How many balaclava-clad people want your precious, really? Alternatively, being a neighbor of Sauron might indicate high risk of losing your precious.

## 49.2 What should I do?

The backbone of everything we will discuss in this topic will be *encryption*, part of cryptography which is very much literally "the science of secret writing"[Page 379, 1]. Other common-sense IT security actions also apply for robotic systems. These include backing things up, using firewalls, having strong passwords, keeping systems updated, not having things connected to the internet when they don't have to be, trusting noone, never storing personal data on a robot, and so on.

## 49.3 Scope of this tutorial

Among cybersecurity controls, two will stand out in this tutorial. Mainly network isolation and encryption, which are types of preventive controls, and backup, which is a type of corrective control.

**References**

[2] Henrique M D Santos. *Cybersecurity*. Chapman and Hall/CRC, Boca Raton, April 2022.

CHAPTER

# FIFTY

# NETWORK

> **Warning**
>
> This topic is under construction and this might not even be its final form. Please feel free to open an issue if you spot any typos or other problems.

There is plenty of software that can be used to protect your network. One of these is **`ufw`**, the uncomplicated firewall. These are good to some extent and I suppose will always be part of the security suite of companies and institutions. However, firewalls have a specific role and must be assisted by other forms of network safety.

## 50.1 A brief word on ROS2 networking

> **See also**
>
> Official information https://docs.ros.org/en/jazzy/Concepts/Intermediate/About-Domain-ID.html

**ROS2** has the so-called `ROS_DOMAIN_ID`. Although this concept exists, it should not be confused with a security measure. Each participant in the network can easily switch to another `ROS_DOMAIN_ID` without authentication or central management. It can be seen as merely a local filter.

To find **ROS2** nodes, **ROS2** makes use of multicast. The port used will depend on the `ROS_DOMAIN_ID`. More tinkering is needed if you want to specify which network interface will be used by **ROS2**. This could also depend on the underlying DDS implementation.

In **ROS2**, nodes communicate peer-to-peer, in the sense that there's no central node as it used to exist in **ROS1**. Each node will be assigned two ports to communicate. The specific port number will depend on the `ROS_DOMAIN_ID`. This is how the domains can be filtered. However, this is not a strict isolation.

> **See also**
>
> Official information https://docs.ros.org/en/jazzy/Concepts/Intermediate/About-Quality-of-Service-Settings.html

Another benefit of **ROS2** is being able to choose connection types. For instance, **ROS1** supported only TCP which can be slow. In contrast, **ROS2** accepts different levels of expectations in transmission protocols, and can suppose UDP which is usually more suitable for streaming information, such as camera images and sensor data.

Given that **ROS2** will also work through properly configured unreliable networks (e.g. Wi-Fi at some distance), you might be confronted with cases in which your nodes seemingly stop working. In this case, you have to choose the correct Quality of Service (QoS) settings to make sure your nodes can communicate reliably.

## 50.2 Network Topologies

**Important**

None of these measures provide full protection on their own. An attacker with an ethernet cable and access to a local port can easily access the entire robot infrastructure of unsuspecting laboratories. It is important to make sure that all robot software, whenever possible, is fully security patched. In addition, it's important to make sure that all user account passwords with a decent level of security. Leaving the robot with the factory password means an attacker can easily login locally or remotely.

An aspect that is often ignored in robotics labs are the physical network topologies, which are important for safety. This safety is not only for cybersecurity reasons. Yes, your robot's computer can be attacked if it's exposed. However, in development environments, you might inadvertently move someone else's robot. In these scenarios it will be much more efficient to physically isolate the robotic setup if many devices in the network can be shared with users with varying levels of network understanding.

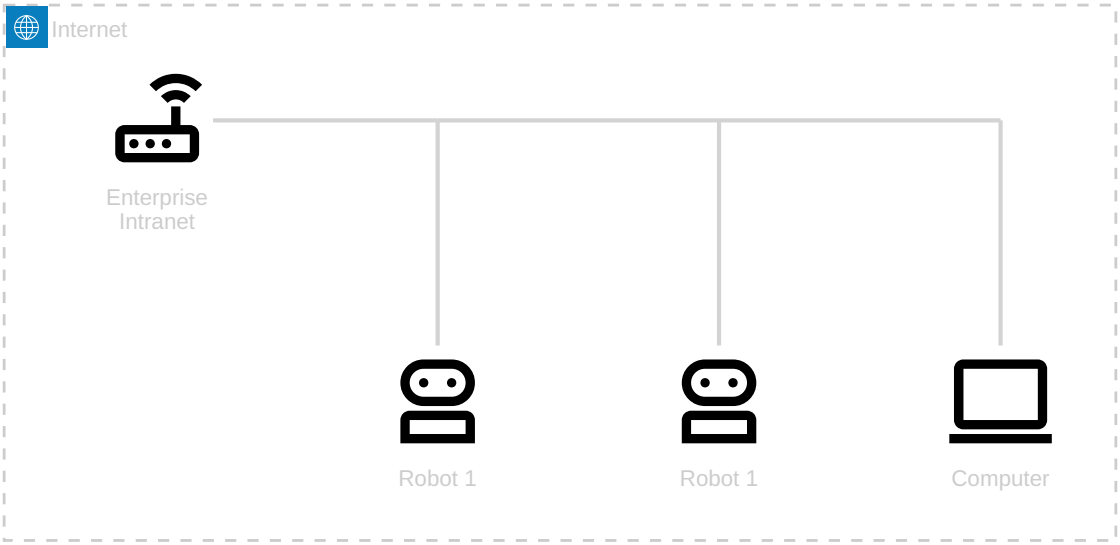
This section will be based on my experience with multiple robotic systems. Two representative examples are[1] and[2].

### 50.2.1 Case 1 - No isolation

A common network architecture in small companies and laboratories is shown below.

[1] Murilo Marques Marinho, Kanako Harada, Akio Morita, and Mamoru Mitsuishi. SmartArm: integration and validation of a versatile surgical robotic system for constrained workspaces. *Int. J. Med. Robot.*, 16(2):e2053, April 2020.

[2] Murilo Marques Marinho, Juan José Quiroz-Omaña, and Kanako Harada. A multiarm robotic platform for scientific exploration: its design, digital twins, and validation. *IEEE Robot. Autom. Mag.*, 31(4):10–20, December 2024.




In it, each computer and robot is directly connected to the internet. This setup is rather tempting because from the point-of-view of the robotics software developer they might want to have as much freedom as possible to develop their software as quickly as possible. Depending on the internet services they want to provide, they might even DMZ or

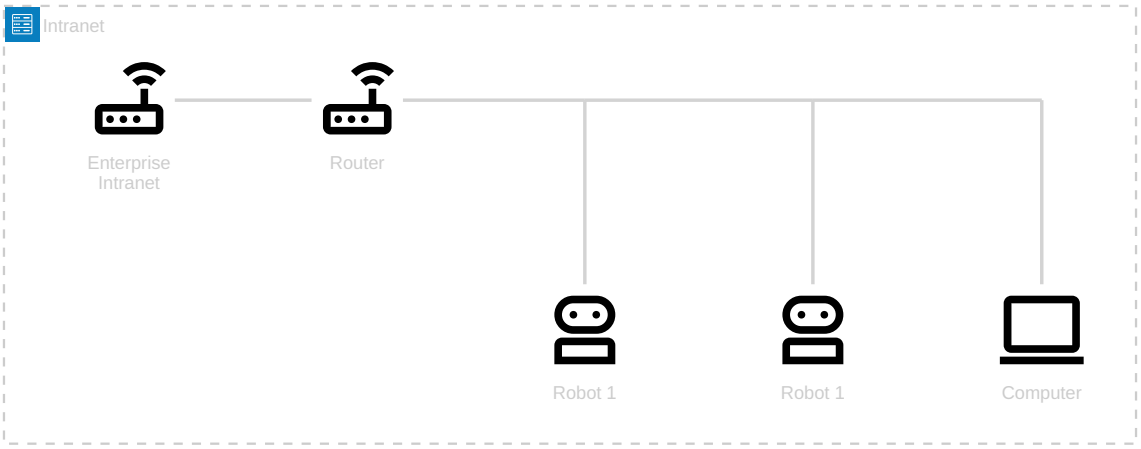
forward ports that give straight access to local computers. That is a huge security concern. If you leave port 22 open, it won't take long for someone to try to hack into your machine from somewhere in the world.

This setup leads to a high level of risk. As mentioned in the previous section, although you *might* be able to keep your computers up-to-date, the same is not usually possible for the robots' control computers. They could have vulnerabilities such as unpatched security bugs that will leave them exposed. In addition, the robots' might have been left with their original credentials and network settings, without a firewall. This means that an attacker could easily ssh into the robot.

Although from a robotics software developer point-of-view you might think that they might have little to gain from accessing a robot computer, that can easily be the first door into any other resource in the network. Just because the computer is attached somehow to a robot it does not make it less of a computer. It just, usually, makes it easier to exploit the computer.

### 50.2.2 Case 2 - Subnet isolation

A somewhat better network architecture is shown below, because there is one extra layer of isolation. The main difference here is that different parts of the company have their own subnets.



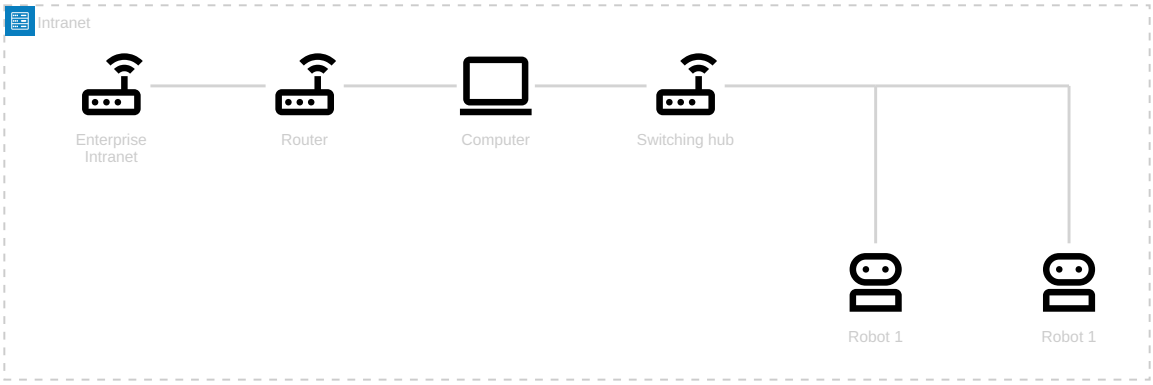

In this setup, you can imagine as each laboratory having their own router, that further isolate the network. In this case, not everyone will be able to see everyone else's devices. In this case, that can have many benefits. One of the benefits is to make sure that things such as ROS2 topics and services do not collide with someone else's.

For instance, suppose that you have your own robot and camera setup. If anyone else anywhere in the same network attempts to use the same topics and services, you can have a collision and unwanted behavior. You could have someone attempting to control the wrong robot arm from a distance. Or, the wrong camera stream is used.

Although `ROS_DOMAIN_ID` can help to filter out unwanted messages, it is too easy to set the wrong `ID`. More importantly, it is expected that people with a minimal understanding of networking would isolate their setup further.

### 50.2.3 Case 3 - Platform isolation

A possibly sufficient setup for most robotic demonstrators that need isolation is shown below.




In this setup, you can imagine each robotic demonstrator having their own, isolated, network. This can be easily achieved physically using a switching hub. This type of physical isolation of interfaces tends to be beneficial in development environments where software infrastructure is often changing.

This is also beneficial in terms of bandwidth. In large robotics laboratories when using a shared network the bandwidth will also be shared. This might be less of a problem when usage is not concentrated. However, it is common for these laboratories to have open days or engagement sessions with stakeholders. In these sessions, a large number of robots and sensors might need to share the same network. When the day arrives, it's already late to fix the network topology. Thence, it is recommended to reflect on situations such as these when assessing demonstrator needs.

**References**

CHAPTER

# FIFTYONE

# PUBLIC-KEY CRYPTOSYSTEMS

**Warning**

This topic is under construction and this might not even be its final form. Please feel free to open an issue if you spot any typos or other problems.

**Note**

This is obviously a simplified discussion of the topic.

## 51.1 ROS2 Security

**See also**

Official information: https://docs.ros.org/en/jazzy/Concepts/Intermediate/About-Security.html

There are facilities in **ROS2** to enable secure communication. The communication uses cryptography which will be shown in the following section. It is important to know that such capability exists. It will have a number of steps needed to create the necessary certificates to guarantee security among nodes.

After that is set up, nodes and program:*ros2cli* tools can be called with additional security.

**See also**

Official information https://docs.ros.org/en/jazzy/Tutorials/Advanced/Security/Introducing-ros2-security.html

Altering these settings can have unwanted side effects to other parts of the tutorial so we will leave this topic only briefly mentioned. In addition, understanding these topics requires first understanding public-key cryptography, shown below.

## 51.2 Scope of this section

Most of the important infrastructure for software for robotics relies heavily the concept of public-key cryptosystems. Many attribute this concept to as early as 1977, in one type of encryption algorithm (RSA) that is still in use.

Understanding the general idea of public-key encryption is rather simple. We do not need to enter the mathematical details.

## 51.3 Assumptions

- **Every bit of every message that anyone sends over the internet can be seen by any number of people and eavesdropped at all times.**
  - This is very much the case. Whatever we do is visible by the ISP and anyone in the route it takes from you and the server.
- **Only trust the identity of someone you can verify.**
  - IP addresses are dynamic, MAC addresses can be easily spoofed, and so on.

What does this mean, in practice? This means that using unencrypted messages someone can easily steal your information through the internet. For instance, suppose that you have to send your password to your bank. If this goes through a large number of unknown computers until reaching your bank's server (which might be anywhere, really), everyone will know that your password is `Ilov3mon3y£££`.

## 51.4 Creating keys

We are going to use the `ssh-keygen` tool to illustrate how this works in practice. This will also be useful for you when working with robots at least in two ways. First, when you need to connect remotely to a robot using `ssh`. Second, when you need to upload your version controlled-data, e.g. fancy backup, in remote servers such as GitHub.

```
ssh-keygen --help
```

This program is used to create the *encryption keys* used in public-key encryption. You will see few algorithms available, namely `dsa`, `ecdsa`, `ecdsa-sk`, `ed25519`, `ed25519-sk`, `rsa`. Please feel free to look them up to see the differences.

```
unknown option -- -
usage: ssh-keygen [-q] [-a rounds] [-b bits] [-C comment] [-f output_keyfile]
                  [-m format] [-N new_passphrase] [-O option]
                  [-t dsa | ecdsa | ecdsa-sk | ed25519 | ed25519-sk | rsa]
                  [-w provider] [-Z cipher]
       ssh-keygen -p [-a rounds] [-f keyfile] [-m format] [-N new_passphrase]
                   [-P old_passphrase] [-Z cipher]
       ssh-keygen -i [-f input_keyfile] [-m key_format]
       ssh-keygen -e [-f input_keyfile] [-m key_format]
       ssh-keygen -y [-f input_keyfile]
       ssh-keygen -c [-a rounds] [-C comment] [-f keyfile] [-P passphrase]
       ssh-keygen -l [-v] [-E fingerprint_hash] [-f input_keyfile]
       ssh-keygen -B [-f input_keyfile]
       ssh-keygen -D pkcs11
       ssh-keygen -F hostname [-lv] [-f known_hosts_file]
       ssh-keygen -H [-f known_hosts_file]
       ssh-keygen -K [-a rounds] [-w provider]
       ssh-keygen -R hostname [-f known_hosts_file]
       ssh-keygen -r hostname [-g] [-f input_keyfile]
       ssh-keygen -M generate [-O option] output_file
       ssh-keygen -M screen [-f input_file] [-O option] output_file
       ssh-keygen -I certificate_identity -s ca_key [-hU] [-D pkcs11_provider]
                  [-n principals] [-O option] [-V validity_interval]
```

(continues on next page)

(continued from previous page)

```
                [-z serial_number] file ...
      ssh-keygen -L [-f input_keyfile]
      ssh-keygen -A [-a rounds] [-f prefix_path]
      ssh-keygen -k -f krl_file [-u] [-s ca_public] [-z version_number]
                file ...
      ssh-keygen -Q [-l] -f krl_file [file ...]
      ssh-keygen -Y find-principals -s signature_file -f allowed_signers_file
      ssh-keygen -Y match-principals -I signer_identity -f allowed_signers_file
      ssh-keygen -Y check-novalidate -n namespace -s signature_file
      ssh-keygen -Y sign -f key_file -n namespace file [-O option] ...
      ssh-keygen -Y verify -f allowed_signers_file -I signer_identity
                -n namespace -s signature_file [-r krl_file] [-O option]
```

We are going to go mostly with the defaults. However, we need to be sure to create the keys in another file so that it does not create problems with your own system. We will create the keys with the following command.

```
ssh-keygen -f example_ed25519
```

Press `Enter` a couple of times to create a key without a passphrase. The passphrase will be useful if someone might have access to your computer but isn't too poorly intended. It's a thin layer of extra safety. We can skip it for the purposes of this tutorial.

```
Generating public/private ed25519 key pair.
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in example_ed25519
Your public key has been saved in example_ed25519.pub
The key fingerprint is:
SHA256:Hb2mpOg1Ya5NUPXzl8JVFzmDdsRtj6JcECsIqnIN/EU root@c869fbce1a11
The key's randomart image is:
+--[ED25519 256]--+
|    . E   o.  +o*|
| . . o . ..+ o *=|
|  +   o o o.= .++|
| . + . . o .++o o|
|o . o . S.ooo+...|
|..     = +oo  .. |
|      . * .      |
|     . = .       |
|      o .        |
+----[SHA256]-----+
```

**Danger**

I will show the keys and share the results here, because this is obviously a tutorial. However, you should always be careful with your keys. If your private key, explained below, is shared, it is compromised. This means you will have to create new keys and re-add them to any relevant systems before someones steals your precious.

In the same folder in which the command was executed, we will see two files `example_ed25519.pub` and `example_ed25519`. The first one, with the extension `.pub` is the **public** key. It is a string of pretty much random characters. The one I just created is shown below.

```
cat example_ed25519.pub
```

The command above will output the public key.

```
ssh-ed25519 AAAAC3NzaC1lZDI1NTE5AAAAIAU1eT9YRbosGzdgxbN38pb+jH3nwXfKwtJPCbQgLY3n␣
↪root@c869fbce1a11
```

We can also see the contents of the private key, similarly. Let's keep the suspense for now, to stay in the proper narrative line.

## 51.5 How do we send a secret message?

Let us start by installing a supporting program, called **age**. This program will allow us to use our encryption keys to encrypt specific messages. Usually, **ssh** would handle those more complex connections for us. We are only using **age** to illustrate the ideas behind public-key cryptography.

```
sudo apt-get update
sudo apt-get install age
```

Suppose that you have a very important text to encrypt and send to your best friend. However, you might be slightly tsundere about it and don't want anyone else to see it.

Let's create the secret message, called `secret_message.txt`.

```
echo "This is the best tutorial I have ever seen thanks Murilo for being so great." >␣
↪secret_message.txt
```

Now, the contents are obviously visible to anyone. You want to be sure that only your friend can see it. Therefore, you grab your friend's public key. Let's use `example_ed25519.pub` I showed as an example for this. Again, the public key is not a secret, it is meant to be *public*, that is, seen by other people. People, then, only need to be able to trust that you are the holder of this specific key pair.

Let's use our cool program, **age**, that allows us to play with encryption. Without further ado, let's go with it.

```
age -R example_ed25519.pub secret_message.txt > secret_message.txt.age
```

Now, the message has been encrypted. We can see the contents of `secret_message.txt.age` as follows. This is what you will send your friend. You can even copy and paste it on something fully public, like an Instagram post. Your other friends might be concerned seeing that but otherwise your message is still a secret.

> **Note**
>
> The encrypted messages will not always show well in the terminal or websites like this. The encrypted messages are not meant to be human readable.

```
age-encryption.org/v1
-> ssh-ed25519 Hb2mpA xxu02M6ZYcgdvpjQ0OtObLX67P+C1cr/6AefZ+w/g1I
R3mtqZ9x8sz54/j8g2qY/2EJvkQytXZnLOPmwTziY+w
--- GiV9DHB2gAr4V6ZFhIMPH81sDEEfjCGocYImCYD/lhA
1
  ,lj
       &
```

(continues on next page)

(continued from previous page)

```
          CFgfVY*ez5
                              bu>ð n=g}M]     e#^Zmuhjjol_root@c869fbce1a11
```

The magic is that nobody will be able to decrypt the message unless the have the *private* key. Let's summarise what just happened.

- Anyone who wants to participate on an encrypted conversation make their *public* encryption key available.
- Anyone who wants to send them a secret message uses that public key to encrypt the message.
- Only the person with the *private* can decrypt the message.

Therefore, they are able to send secret messages through a compromised and public channel, e.g. the internet. As long as the public key is correct, we do not have to trust the identity of anyone either. **ONLY** someone who holds the private key will be able to read the true contents of the original message.

## 51.6 How do we read a secret message?

So, suppose that you sent the secret message, `secret_message.txt.age`, above to your friend. Your friend will now have to use their private key to decrypt the message. Again, suppose that the keys we just created are safely held by your friend. If they want to read the contents of the secret message they will have to use the matching private key.

```
cat example_ed25519
```

The output is somewhat similar to the public key. I will not show it here to reinforce that it is something you should keep private. Your friend will use the private key `example_ed25519` and the secret file they just received from you `secret_message.txt.age`.

```
age -d -i example_ed25519 secret_message.txt.age > secret_message_decrypted.txt
```

Then, you can confirm that it has been decrypted with the following command.

```
cat secret_message_decrypted.txt
```

In which the original message is restored.

```
This is the best tutorial I have ever seen thanks Murilo for being so great.
```

You can confirm that you won't be able to decrypt anything that was encrypted with the example public key. That is because I haven't showed you the private key. I'm pretty sure I lost it too.

> **Danger**
>
> If you lose your private key, any information you had only in encrypted form is lost forever. FOREVER.

## 51.7 Wait, what?

Yes, the magic happens because of the key pair. Each key, alone, is weak (actually, meaningless). Keys together, strong.

So, for an encrypted conversation between two participants, there will be two key pairs. One for each participant. Suppose that participant A has public key A and private key A. Then, suppose that participant B has public key B and private key B.

The flow in this case would be as follows.

- Participant A will use public key B to encrypt a message.
- It will be sent to participant B.
- Participant B will decrypt the message with private key B.
- Participant B will use public key A to encrypt a response.
- It will be sent to participant A.
- Participant A will decrypt the message with private key A.

Because the public keys can be freely seen through a public channel, e.g., the internet, the information exchanged is safe. This does not mean that encryption is not crackable. With enough time and opportunities to attack, a private key can theoretically be eventually guessed. This is to loosely one of the ideas behind cryptocurrencies, in which a hash must be guessed.

## 51.8 Exercises

We can think of decryption and encryption exercises that help illustrate the process.

### 51.8.1 Decryption

Suppose that you receive the following message, which has been encrypted with your public key. It was clearly done so using **age**.

> **Caution**
>
> Please download the file using the link below. It cannot be correctly displayed on this webpage. Encrypted messages are not meant to be human readable.

`exercise_message.txt.age`

Below is the pairing private key, which you should never ever share with anyone for any reason. Anyone with this key can decode the message. If anyone can decode the message, then it's no longer a secret.

> **Caution**
>
> I will share the private key here because this is a tutorial. DO NOT SHARE YOUR PRIVATE KEY WITH ANYONE.

`exercise_ed25519`

```
-----BEGIN OPENSSH PRIVATE KEY-----
b3BlbnNzaC1rZXktdjEAAAAABG5vbmUAAAAEbm9uZQAAAAAAAAABAAAAMwAAAAtzc2gtZW
QyNTUxOQAAACCmUlsdqJr1dJUzSX2rSctLWrifN3FtXw0bhd+ACRet/QAAAKDv9N2x7/Td
sQAAAAtzc2gtZWQyNTUxOQAAACCmUlsdqJr1dJUzSX2rSctLWrifN3FtXw0bhd+ACRet/Q
AAAEDmqGfF7PfgEOBtbzsuZqocWgSAAmX4+zqMmhZZ+NBZDKZSWx2omvV0lTNJfatJy0ta
uJ83cW1fDRuF34AJF639AAAAHWI0MDYxN21tQMOvwr/CvcOvwr/CvcOvwr/CvWVV
-----END OPENSSH PRIVATE KEY-----
```

What are the decrypted contents of this message?

### 51.8.2 Encryption

Suppose that you have the following public key.

`exercise_ed25519.pub`

```
ssh-ed25519 AAAAC3NzaC1lZDI1NTE5AAAAIKZSWx2omvV0lTNJfatJy0tauJ83cW1fDRuF34AJF639␣
↪b40617mm@ï¿½ï¿½ï¿½eU
```

Encrypt the message below. Well, anything will do, really.

> "Sorry kid, you got the gift but it looks like you are waiting for something, next life maybe who knows".

The private key for you to test is the same as the previous example.

CHAPTER

# FIFTYTWO

# CREATING C++ NODES (FOR `AMENT_CMAKE`)

**The C++ binary block for `ament_cmake`**

## TL;DR

When adding a new Node in an existing `CMakeLists.txt`, you might benefit from using the following template.

Remember to:

1. Add **ALL** dependencies (including ROS2 ones) with `find_package`, if applicable.

```
# find dependencies
find_package(ament_cmake REQUIRED)
find_package(rclcpp REQUIRED)
```

2. Change `print_forever_node` to the name of your Node.
3. Add all source files to `add_executable`.
4. Add all ROS2 dependencies of this binary to `ament_target_dependencies`.
5. Add any other (**NOT ROS2**) libraries to `target_link_libraries`.

```
############################
# CPP Binary Block [BEGIN] #
# vvvvvvvvvvvvvvvvvvvvvvvv #
# https://ros2-tutorial.readthedocs.io/en/latest/
# While we cant use blocks https://cmake.org/cmake/help/latest/command/block.html
↪#command:block
# we use set--unset
set(RCLCPP_LOCAL_BINARY_NAME print_forever_node)

add_executable(${RCLCPP_LOCAL_BINARY_NAME}
    src/print_forever_node_main.cpp
    src/print_forever_node.cpp

    )

ament_target_dependencies(${RCLCPP_LOCAL_BINARY_NAME}
    rclcpp

    )

target_link_libraries(${RCLCPP_LOCAL_BINARY_NAME}

    )

target_include_directories(${RCLCPP_LOCAL_BINARY_NAME} PUBLIC
    $<BUILD_INTERFACE:${CMAKE_CURRENT_SOURCE_DIR}/include>
    $<INSTALL_INTERFACE:include>)

target_compile_features(${RCLCPP_LOCAL_BINARY_NAME} PUBLIC c_std_99 cxx_std_17)
```

## 52.1 Create the package

> **Warning**
>
> We'll skip using the `--node-name` option to create the Node template, because, currently, it generates a Node and a `CMakeLists.txt` different from my advice.

```
cd ~/ros2_tutorial_workspace/src
ros2 pkg create cpp_package_with_a_node \
--build-type ament_cmake \
--dependencies rclcpp
```

which outputs

### ros2 pkg create output

```
going to create a new package
package name: cpp_package_with_a_node
destination directory: /home/murilo/ROS2_Tutorial/ros2_tutorial_workspace/src
package format: 3
version: 0.0.0
description: TODO: Package description
maintainer: ['murilo <murilomarinho@ieee.org>']
licenses: ['TODO: License declaration']
build type: ament_cmake
dependencies: ['rclcpp']
creating folder ./cpp_package_with_a_node
creating ./cpp_package_with_a_node/package.xml
creating source and include folder
creating folder ./cpp_package_with_a_node/src
creating folder ./cpp_package_with_a_node/include/cpp_package_with_a_node
creating ./cpp_package_with_a_node/CMakeLists.txt

[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 identitifers:
Apache-2.0
BSL-1.0
BSD-2.0
BSD-2-Clause
BSD-3-Clause
GPL-3.0-only
LGPL-3.0-only
MIT
MIT-0
```

## 52.2 Package-related sources

**In this step, we'll work on these.**

```
cpp_package_with_a_node
├── CMakeLists.txt
├── include
│   └── cpp_package_with_a_node
│       └── .placeholder
├── package.xml
└── src
    ├── print_forever_node.cpp
    ├── print_forever_node.hpp
    └── print_forever_node_main.cpp
```

The files already exist, we just need to modify them as follows

### package.xml

The `package.xml` works the same way as in **ament_python**, with the exception of the two lines about **ament_cmake** shown below.

`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>cpp_package_with_a_node</name>
  <version>0.0.0</version>
  <description>TODO: Package description</description>
  <maintainer email="murilomarinho@ieee.org">murilo</maintainer>
  <license>TODO: License declaration</license>

  <buildtool_depend>ament_cmake</buildtool_depend>

  <depend>rclcpp</depend>

  <test_depend>ament_lint_auto</test_depend>
  <test_depend>ament_lint_common</test_depend>

  <export>
    <build_type>ament_cmake</build_type>
  </export>
</package>
```

### CMakeLists.txt

A *one-size-fits-most* solution is shown below. For each new Node we add a block to the `CMakeLists.txt` with the following format.

`CMakeLists.txt`

```
cmake_minimum_required(VERSION 3.8)
project(cpp_package_with_a_node)

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

# find dependencies
find_package(ament_cmake REQUIRED)
find_package(rclcpp REQUIRED)

############################
# CPP Binary Block [BEGIN] #
# vvvvvvvvvvvvvvvvvvvvvvvv #
# https://ros2-tutorial.readthedocs.io/en/latest/
# While we cant use blocks https://cmake.org/cmake/help/latest/command/block.html
↪#command:block
# we use set--unset
set(RCLCPP_LOCAL_BINARY_NAME print_forever_node)

add_executable(${RCLCPP_LOCAL_BINARY_NAME}
    src/print_forever_node_main.cpp
    src/print_forever_node.cpp

    )

ament_target_dependencies(${RCLCPP_LOCAL_BINARY_NAME}
    rclcpp

    )

target_link_libraries(${RCLCPP_LOCAL_BINARY_NAME}

    )

target_include_directories(${RCLCPP_LOCAL_BINARY_NAME} PUBLIC
    $<BUILD_INTERFACE:${CMAKE_CURRENT_SOURCE_DIR}/include>
    $<INSTALL_INTERFACE:include>)

target_compile_features(${RCLCPP_LOCAL_BINARY_NAME} PUBLIC c_std_99 cxx_std_17)

install(TARGETS ${RCLCPP_LOCAL_BINARY_NAME}
    DESTINATION lib/${PROJECT_NAME})

unset(RCLCPP_LOCAL_BINARY_NAME)
# ^^^^^^^^^^^^^^^^^^^^^^^^ #
# CPP Binary Block [END] #
##########################

if(BUILD_TESTING)
    find_package(ament_lint_auto REQUIRED)
    # the following line skips the linter which checks for copyrights
    # comment the line when a copyright and license is added to all source files
```

(continues on next page)

(continued from previous page)

```
    set(ament_cmake_copyright_FOUND TRUE)
    # the following line skips cpplint (only works in a git repo)
    # comment the line when this package is in a git repo and when
    # a copyright and license is added to all source files
    set(ament_cmake_cpplint_FOUND TRUE)
    ament_lint_auto_find_test_dependencies()
endif()

ament_package()
```

## 52.3 Making C++ ROS2 Nodes

**(Murilo's) `rclcpp` best practices**

For each new C++ Node, we make three files following the style below.

For a Node called `print_forever_node` we have

1. `src/print_forever_node.hpp` with the Node's class definition. In general, this is not exported to other packages, so it should not be in the package's `include` folder.
2. `src/print_forever_node.cpp` with the Node's class implementation.
3. `src/print_forever_node_main.cpp` with the Node's main function implementation.

**In this step, we'll work on these.**

```
cpp_package_with_a_node
├── CMakeLists.txt
├── include
│   └── cpp_package_with_a_node
│       └── .placeholder
├── package.xml
└── src
    ├── print_forever_node.cpp
    ├── print_forever_node.hpp
    └── print_forever_node_main.cpp
```

These files do not exist, so we'll create them.

### folder

Create the folder.

```
cd ~/ros2_tutorial_workspace/src/cpp_package_with_a_node
mkdir src
```

### src/..._node.hpp

Similar to what we did in Python, we inherit from `rclcpp::Node`. Whatever is different is owing to differences in languages.

print_forever_node.hpp

```hpp
#pragma once

#include <memory>
#include <rclcpp/rclcpp.hpp>

/**
 * @brief A ROS2 Node that prints to the console periodically, but in C++.
 */
class PrintForeverNode: public rclcpp::Node
{
private:
    double timer_period_;
    int print_count_;
    //also equivalent to rclcpp::TimerBase::SharedPtr
    std::shared_ptr<rclcpp::TimerBase> timer_;

    void _timer_callback();
public:
    PrintForeverNode();

};
```

### src/..._node.cpp

The implementation has nothing special, just don't forget to initialize the parent class, `rclcpp::Node`, with the name of the node.

print_forever_node.cpp

```cpp
#include "print_forever_node.hpp"

/**
 * @brief PrintForeverNode::PrintForeverNode Default constructor.
 */
PrintForeverNode::PrintForeverNode():
    rclcpp::Node("print_forever_cpp"),
    timer_period_(0.5),
    print_count_(0)
{
    //(Smart) pointers at the one thing that it doesn't matter much if they are not␣
↪initialized in the member initializer list
    //and this is a bit more readable.
    timer_ = create_wall_timer(
                std::chrono::milliseconds(long(timer_period_*1e3)),
                std::bind(&PrintForeverNode::_timer_callback, this) //Note here the use␣
↪of std::bind to build a single argument
                );
}
```

(continues on next page)

(continued from previous page)

```

/**
 * @brief PrintForeverNode::_timer_callback periodically prints class info using RCLCPP_
↪INFO.
 */
void PrintForeverNode::_timer_callback()
{
    RCLCPP_INFO_STREAM(get_logger(),
                       std::string("Printed ") +
                       std::to_string(print_count_) +
                       std::string(" times.")
                       );
    print_count_++;
}
```

### src/..._main.cpp

Given that we are using `rclcpp::spin()`, there is nothing special here either. Just remember to not mess up the `std::make_shared` and always use perfect forwarding. The `rclcpp::spin()` handles the `SIGINT` when we, for example, press `CTRL+C` on the terminal. It is not perfect, but it does the trick for simple nodes like this one.

`print_forever_node_main.cpp`

```
#include <rclcpp/rclcpp.hpp>

#include "print_forever_node.hpp"

int main(int argc, char** argv)
{
    rclcpp::init(argc,argv);

    try
    {
        auto node = std::make_shared<PrintForeverNode>();

        rclcpp::spin(node);
    }
    catch (const std::exception& e)
    {
        std::cerr << std::string("::Exception::") << e.what();
    }

    return 0;
}
```

## 52.4 Add a `.placeholder` if your `include/<PACKAGE_NAME>` is empty

> **Warning**
>
> If you don't do this and add this package as a git repository without any files on the `include/`, **`CMake`** might return with an error when trying to compile your package.

```
cpp_package_with_a_node
├── CMakeLists.txt
├── include
│   └── cpp_package_with_a_node
│       └── .placeholder
├── package.xml
└── src
    ├── print_forever_node.cpp
    ├── print_forever_node.hpp
    └── print_forever_node_main.cpp
```

Empty directories will not be tracked by git. A file has to be added to the index. We can create an empty file in the `include` folder as follows

```
cd ~/ros2_tutorial_workspace/src/cpp_package_with_a_node/src
touch include/cpp_package_with_a_node/.placeholder
```

## 52.5 Running a C++ Node

As simple as it has always been, see *Running a node (ros2 run)*.

```
ros2 run cpp_package_with_a_node print_forever_node
```

which returns

```
[INFO] [1688620414.406930812] [print_forever_node]: Printed 0 times.
[INFO] [1688620414.906890884] [print_forever_node]: Printed 1 times.
[INFO] [1688620415.406907619] [print_forever_node]: Printed 2 times.
[INFO] [1688620415.906881003] [print_forever_node]: Printed 3 times.
[INFO] [1688620416.406900108] [print_forever_node]: Printed 4 times.
[INFO] [1688620416.906886691] [print_forever_node]: Printed 5 times.
[INFO] [1688620417.406881803] [print_forever_node]: Printed 6 times.
[INFO] [1688620417.906858551] [print_forever_node]: Printed 7 times.
[INFO] [1688620418.406894922] [print_forever_node]: Printed 8 times.
```

and we'll use CTRL+C to stop the node, resulting in

```
[INFO] [1688620418.725674401] [rclcpp]: signal_handler(signum=2)
```

CHAPTER

# FIFTYTHREE

# CREATING C++ LIBRARIES (FOR `AMENT_CMAKE`)

**The C++ library block for `ament_cmake`**

## TL;DR

When your project exports a library, you might benefit from using the following template. Note that there is, in general, no reason to define multiple libraries. A single shared library can hold all the content that you want to export from a package, hence the library named `${PROJECT_NAME}`.

Remember to

1. Add all exported headers to `include/<PACKAGE_NAME>` otherwise other packages cannot see it.
2. Add all source files of the library to `add_library`.
3. Add all ROS2 dependencies of the library to `ament_target_dependencies`.
4. Add **ALL** dependencies for which you used `find_package` to `ament_export_dependencies`, otherwise dependencies might become complex for projects that use your library.
5. Add any other (**NOT ROS2**) libraries to `target_link_libraries`.

```
####################################
# CPP Shared Library Block [BEGIN] #
# vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv #
# https://ros2-tutorial.readthedocs.io/en/latest/
# The most common use case is to merge everything you need to export
# into the same shared library called ${PROJECT_NAME}.
add_library(${PROJECT_NAME} SHARED
    src/sample_class.cpp

    )

ament_target_dependencies(${PROJECT_NAME}
    rclcpp

    )

target_include_directories(${PROJECT_NAME}
    PUBLIC
    $<BUILD_INTERFACE:${CMAKE_CURRENT_SOURCE_DIR}/include>
    $<INSTALL_INTERFACE:include>)

ament_export_targets(export_${PROJECT_NAME} HAS_LIBRARY_TARGET)
ament_export_dependencies(
```

```
    rclcpp
    Eigen3
    Qt5Core

    )

target_link_libraries(${PROJECT_NAME}
    Qt5::Core

    )

install(
    DIRECTORY include/
    DESTINATION include
    )

install(
    TARGETS ${PROJECT_NAME}
    EXPORT export_${PROJECT_NAME}
    LIBRARY DESTINATION lib
    ARCHIVE DESTINATION lib
    RUNTIME DESTINATION bin
    INCLUDES DESTINATION include
    )
# ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^ #
# CPP Shared Library Block [END] #
##################################
```

The base package can be created with

```
cd ~/ros2_tutorial_workspace/src
ros2 pkg create cpp_package_with_a_library \
--build-type ament_cmake \
--dependencies rclcpp
```

resulting in the following output

### ros2 pkg create output

```
ros2 pkg create cpp_package_with_a_library \
--build-type ament_cmake \
--dependencies rclcpp
going to create a new package
package name: cpp_package_with_a_library
destination directory: /home/murilo/ROS2_Tutorial/ros2_tutorial_workspace/src
package format: 3
version: 0.0.0
description: TODO: Package description
maintainer: ['murilo <murilomarinho@ieee.org>']
licenses: ['TODO: License declaration']
build type: ament_cmake
dependencies: ['rclcpp']
```

(continues on next page)

(continued from previous page)

```
creating folder ./cpp_package_with_a_library
creating ./cpp_package_with_a_library/package.xml
creating source and include folder
creating folder ./cpp_package_with_a_library/src
creating folder ./cpp_package_with_a_library/include/cpp_package_with_a_library
creating ./cpp_package_with_a_library/CMakeLists.txt

[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 identitifers:
Apache-2.0
BSL-1.0
BSD-2.0
BSD-2-Clause
BSD-3-Clause
GPL-3.0-only
LGPL-3.0-only
MIT
MIT-0
```

## 53.1 Package-related sources

**In this step, we'll work on these.**

```
cpp_package_with_a_library
├── CMakeLists.txt
├── include
│   └── cpp_package_with_a_library
│       └── sample_class.hpp
├── package.xml
└── src
    ├── sample_class.cpp
    ├── sample_class_local_node.cpp
    ├── sample_class_local_node.hpp
    └── sample_class_local_node_main.cpp
```

The files already exist, we just need to modify them as follows

### package.xml

Nothing new here.

`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>cpp_package_with_a_library</name>
  <version>0.0.0</version>
```

(continues on next page)

(continued from previous page)

```
  <description>TODO: Package description</description>
  <maintainer email="murilomarinho@ieee.org">murilo</maintainer>
  <license>TODO: License declaration</license>

  <buildtool_depend>ament_cmake</buildtool_depend>

  <depend>rclcpp</depend>

  <test_depend>ament_lint_auto</test_depend>
  <test_depend>ament_lint_common</test_depend>

  <export>
    <build_type>ament_cmake</build_type>
  </export>
</package>
```

### CMakeLists.txt

A *one-size-fits-most* solution is shown below. We don't need to add multiple libraries, so a single library can hold all the content you might want to export. The user of the library will see it nicely split by your header files, so it will be as neat as you make them.

Note that, because the local Node depends on the library being exported by this project, it needs to explicitly link to it.

`CMakeLists.txt`

```
cmake_minimum_required(VERSION 3.8)
project(cpp_package_with_a_library)

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

# find dependencies
find_package(ament_cmake REQUIRED)
find_package(rclcpp REQUIRED)
find_package(Eigen3 REQUIRED)
find_package(Qt5Core REQUIRED)

####################################
# CPP Shared Library Block [BEGIN] #
# vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv #
# https://ros2-tutorial.readthedocs.io/en/latest/
# The most common use case is to merge everything you need to export
# into the same shared library called ${PROJECT_NAME}.
add_library(${PROJECT_NAME} SHARED
    src/sample_class.cpp

    )

ament_target_dependencies(${PROJECT_NAME}
    rclcpp

```

(continues on next page)

(continued from previous page)

```
    )

target_include_directories(${PROJECT_NAME}
    PUBLIC
    $<BUILD_INTERFACE:${CMAKE_CURRENT_SOURCE_DIR}/include>
    $<INSTALL_INTERFACE:include>)

ament_export_targets(export_${PROJECT_NAME} HAS_LIBRARY_TARGET)
ament_export_dependencies(
    rclcpp
    Eigen3
    Qt5Core

    )

target_link_libraries(${PROJECT_NAME}
    Qt5::Core

    )

install(
    DIRECTORY include/
    DESTINATION include
    )

install(
    TARGETS ${PROJECT_NAME}
    EXPORT export_${PROJECT_NAME}
    LIBRARY DESTINATION lib
    ARCHIVE DESTINATION lib
    RUNTIME DESTINATION bin
    INCLUDES DESTINATION include
    )
# ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^ #
# CPP Shared Library Block [END] #
##################################

############################
# CPP Binary Block [BEGIN] #
# vvvvvvvvvvvvvvvvvvvvvvvv #
# https://ros2-tutorial.readthedocs.io/en/latest/
# While we cant use blocks https://cmake.org/cmake/help/latest/command/block.html
↪#command:block
# we use set--unset
set(RCLCPP_LOCAL_BINARY_NAME sample_class_local_node)

add_executable(${RCLCPP_LOCAL_BINARY_NAME}
    src/sample_class_local_node_main.cpp
    src/sample_class_local_node.cpp
    src/sample_class.cpp

    )
```

(continues on next page)

(continued from previous page)

```
79
80  ament_target_dependencies(${RCLCPP_LOCAL_BINARY_NAME}
81      rclcpp
82
83      )
84
85  target_link_libraries(${RCLCPP_LOCAL_BINARY_NAME}
86      ${PROJECT_NAME}
87
88      )
89
90  target_include_directories(${RCLCPP_LOCAL_BINARY_NAME} PUBLIC
91      $<BUILD_INTERFACE:${CMAKE_CURRENT_SOURCE_DIR}/include>
92      $<INSTALL_INTERFACE:include>)
93
94  target_compile_features(${RCLCPP_LOCAL_BINARY_NAME} PUBLIC c_std_99 cxx_std_17)
95
96  install(TARGETS ${RCLCPP_LOCAL_BINARY_NAME}
97      DESTINATION lib/${PROJECT_NAME})
98
99  unset(RCLCPP_LOCAL_BINARY_NAME)
100 # ^^^^^^^^^^^^^^^^^^^^^^^^ #
101 # CPP Binary Block [END] #
102 ##########################
103
104 if(BUILD_TESTING)
105     find_package(ament_lint_auto REQUIRED)
106     # the following line skips the linter which checks for copyrights
107     # comment the line when a copyright and license is added to all source files
108     set(ament_cmake_copyright_FOUND TRUE)
109     # the following line skips cpplint (only works in a git repo)
110     # comment the line when this package is in a git repo and when
111     # a copyright and license is added to all source files
112     set(ament_cmake_cpplint_FOUND TRUE)
113     ament_lint_auto_find_test_dependencies()
114 endif()
115
116 ament_package()
```

## 53.2 Library sources

**In this step, we'll work on these.**

```
cpp_package_with_a_library
├── CMakeLists.txt
├── include
│   └── cpp_package_with_a_library
│       └── sample_class.hpp
├── package.xml
└── src
    ├── sample_class.cpp
```

```
    ├── sample_class_local_node.cpp
    ├── sample_class_local_node.hpp
    └── sample_class_local_node_main.cpp
```

### sample_class.hpp

A class that does a bunch of nothing, but that depends on Eigen3 and Qt, as an example.

sample_class.hpp

```hpp
#pragma once

#include <ostream>

#include <QString>
#include <eigen3/Eigen/Dense>

class SampleClass
{
private:
    int a_private_member_;
    const QString a_private_qt_string_;
    const Eigen::MatrixXd a_private_eigen3_matrix_;

public:
    SampleClass();

    int get_a_private_member() const;
    void set_a_private_member(int value);
    std::string to_string() const;

    static double sum_of_squares(const double&a, const double& b);
};

std::ostream& operator<<(std::ostream& os, const SampleClass& sc);
```

### sample_class.cpp

sample_class.cpp

```cpp
#include <cpp_package_with_a_library/sample_class.hpp>



/**
 * @brief SampleClass::SampleClass the default constructor.
 */
SampleClass::SampleClass():
    a_private_qt_string_("I am a QString"),
    a_private_eigen3_matrix_((Eigen::Matrix2d() << 1,2,3,4).finished())
{

}
```

(continues on next page)

(continued from previous page)

```
14
15  /**
16   * @brief SampleClass::get_a_private_member.
17   * @return an int with the value of a_private_member_.
18   */
19  int SampleClass::get_a_private_member() const
20  {
21      return a_private_member_;
22  }
23
24  /**
25   * @brief SampleClass::set_a_private_member.
26   * @param value The new value for a_private_member_.
27   */
28  void SampleClass::set_a_private_member(int value)
29  {
30      a_private_member_ = value;
31  }
32
33  /**
34   * @brief SampleClass::sum_of_squares.
35   * @param a The first number.
36   * @param b The second number.
37   * @return a*a + 2*a*b + b*b.
38   */
39  double SampleClass::sum_of_squares(const double &a, const double &b)
40  {
41      return a*a + 2*a*b + b*b;
42  }
43
44  /**
45   * @brief SampleClass::to_string converts a SampleClass to a std::string representation.
46   * @return a pretty(-ish) std::string representation of the object.
47   */
48  std::string SampleClass::to_string() const
49  {
50      std::stringstream ss;
51      ss << "Sample_Class:: " << std::endl <<
52            "a_private_member_ = "        << std::to_string(a_private_member_) <<␣
    ↪std::endl <<
53            "a_private_qt_string_ = "     << a_private_qt_string_.toStdString() <<␣
    ↪std::endl <<
54            "a_private_eigen3_matrix_ = " << a_private_eigen3_matrix_ << std::endl;
55      return ss.str();
56  }
57
58  /**
59   * @brief operator << the stream operator for SampleClass objects.
60   * @param [in/out] the std::ostream to be modified.
61   * @param [in] sc the SampleClass whose representation is to be streamed.
62   * @return the modified os with the added SampleClass string representation.
63   * @see SampleClass::to_string().
```

(continues on next page)

(continued from previous page)

```
 */
std::ostream &operator<<(std::ostream &os, const SampleClass &sc)
{
    return os << sc.to_string();
}
```

## 53.3 Sources for a local node that uses the library

**In this step, we'll work on these.**

```
cpp_package_with_a_library
├── CMakeLists.txt
├── include
│   └── cpp_package_with_a_library
│       └── sample_class.hpp
├── package.xml
└── src
    ├── sample_class.cpp
    ├── sample_class_local_node.cpp
    ├── sample_class_local_node.hpp
    └── sample_class_local_node_main.cpp
```

Just in case you need to have a node, in the same package, that also uses the library exported by this package. Nothing too far from what we have already done.

### sample_class_local_node.cpp

`sample_class.cpp`

```
#include "sample_class_local_node.hpp"

/**
 * @brief SampleClassLocalNode::SampleClassLocalNode Default constructor.
 */
SampleClassLocalNode::SampleClassLocalNode():
    rclcpp::Node("sample_class_local_node"),
    timer_period_(0.5),
    print_count_(0)
{
    timer_ = create_wall_timer(
                std::chrono::milliseconds(long(timer_period_*1e3)),
                std::bind(&SampleClassLocalNode::_timer_callback, this)
                );
}

/**
 * @brief SampleClassLocalNode::_timer_callback periodically prints class info using␣
↪RCLCPP_INFO.
 */
```

(continues on next page)

(continued from previous page)

```
void SampleClassLocalNode::_timer_callback()
{
    RCLCPP_INFO_STREAM(get_logger(),
                       std::string("sum_of_squares = ") +
                       std::to_string(SampleClass::sum_of_squares(print_count_,print_
↪count_-5))
                       );

    RCLCPP_INFO_STREAM(get_logger(),
                       sample_class_.to_string() +
                       std::to_string(print_count_) +
                       std::string(" times.")
                       );
    print_count_++;
}
```

### sample_class_local_node.hpp

sample_class_local_node.cpp

```
#pragma once

#include <rclcpp/rclcpp.hpp>
#include <cpp_package_with_a_library/sample_class.hpp>

/**
 * @brief A ROS2 Node that uses the SampleClass within the same package.
 */
class SampleClassLocalNode: public rclcpp::Node
{
private:
    SampleClass sample_class_;

    double timer_period_;
    int print_count_;
    rclcpp::TimerBase::SharedPtr timer_;

    void _timer_callback();
public:
    SampleClassLocalNode();

};
```

### sample_class_local_node_main.cpp

sample_class.cpp

```
#include <rclcpp/rclcpp.hpp>

#include "sample_class_local_node.hpp"

int main(int argc, char** argv)
```

(continues on next page)

(continued from previous page)

```
{
    rclcpp::init(argc,argv);

    try
    {
        auto node = std::make_shared<SampleClassLocalNode>();

        rclcpp::spin(node);
    }
    catch (const std::exception& e)
    {
        std::cerr << std::string("::Exception::") << e.what();
    }

    return 0;
}
```

CHAPTER

# FIFTYFOUR

# #VENT DEMYSTIFYING C++

> **Warning**
>
> Anything below this point is just me venting about topics that frequently come up when C++ is mentioned.

## 54.1 But, C++ is difficult

I think C++ organically follows Bushnell's Law, adjusted for the topic

> All the best [programming languages] are easy to learn and difficult to master. They should reward the first quarter and the hundredth.

Beauty is in the eye of the beholder, but soon enough, if you're doing anything state-of-the-art, you'll hit performance bottlenecks with Python (and friends) that will naturally pull you towards C++.

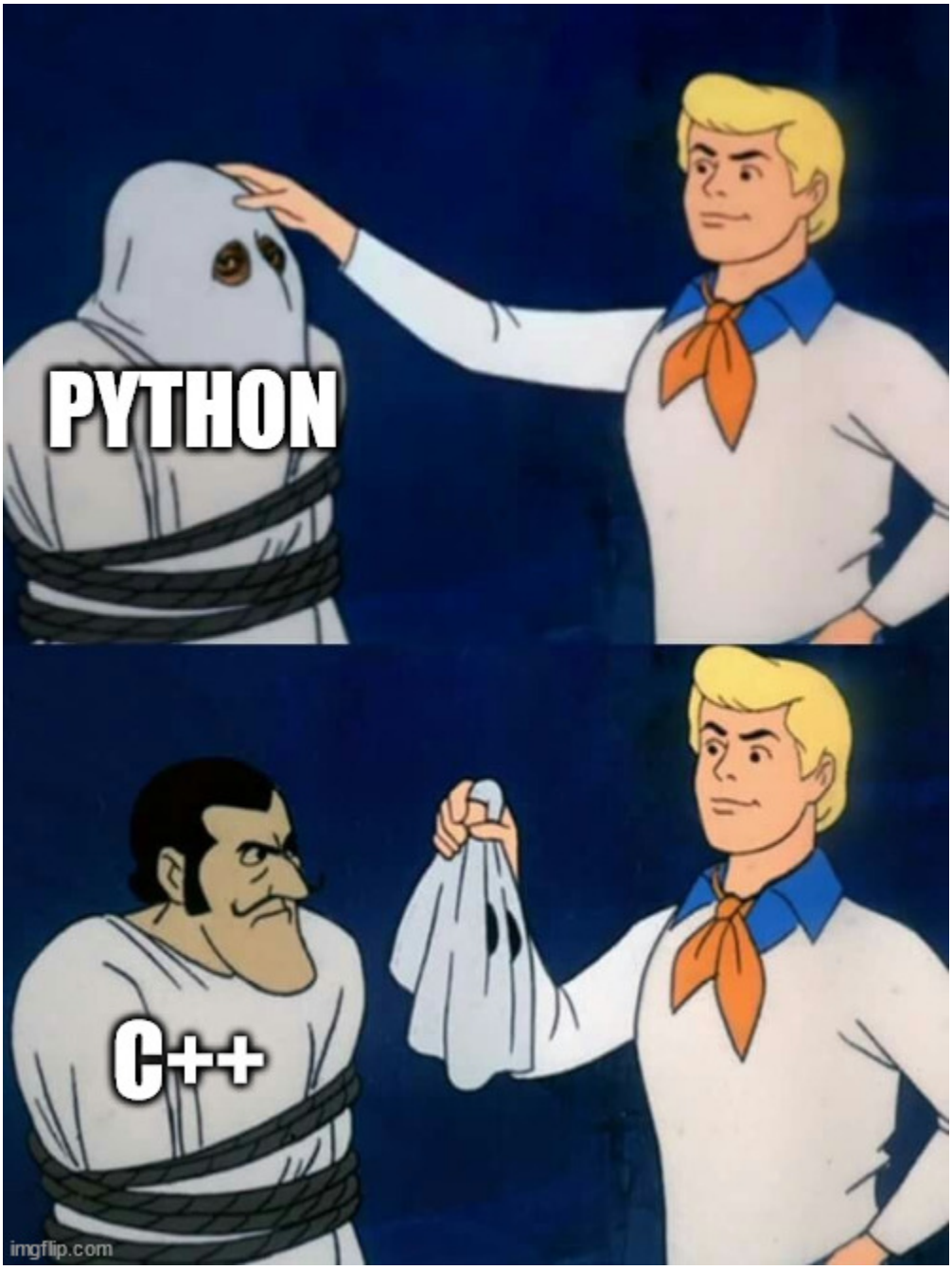
## 54.2 But with Python, we don't need C++

This makes me feel like breaking the news to someone that Santa isn't real, but just as an example, see numpy and PyTorch.

**Why is NumPy Fast?**
: [...] these things are taking place, of course, just "behind the scenes" in optimized, pre-compiled C code [...]

**Using the PyTorch C++ Frontend**
: [...] While the primary interface to PyTorch naturally is Python, this Python API sits atop a substantial C++ codebase providing foundational data structures and functionality such as tensors and automatic differentiation. [...]

The memefied version of this discussion is




## 54.3 Why use C++ if it sucks??

There's much folklore around C++. "C is faster than C++." "C++ is unsafe" (I'm looking at you, Rust). Anyhow, we'd all benefit if people stopped spreading weird fallacies about the C++ language when the problems they have can usually be attributed instead to a skill issue. Some quick info from Stroustrup's FAQ, also known as the person who designed and implemented the C++ programming language.

`<begin Stroustrup's FAQ quote>`

**What is the difference between C and C++?**

C++ is a direct descendant of C that retains almost all of C as a subset. C++ provides stronger type checking

than C and directly supports a wider range of programming styles than C. C++ is "a better C" in the sense that it supports the styles of programming done using C with better type checking and more notational support (without loss of efficiency). In the same sense, ANSI C is a better C than K&R C. In addition, C++ supports data abstraction, object-oriented programming, and generic programming (see my books). I have never seen a program that could be expressed better in C than in C++ (and I don't think such a program could exist - every construct in C has an obvious C++ equivalent). [...]

**C++ is low-level?**
No. C++ offers both low-level and high-level features. C++ has low-level parts, such as pointers, arrays, and casts. These facilities are (almost identical to what C offers) are essential (in some form or other) for close-to-the-hardware work. So, if you want low-level language facilities, yes C++ provides a well-tried set of facilities for you. However, when you don't want to use low-level features, you don't need to use the C++ facilities (directly). Instead, you can rely on higher-level facilities, including libraries. For example, if you don't want to use arrays and pointers, standard library strings and containers are (better) alternatives in many cases. If you use only low-level facilities, you are almost certainly wasting time and complicating maintenance without performance advantages (see Learning Standard C++ as a New Language). You may also be laying your systems open to attacks (e.g. buffer overflows).

**C++ too slow for low-level work?**
No. If you can afford to use C, you can afford to use C++, even the higher-level facilities of C++ where you need their functionality. See Abstraction and the C++ machine model and the ISO C++ standards committee's Technical Report on Performance.

**C++ is useful only if you write truly object-oriented code?**
No. That is, "no" for just about any reasonable definition of "object-oriented". C++ provides support for a wide variety of needs, not just for one style or for one kind of application. In fact, compared to C, C++ provides more support for very simple programming tasks. For example, the standard library and other libraries radically simplifies many otherwise tedious and error-prone tasks. C++ is widely used for huge applications but it also provides benefits for even tiny programming tasks.

`<end Stroustrup's FAQ quote>`

## 54.4 But I hate pointers, and pointers hate me: The ballad of `segmentation fault (core dumped)`

In things entirely written in modern C++ (loosely C++11 and above, but C++14 and above for what I want to say here), you shouldn't see any `new` or any loose raw pointer modifiers `*`.

Use smart pointers. In general, std::shared_ptr and, if needed, std::unique_ptr.

If only using smart pointers, you still manage to get a segmentation fault, then hats off to you.

## 54.5 But I can get segfaults with `std::vector`

As a successor of C, the standard library in C++ kept some of its predecessor's behavior of not generating exceptions.

For example, with trigonometric functions in C++, the error handling is C-like

For instance, getting the `acos` of 1.1, which is invalid, will fail silently in C++. We must check if the output is `NaN`, e.g., with

```
#include <cmath>
#include <iostream>

int main()
{
```

(continues on next page)

(continued from previous page)

```
  auto a = std::acos(1.1);
  std::cout << std::isnan(a) ? "the output was invalid but no exception was thrown " : a
↪<< std::endl;
}
```

The same applies if we try to access beyond a vector's limits with the good and old `operator[]`. Instead of doing that, use the method `.at()`, which checks the bounds.

```
#include <iostream>
#include <vector>
#include <exception>

int main()
{
  auto v = {1.0,2.0,3.0,4.0};
  try
  {
    std::cout << v.at(22) << std::endl;
  }
  catch (const std::out_of_range& e)
  {
    std::cout << e.what() << std::endl;
  }
}
```

As a conclusion, find the correct function/method or throw an exception yourself.

## 54.6 But C++ makes too many copies of objects: The sonata of "I don't know perfect forwarding"

I see this claim all the time, and it has many skill-issue-related causes, but basically, it shows up more frequently in the constructors of `std::vector` and `std::shared_ptr`.

Let's suppose that we have a class

```
class Potato{
  private:
    double size_;
  public:
    Potato(const double& size):
    size_(size)
    {};
};
```

for which we want to get a `std::shared_ptr`. Do not do this

```
auto potato_ptr = std::make_shared<Potato>(Potato(20.0));
```

> **Warning**
>
> This is not the only issue you can have by doing this. It can generate all sorts of issues, in particular with classes

that are not copyable.

because that will create one instance of `Potato(20.0)`, just to copy it when creating the `std::shared_ptr`. Do this, instead

```
auto potato_ptr = std::make_shared<Potato>(20.0);
```

by forwarding the argument to the constructor instead of calling it explicitly.

For everything else that you don't want to copy, use `std::move()`, but you don't see it that much unless you're designing a library.

CHAPTER

FIFTYFIVE

# THE SAS TUTORIALS MURILO PROMISED BUT NEVER WRITES

## 55.1 Pre-requisites

The reader is highly recommended to go through the earlier parts of the tutorial.

## 55.2 Workshops

- *SAS* Workshop 1 (Nov 29th, 2024). `pdf`
- *SAS* Workshop 2 (April 2nd, 2025). `pdf`

## 55.3 Quick overview

1. ***Installation***
   Setting up your system to use *SAS*.
2. ***Creating a new SASRobotDriver***
   Creating a new subclass of *SASRobotDriver* and use it in *SAS*. First shown in *SAS* Workshop 2.

CHAPTER

# FIFTYSIX

# OVERVIEW

**Warning**

These instructions are for Ubuntu 24.04 and ROS2 Jazzy. You might be able to make this run in other settings, but I am currently unable to provide support for those.

**Note**

Information is centralised in https://smartarmstack.github.io.

## 56.1 Docker image

**See also**

https://smartarmstack.github.io/#docker

A docker image with all *sas* software is available as follows.

```
docker run -it murilomarinho/sas:latest
```

## 56.2 Installing on a given system

**See also**

https://smartarmstack.github.io/#installation

### 56.2.1 Setting up PPA

```
curl -s --compressed "https://smartarmstack.github.io/smart_arm_stack_ROS2/KEY.gpg" \
| gpg --dearmor \
| sudo tee /etc/apt/trusted.gpg.d/smartarmstack_lgpl.gpg >/dev/null
sudo curl -s --compressed -o /etc/apt/sources.list.d/smartarmstack_lgpl.list \
"https://smartarmstack.github.io/smart_arm_stack_ROS2/smartarmstack_lgpl.list"
```

(continues on next page)

(continued from previous page)

```
sudo apt update
sudo apt-get install ros-jazzy-sas-*
```

## 56.3 Create the tutorial folder

Below, we make the directory used throughout the tutorial

```
mkdir -p ~/sas_tutorial_workspace/src
```

## 56.4 Developer environment

It is recommended to use the latest versions of

1. PyCharm Community
2. QtCreator

CHAPTER

# FIFTYSEVEN

# CREATING A NEW `sas_robot_driver` PACKAGE

Fortunately, `sas` already has a good number of drivers for popular robotic manipulators. Nonetheless, it is common to need the integration with new robotic systems. This tutorial will assist you in creating a suitable package.

You might be wondering why go through the trouble of doing this. Simply put, creating a suitable `sas` package that has a subclass of `sas::RobotDriver` will allow you to

1. Expose joint states and robot control inputs in ROS2 without programming a single subscriber, publisher, or service.
2. Access a C++ driver via Python without any particular Python code for the new robot.
3. Integrate with all other packages of `sas`, such as the teleoperation packages.

It's not too late to turn back now. Once you taste the forbidden `sas` fruit you will not want to go back to writing ROS2 publishers and subscribers by yourself.

## 57.1 Creating the package

The first step is to create the package with the correct dependencies. In this example we depend on **`sas_core`**, **`sas_common`**, and **`sas_robot_driver`**. Using **`ros2 pkg create`** we do

```
cd ~/sas_tutorial_workspace/src
ros2 pkg create sas_robot_driver_myrobot \
--build-type ament_cmake \
--dependencies rclcpp sas_core sas_common sas_robot_driver
```

which outputs

**ros2 pkg create output**

```
going to create a new package
package name: sas_robot_driver_myrobot
destination directory: /home/murilo/Downloads/pycharm-community-2024.3.5/bin
package format: 3
version: 0.0.0
description: TODO: Package description
maintainer: ['murilo <murilo.marinho@manchester.ac.uk>']
licenses: ['TODO: License declaration']
build type: ament_cmake
dependencies: ['rclcpp', 'sas_core', 'sas_common', 'sas_robot_driver']
creating folder ./sas_robot_driver_myrobot
creating ./sas_robot_driver_myrobot/package.xml
```

(continues on next page)

(continued from previous page)

```
creating source and include folder
creating folder ./sas_robot_driver_myrobot/src
creating folder ./sas_robot_driver_myrobot/include/sas_robot_driver_myrobot
creating ./sas_robot_driver_myrobot/CMakeLists.txt

[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
BSL-1.0
BSD-2.0
BSD-2-Clause
BSD-3-Clause
GPL-3.0-only
LGPL-3.0-only
MIT
MIT-0
```

## 57.2 Package-related sources

**In this step, we'll work on these.**

```
└── sas_robot_driver_myrobot
    ├── CMakeLists.txt
    ├── include
    │   └── sas_robot_driver_myrobot
    │       └── sas_robot_driver_myrobot.hpp
    ├── launch
    │   └── real_robot_launch.py
    ├── package.xml
    ├── scripts
    │   └── joint_interface_example.py
    └── src
        ├── sas_robot_driver_myrobot.cpp
        └── sas_robot_driver_myrobot_node.cpp
```

The files already exist, we just need to modify them as follows

### package.xml

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>sas_robot_driver_myrobot</name>
  <version>0.0.0</version>
  <description>TODO: Package description</description>
```

(continues on next page)

(continued from previous page)

```
  <maintainer email="murilo.marinho@manchester.ac.uk">murilo</maintainer>
  <license>TODO: License declaration</license>

  <buildtool_depend>ament_cmake</buildtool_depend>

  <depend>rclcpp</depend>
  <depend>sas_core</depend>
  <depend>sas_common</depend>
  <depend>sas_robot_driver</depend>

  <test_depend>ament_lint_auto</test_depend>
  <test_depend>ament_lint_common</test_depend>

  <export>
    <build_type>ament_cmake</build_type>
  </export>
</package>
```

### CMakeLists.txt

CMakeLists.txt

```
cmake_minimum_required(VERSION 3.8)
project(sas_robot_driver_myrobot)

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

# find dependencies
find_package(ament_cmake REQUIRED)
find_package(Eigen3 REQUIRED)
find_package(rclcpp REQUIRED)
find_package(sas_core REQUIRED)
find_package(sas_common REQUIRED)
find_package(sas_robot_driver REQUIRED)

###################################
# CPP Shared Library Block [BEGIN] #
# vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv #
# https://ros2-tutorial.readthedocs.io/en/latest/
# The most common use case is to merge everything you need to export
# into the same shared library called ${PROJECT_NAME}.
add_library(${PROJECT_NAME} SHARED
    src/sas_robot_driver_myrobot.cpp
)

ament_target_dependencies(${PROJECT_NAME}
    sas_core
    sas_common
    sas_robot_driver

```

(continues on next page)

(continued from previous page)

```
    )

target_include_directories(${PROJECT_NAME}
    PUBLIC
    $<INSTALL_INTERFACE:include>)

target_include_directories(${PROJECT_NAME}
    PRIVATE
    $<BUILD_INTERFACE:${CMAKE_CURRENT_SOURCE_DIR}/include>
    )

ament_export_targets(export_${PROJECT_NAME} HAS_LIBRARY_TARGET)
ament_export_dependencies(
    sas_core
    sas_common
    sas_robot_driver

    )

install(
    DIRECTORY include/
    DESTINATION include

    )

install(
    TARGETS ${PROJECT_NAME}
    EXPORT export_${PROJECT_NAME}
    LIBRARY DESTINATION lib
    ARCHIVE DESTINATION lib
    RUNTIME DESTINATION bin
    INCLUDES DESTINATION include

    )
# ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^ #
# CPP Shared Library Block [END] #
##################################

############################
# CPP Binary Block [BEGIN] #
# vvvvvvvvvvvvvvvvvvvvvvvv #
# https://ros2-tutorial.readthedocs.io/en/latest/
# While we cant use blocks https://cmake.org/cmake/help/latest/command/block.html
 ↪#command:block
# we use set--unset
set(RCLCPP_LOCAL_BINARY_NAME sas_robot_driver_myrobot_node)

add_executable(${RCLCPP_LOCAL_BINARY_NAME}
    src/sas_robot_driver_myrobot_node.cpp

    )

```

(continues on next page)

(continued from previous page)

```
ament_target_dependencies(${RCLCPP_LOCAL_BINARY_NAME}
    rclcpp
    sas_common
    sas_core
    sas_robot_driver

)

target_link_libraries(${RCLCPP_LOCAL_BINARY_NAME}
    ${PROJECT_NAME}

    )

target_include_directories(${RCLCPP_LOCAL_BINARY_NAME} PUBLIC
    $<BUILD_INTERFACE:${CMAKE_CURRENT_SOURCE_DIR}/include>
    $<INSTALL_INTERFACE:include>)

target_compile_features(${RCLCPP_LOCAL_BINARY_NAME} PUBLIC c_std_99 cxx_std_17)

install(TARGETS ${RCLCPP_LOCAL_BINARY_NAME}
    DESTINATION lib/${PROJECT_NAME})

unset(RCLCPP_LOCAL_BINARY_NAME)
# ^^^^^^^^^^^^^^^^^^^^^^ #
# CPP Binary Block [END] #
##########################

########################
# Launch Block [BEGIN] #
# vvvvvvvvvvvvvvvvvvv #
# According to https://github.com/SmartArmStack/sas_robot_driver/blob/ros2/CMakeLists.txt
install(DIRECTORY
  launch
  DESTINATION share/${PROJECT_NAME}/
)
# ^^^^^^^^^^^^^^^^^^ #
# Launch Block [END] #
######################

##########################
# Scripts Block [BEGIN] #
# vvvvvvvvvvvvvvvvvvvvvv #
# https://ros2-tutorial.readthedocs.io/en/latest/
install(DIRECTORY
  scripts/
  FILE_PERMISSIONS OWNER_EXECUTE OWNER_WRITE OWNER_READ
  DESTINATION lib/${PROJECT_NAME}
)
# ^^^^^^^^^^^^^^^^^^^^ #
# Scripts Block [END] #
########################

```

(continues on next page)

(continued from previous page)

```
if(BUILD_TESTING)
  find_package(ament_lint_auto REQUIRED)
  # the following line skips the linter which checks for copyrights
  # comment the line when a copyright and license is added to all source files
  set(ament_cmake_copyright_FOUND TRUE)
  # the following line skips cpplint (only works in a git repo)
  # comment the line when this package is in a git repo and when
  # a copyright and license is added to all source files
  set(ament_cmake_cpplint_FOUND TRUE)
  ament_lint_auto_find_test_dependencies()
endif()

ament_package()
```

In `CMakeLists.txt` we have a sequence of four blocks. These are all directly related to ROS2 and although in this tutorial I define a best practice, this is not particular to `sas`. My advice is to always rely on these blocks because `CMakeLists.txt` can quickly become impossible to maintain if it is not organized.

The first block, below, is made to create a C++ library that will contain all the necessary driver information and our new `sas::RobotDriver` subclass. Doing so allows this project to have organized access to this library and exposes it to other packages. We don't need direct access to this class in other `sas` packages, but it is important to have this freedom.

```
####################################
# CPP Shared Library Block [BEGIN] #
# vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv #
# https://ros2-tutorial.readthedocs.io/en/latest/
# The most common use case is to merge everything you need to export
# into the same shared library called ${PROJECT_NAME}.
add_library(${PROJECT_NAME} SHARED
    src/sas_robot_driver_myrobot.cpp
)

ament_target_dependencies(${PROJECT_NAME}
    sas_core
    sas_common
    sas_robot_driver

    )

target_include_directories(${PROJECT_NAME}
    PUBLIC
    $<INSTALL_INTERFACE:include>)

target_include_directories(${PROJECT_NAME}
    PRIVATE
    $<BUILD_INTERFACE:${CMAKE_CURRENT_SOURCE_DIR}/include>
    )

ament_export_targets(export_${PROJECT_NAME} HAS_LIBRARY_TARGET)
ament_export_dependencies(
    sas_core
    sas_common
```

(continues on next page)

(continued from previous page)

```
    sas_robot_driver

    )

install(
    DIRECTORY include/
    DESTINATION include

    )

install(
    TARGETS ${PROJECT_NAME}
    EXPORT export_${PROJECT_NAME}
    LIBRARY DESTINATION lib
    ARCHIVE DESTINATION lib
    RUNTIME DESTINATION bin
    INCLUDES DESTINATION include

    )
# ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^ #
# CPP Shared Library Block [END] #
##################################
```

The second block, below, is made to compile the binary `sas_robot_driver_myrobot_node` which is the ROS2 node that manages the driver for us in ROS2. We will create this file in the following sections.

```
############################
# CPP Binary Block [BEGIN] #
# vvvvvvvvvvvvvvvvvvvvvvvv #
# https://ros2-tutorial.readthedocs.io/en/latest/
# While we cant use blocks https://cmake.org/cmake/help/latest/command/block.html
↪#command:block
# we use set--unset
set(RCLCPP_LOCAL_BINARY_NAME sas_robot_driver_myrobot_node)

add_executable(${RCLCPP_LOCAL_BINARY_NAME}
    src/sas_robot_driver_myrobot_node.cpp

    )

ament_target_dependencies(${RCLCPP_LOCAL_BINARY_NAME}
    rclcpp
    sas_common
    sas_core
    sas_robot_driver

)

target_link_libraries(${RCLCPP_LOCAL_BINARY_NAME}
    ${PROJECT_NAME}

    )
```

(continues on next page)

(continued from previous page)

```
target_include_directories(${RCLCPP_LOCAL_BINARY_NAME} PUBLIC
    $<BUILD_INTERFACE:${CMAKE_CURRENT_SOURCE_DIR}/include>
    $<INSTALL_INTERFACE:include>)

target_compile_features(${RCLCPP_LOCAL_BINARY_NAME} PUBLIC c_std_99 cxx_std_17)

install(TARGETS ${RCLCPP_LOCAL_BINARY_NAME}
    DESTINATION lib/${PROJECT_NAME})

unset(RCLCPP_LOCAL_BINARY_NAME)
# ^^^^^^^^^^^^^^^^^^^^^^ #
# CPP Binary Block [END] #
##########################
```

The third block, below, is meant to install any launch files that we add to the folder `launch`. Remember that if these files are not installed we won't be able to call them with **`ros2 launch`**.

```
########################
# Launch Block [BEGIN] #
# vvvvvvvvvvvvvvvvvvvv #
# According to https://github.com/SmartArmStack/sas_robot_driver/blob/ros2/CMakeLists.txt
install(DIRECTORY
  launch
  DESTINATION share/${PROJECT_NAME}/
)
# ^^^^^^^^^^^^^^^^^^ #
# Launch Block [END] #
######################
```

Lastly, the fourth block, below, is meant to install any Python files in the folder `scripts`. Notice that we need to change the permissions for the files to be executable otherwise we won't be able to find them with **`ros2 run`**.

```
#########################
# Scripts Block [BEGIN] #
# vvvvvvvvvvvvvvvvvvvvv #
# https://ros2-tutorial.readthedocs.io/en/latest/
install(DIRECTORY
  scripts/
  FILE_PERMISSIONS OWNER_EXECUTE OWNER_WRITE OWNER_READ
  DESTINATION lib/${PROJECT_NAME}
)
# ^^^^^^^^^^^^^^^^^^^^^ #
# Scripts Block [END] #
#######################
```

## 57.3 TL;DR

**(Murilo's) `sas_robot_driver` best practices**

For each new robot called `myrobot` we have the three steps below as a must

1. `sas_robot_driver_myrobot.hpp` with the driver's class definition that inherits from `sas_robot_driver`. This file must not include any internal driver or library files because it will be exported.
2. `sas_robot_driver_myrobot.cpp` with the driver's class implementation. Any internal libraries or drivers must be included here so that they are not externally visible.
3. `sas_robot_driver_myrobot_node.cpp` that configures the driver and calls the ROS2 loop.

The creation of the following two is trivial

1. `real_robot_launch.py` a suitable launch file to properly configure `sas_robot_driver_myrobot_node.cpp`.
2. `joint_interface_example.py` a Python script to control the C++ node (if needed).

## 57.4 Creating the ROS2 package

Let's create all the files used in the remainder of this tutorial.

```
cd ~/sas_tutorial_workspace/src/sas_robot_driver_myrobot
mkdir -p src
touch src/sas_robot_driver_myrobot.cpp
touch src/sas_robot_driver_myrobot_node.cpp
mkdir -p include/sas_robot_driver_myrobot
touch include/sas_robot_driver_myrobot/sas_robot_driver_myrobot.hpp
mkdir -p launch
touch launch/real_robot_launch.py
mkdir -p scripts
touch scripts/joint_interface_example.py
```

## 57.5 The subclass of `sas::RobotDriver`

**In this step, we'll work on these.**

```
└── sas_robot_driver_myrobot
    ├── CMakeLists.txt
    ├── include
    │   └── sas_robot_driver_myrobot
    │       └── sas_robot_driver_myrobot.hpp
    ├── launch
    │   └── real_robot_launch.py
    ├── package.xml
    ├── scripts
    │   └── joint_interface_example.py
    └── src
        ├── sas_robot_driver_myrobot.cpp
        └── sas_robot_driver_myrobot_node.cpp
```

The files in question are as follows.

### sas_robot_driver_myrobot.hpp

sas_robot_driver_myrobot.hpp

```
#include <atomic>
#include <memory>
#include <sas_core/sas_robot_driver.hpp>

using namespace Eigen;

namespace sas
{

struct RobotDriverMyrobotConfiguration
{
    std::string ip;
    std::tuple<VectorXd,VectorXd> joint_limits;
};

class RobotDriverMyrobot: public RobotDriver
{
private:
    RobotDriverMyrobotConfiguration configuration_;

    //Use the Impl idiom to "hide" internal driver sources
    class Impl;
    std::unique_ptr<Impl> impl_;
public:

    // Prevent copies as usually drivers have threads
    RobotDriverMyrobot(const RobotDriverMyrobot&)=delete;
    RobotDriverMyrobot()=delete;
    ~RobotDriverMyrobot();

    // This boilderplate constructor usually does the job well and prevent big changes␣
↪when
    // parameters change
    RobotDriverMyrobot(const RobotDriverMyrobotConfiguration &configuration, std::atomic_
↪bool* break_loops);

    /// Everything below this line is an override
    /// the concrete implementations are needed
    VectorXd get_joint_positions() override;
    void set_target_joint_positions(const VectorXd& desired_joint_positions_rad)␣
↪override;

    void connect() override;
    void disconnect() override;

    void initialize() override;
    void deinitialize() override;

```

(continues on next page)

(continued from previous page)

```
};
}
```

### sas_robot_driver_myrobot.cpp

sas_robot_driver_myrobot.cpp

```
#include "sas_robot_driver_myrobot/sas_robot_driver_myrobot.hpp"
#include <iostream>
#include <memory>
#include <sas_core/eigen3_std_conversions.hpp>

namespace sas
{


class RobotDriverMyrobot::Impl
{

public:
    bool connected{false};
    bool motor_on{false};

    VectorXd joint_positions_;


    Impl()
        {

        }

};

RobotDriverMyrobot::RobotDriverMyrobot(const RobotDriverMyrobotConfiguration&
↪configuration, std::atomic_bool* break_loops):
    RobotDriver(break_loops),
    configuration_(configuration)
{
    joint_limits_ = configuration.joint_limits; //for the superclass
    impl_ = std::make_unique<RobotDriverMyrobot::Impl>();
}

RobotDriverMyrobot::~RobotDriverMyrobot()
{

}

/**
 * @brief RobotDriverMyrobot::get_joint_positions
 * This method should always throw an exception if the user
 * tries to obtain the joint positions in an invalid state.
 *
```

(continues on next page)

(continued from previous page)

```
 * One useful way of defining that is with a VectorXd(), which
 * has by default size zero until it is initialized.
 *
 * @return a VectorXd representing the configuration space in radians.
 */
VectorXd RobotDriverMyrobot::get_joint_positions()
{
    if(impl_->joint_positions_.size()==0)
        throw std::runtime_error("Tried to obtain invalid joint positions");

    return impl_->joint_positions_;
}

/**
 * @brief RobotDriverMyrobot::set_target_joint_positions
 * This method expects the desired joint positions in radians. The most basic
 * check is for the correct
 *
 * @param desired_joint_positions_rad
 */
void RobotDriverMyrobot::set_target_joint_positions(const VectorXd &desired_joint_
↪positions_rad)
{
    if(desired_joint_positions_rad.size() != 6)
        throw std::runtime_error("Incorrect vector size in RobotDriverMyrobot::set_
↪target_joint_positions");

    impl_->joint_positions_ = desired_joint_positions_rad;
}

/**
 * @brief RobotDriverMyrobot::connect
 *
 * Usually this method will connect to a given ip address. It is also common
 * for this part of the code to stop running programs or turn the robot off.
 * This function is expected to throw an exception of something goes wrong.
 * For instance, if the connection is not established an exception MUST
 * be thrown.
 */
void RobotDriverMyrobot::connect()
{
    //An example of exception to throw.
    if(impl_->connected)
        throw std::runtime_error("Already connected.");

    impl_->connected = true;

    impl_->motor_on = false;

    //Usually after the connection is established we can read joint positions
    //but not all drivers work like this
    impl_->joint_positions_ = (VectorXd(6) << 0, 0, 0, 0, 0, 0).finished();
```

(continues on next page)

(continued from previous page)

```
}

/**
 * @brief RobotDriverMyrobot::initialize
 *
 * This method is expected to turn the robot on and initialize the internal joint control loop.
 * If there are any issues, this MUST throw an exception so that the program will finish cleanly.
 *
 * After this method finishes target joint states can be received.
 */
void RobotDriverMyrobot::initialize()
{
    impl_->motor_on = true;


}

/**
 * @brief RobotDriverMyrobot::deinitialize.
 * For safety reasons, this MUST NOT throw exceptions.
 */
void RobotDriverMyrobot::deinitialize()
{
    impl_->motor_on = false;

}

/**
 * @brief RobotDriverMyrobot::disconnect
 * For safety reasons, this MUST NOT throw exceptions.
 */
void RobotDriverMyrobot::disconnect()
{
    impl_->connected = false;
    impl_->joint_positions_ = VectorXd();
}

}
```

The example class file has three important design choices to note.

First, although self evident, we rely on subclass polymorphism to integrate new classes using the same code. To that end, our class inherits from `sas::RobotDriver`. You will notice that `sas::RobotDriver` is defined in the package `sas_core`. This is because the package `sas_core` holds every code that does not depend on ROS2. Eventually the idea is to make a standalone package for this part of `sas`.

```
#include <sas_core/sas_robot_driver.hpp>

using namespace Eigen;

namespace sas
```

(continues on next page)

(continued from previous page)

```
{

struct RobotDriverMyrobotConfiguration
{
    std::string ip;
    std::tuple<VectorXd,VectorXd> joint_limits;
};

class RobotDriverMyrobot: public RobotDriver
```

Second, we rely on the struct `RobotDriverMyrobotConfiguration` to simplify interaction with the constructor. This reduces the amount of code that needs to be changed if a parameter is added or removed.

```
struct RobotDriverMyrobotConfiguration
{
    std::string ip;
    std::tuple<VectorXd,VectorXd> joint_limits;
};
```

Third, we rely on the PIMPL idiom. This idiom is important to prevent driver internals to pollute the exported header. This is a very important step to guarantee that your users don't have to worry about source files specific to the robot and that your package is correctly self-contained. This is an important design aspect and should not be confused simply with aesthetics or my constant need to sound smart.

```
    //Use the Impl idiom to "hide" internal driver sources
    class Impl;
    std::unique_ptr<Impl> impl_;
```

When using the PIMPL idiom it is important not to forget that the definition of the implementation class is made in the source. In this example, it is simply a dummy, but in practice this will depend heavily on the robot drivers.

```
class RobotDriverMyrobot::Impl
{

public:
    bool connected{false};
    bool motor_on{false};

    VectorXd joint_positions_;


    Impl()
        {

        }

};
```

## 57.6 Writing the ROS2 Node

**In this step, we'll work on this.**

```
└── sas_robot_driver_myrobot
    ├── CMakeLists.txt
    ├── include
    │   └── sas_robot_driver_myrobot
    │       └── sas_robot_driver_myrobot.hpp
    ├── launch
    │   └── real_robot_launch.py
    ├── package.xml
    ├── scripts
    │   └── joint_interface_example.py
    └── src
        ├── sas_robot_driver_myrobot.cpp
        └── sas_robot_driver_myrobot_node.cpp
```

sas_robot_driver_myrobot_node.cpp

```cpp
#include <rclcpp/rclcpp.hpp>
#include <sas_common/sas_common.hpp>
#include <sas_core/eigen3_std_conversions.hpp>
#include <sas_robot_driver/sas_robot_driver_ros.hpp>
#include <sas_robot_driver_myrobot/sas_robot_driver_myrobot.hpp>
#include <dqrobotics/utils/DQ_Math.h>

/**********************************************
 * SIGNAL HANDLER
 * ********************************************/
#include<signal.h>
static std::atomic_bool kill_this_process(false);
void sig_int_handler(int)
{
    kill_this_process = true;
}

int main(int argc, char** argv)
{
    if(signal(SIGINT, sig_int_handler) == SIG_ERR)
    {
        throw std::runtime_error("::Error setting the signal int handler.");
    }

    rclcpp::init(argc,argv,rclcpp::InitOptions(),rclcpp::SignalHandlerOptions::None);

    auto node = std::make_shared<rclcpp::Node>("sas_robot_driver_myrobot");

    try
    {
        RCLCPP_INFO_STREAM_ONCE(node->get_logger(), "::Loading parameters from parameter␣
```

(continues on next page)

(continued from previous page)

```
↪server.");

        sas::RobotDriverMyrobotConfiguration configuration;

        sas::get_ros_parameter(node,"ip",configuration.ip);
        std::vector<double> joint_limits_min;
        std::vector<double> joint_limits_max;
        sas::get_ros_parameter(node,"joint_limits_min",joint_limits_min);
        sas::get_ros_parameter(node,"joint_limits_max",joint_limits_max);
        configuration.joint_limits = {deg2rad(sas::std_vector_double_to_vectorxd(joint_
↪limits_min)),
                                      deg2rad(sas::std_vector_double_to_vectorxd(joint_
↪limits_max))};

        sas::RobotDriverROSConfiguration robot_driver_ros_configuration;
        sas::get_ros_parameter(node,"thread_sampling_time_sec",robot_driver_ros_
↪configuration.thread_sampling_time_sec);
        robot_driver_ros_configuration.robot_driver_provider_prefix = node->get_name();

        RCLCPP_INFO_STREAM_ONCE(node->get_logger(), "::Parameters OK.");

        RCLCPP_INFO_STREAM_ONCE(node->get_logger(), "::Instantiating RobotDriverMyRobot.
↪");
        auto robot_driver_myrobot = std::make_shared<sas::RobotDriverMyrobot>
↪(configuration,
                                                                    &kill_this_process);

        RCLCPP_INFO_STREAM_ONCE(node->get_logger(), "::Instantiating RobotDriverROS.");
        sas::RobotDriverROS robot_driver_ros(node,
                                             robot_driver_myrobot,
                                             robot_driver_ros_configuration,
                                             &kill_this_process);
        robot_driver_ros.control_loop();

    }
    catch (const std::exception& e)
    {
        RCLCPP_ERROR_STREAM_ONCE(node->get_logger(), std::string("::Exception::") + e.
↪what());
    }

    return 0;
}
```

There are two notable steps for this integration.

First, we configure our newly created `RobotDriverMyrobotConfiguration` and the existing `sas::RobotDriverROSConfiguration` by obtaining parameters from ROS2. Using `sas::get_ros_parameter` to do that reduces the amount of code to write and allows Exception generation and handling.

```
        sas::RobotDriverMyrobotConfiguration configuration;

        sas::get_ros_parameter(node,"ip",configuration.ip);
```

(continues on next page)

(continued from previous page)

```
        std::vector<double> joint_limits_min;
        std::vector<double> joint_limits_max;
        sas::get_ros_parameter(node,"joint_limits_min",joint_limits_min);
        sas::get_ros_parameter(node,"joint_limits_max",joint_limits_max);
        configuration.joint_limits = {deg2rad(sas::std_vector_double_to_vectorxd(joint_
↪limits_min)),
                                      deg2rad(sas::std_vector_double_to_vectorxd(joint_
↪limits_max))};

        sas::RobotDriverROSConfiguration robot_driver_ros_configuration;
        sas::get_ros_parameter(node,"thread_sampling_time_sec",robot_driver_ros_
↪configuration.thread_sampling_time_sec);
        robot_driver_ros_configuration.robot_driver_provider_prefix = node->get_name();

        RCLCPP_INFO_STREAM_ONCE(node->get_logger(), "::Parameters OK.");
```

Second, we create an instance of `sas::RobotDriverROS`. This class will manage the creation of all ROS2 elements, such as publishers and subscribers, and loop through our `sas::RobotDriver` subclass. Notice that it has a smart pointer parameter of `sas::RobotDriver`, so we just need to add as argument any suitable pointer to a subclass of it.

```
        RCLCPP_INFO_STREAM_ONCE(node->get_logger(), "::Instantiating RobotDriverMyRobot.
↪");
        auto robot_driver_myrobot = std::make_shared<sas::RobotDriverMyrobot>
↪(configuration,
                                                                    &kill_this_process);

        RCLCPP_INFO_STREAM_ONCE(node->get_logger(), "::Instantiating RobotDriverROS.");
        sas::RobotDriverROS robot_driver_ros(node,
                                             robot_driver_myrobot,
                                             robot_driver_ros_configuration,
                                             &kill_this_process);
        robot_driver_ros.control_loop();
```

## 57.7 Contents of the launch file

**In this step, we'll work on this.**

```
└── sas_robot_driver_myrobot
    ├── CMakeLists.txt
    ├── include
    │   └── sas_robot_driver_myrobot
    │       └── sas_robot_driver_myrobot.hpp
    ├── launch
    │   └── real_robot_launch.py
    ├── package.xml
    ├── scripts
    │   └── joint_interface_example.py
    └── src
        ├── sas_robot_driver_myrobot.cpp
        └── sas_robot_driver_myrobot_node.cpp
```

The launch file will be like so.

real_robot_launch.py

```
from launch import LaunchDescription
from launch_ros.actions import Node


def generate_launch_description():

    return LaunchDescription([
        Node(
            output='screen',
            emulate_tty=True,
            package='sas_robot_driver_myrobot',
            executable='sas_robot_driver_myrobot_node',
            name='myrobot_1',
            parameters=[{
                "ip": "127.0.0.1",
                "joint_limits_min": [-360.0, -360.0, -360.0, -360.0, -360.0, -720.0],
                "joint_limits_max": [360.0, 360.0, 360.0, 360.0, 360.0, 720.0],
                "thread_sampling_time_sec": 0.002 # Robot thread is at 500 Hz
            }]
        ),

    ])
```

The parameters should be no surprise defined as follows. The only one that was not defined by our struct `RobotDriverMyrobotConfiguration`, namely `thread_sampling_time_sec`, is a parameter of `sas::RobotDriverROSConfiguration` that defines the sampling time of the ROS2 loop.

```
            parameters=[{
                "ip": "127.0.0.1",
                "joint_limits_min": [-360.0, -360.0, -360.0, -360.0, -360.0, -720.0],
                "joint_limits_max": [360.0, 360.0, 360.0, 360.0, 360.0, 720.0],
                "thread_sampling_time_sec": 0.002 # Robot thread is at 500 Hz
            }]
```

The most memorable aspect with respect to `sas` is that the `name` will define the topic prefixes. This is important to match other elements of `sas`.

```
            name='myrobot_1',
```

## 57.8 Running the launch file

Before running the launch file, remember to build and source

```
cd ~/sas_tutorial_workspace
colcon build
source install/setup.bash
```

```
ros2 launch sas_robot_driver_myrobot real_robot_launch.py
```

In another terminal

```
ros2 topic list
```

will show all the available topics that were created for you, freely. Notice that in none of the source files we created so far had any mention to topics or subscribers. All are created by **sas**.

```
/myrobot_1/get/home_states
/myrobot_1/get/joint_positions_max
/myrobot_1/get/joint_positions_min
/myrobot_1/get/joint_states
/myrobot_1/set/clear_positions
/myrobot_1/set/homing_signal
/myrobot_1/set/target_joint_forces
/myrobot_1/set/target_joint_positions
/myrobot_1/set/target_joint_velocities
/parameter_events
/rosout
```

## 57.9 Accessing through Python

**In this step, we'll work on this.**

```
└── sas_robot_driver_myrobot
    ├── CMakeLists.txt
    ├── include
    │   └── sas_robot_driver_myrobot
    │       └── sas_robot_driver_myrobot.hpp
    ├── launch
    │   └── real_robot_launch.py
    ├── package.xml
    ├── scripts
    │   └── joint_interface_example.py
    └── src
        ├── sas_robot_driver_myrobot.cpp
        └── sas_robot_driver_myrobot_node.cpp
```

joint_interface_example.py

```
#!/usr/bin/python3
import time

from math import sin, pi

import numpy
from dqrobotics import *  # Despite what PyCharm might say, this is very much necessary␣
↪or DQs will not be recognized
from dqrobotics.utils.DQ_Math import deg2rad

from sas_common import rclcpp_init, rclcpp_Node, rclcpp_spin_some, rclcpp_shutdown
from sas_robot_driver import RobotDriverClient

```

(continues on next page)

(continued from previous page)

```
from sas_core import Clock, Statistics


def main(args=None):
    try:
        rclcpp_init()
        node = rclcpp_Node("sas_robot_driver_myrobot_joint_space_example_node_cpp")

        # 10 ms clock
        clock = Clock(0.01)
        clock.init()

        # Initialize the RobotDriverClient
        rdi = RobotDriverClient(node, 'myrobot_1')

        # Wait for RobotDriverClient to be enabled
        while not rdi.is_enabled():
            rclcpp_spin_some(node)
            time.sleep(0.1)

        # Get topic information
        print(f"topic prefix = {rdi.get_topic_prefix()}")

        # Read the values sent by the RobotDriverServer
        joint_positions = rdi.get_joint_positions()
        print(f"joint positions = {joint_positions}")

        # For some iterations. Note that this can be stopped with CTRL+C.
        for i in range(0, 5000):
            clock.update_and_sleep()

            # Move the joints
            target_joint_positions = joint_positions + deg2rad([10.0 * sin(i / (50.0 *␣
↪pi))] * 6)
            # print(target_joint_positions)
            rdi.send_target_joint_positions(target_joint_positions)

            rclcpp_spin_some(node)

        # Statistics
        print("Statistics for the entire loop")
        print("  Mean computation time: {}".format(clock.get_statistics(
            Statistics.Mean, Clock.TimeType.Computational)
        ))
        print("  Mean idle time: {}".format(clock.get_statistics(
            Statistics.Mean, Clock.TimeType.Idle)
        ))
        print("  Mean effective thread sampling time: {}".format(clock.get_statistics(
            Statistics.Mean, Clock.TimeType.EffectiveSampling)
        ))

        rclcpp_shutdown()
```

(continues on next page)

(continued from previous page)

```
    except KeyboardInterrupt:
        print("Interrupted by user")
    except Exception as e:
        print("Unhandled excepts", e)


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

With the `launch` file running in one terminal, open another one and run

```
ros2 topic echo /myrobot_1/get/joint_states
```

Then, in yet another terminal, run the sample Python script. Before running the script, remember to build and source

```
cd ~/sas_tutorial_workspace
colcon build
source install/setup.bash
```

```
ros2 run sas_robot_driver_myrobot joint_interface_example.py
```

The output of `joint_interface_example.py` will be only descriptive.

### joint_interface_example.py output

```
********************************************************************************
sas::Clock (c) Murilo M. Marinho (murilomarinho.info) 2016-2023 LGPLv3
********************************************************************************
********************************************************************************************************
sas::RobotDriverClient (c) Murilo M. Marinho (murilomarinho.info) 2016-2023 LGPLv3
********************************************************************************************************
[INFO] [1743579389.703532605] [sas_robot_driver_myrobot_joint_space_example_node_cpp]:␣
↪::Initializing RobotDriverClient with prefix myrobot_1
topic prefix = myrobot_1
joint positions = [0. 0. 0. 0. 0. 0.]
```

While the magic happens in ROS2, therefore in the terminal in which we executed `ros2 topic echo`

### ros2 topic echo output

```
header:
  stamp:
    sec: 1743579392
    nanosec: 471990465
  frame_id: ''
name: []
position:
- 0.17141407117840998
- 0.17141407117840998
- 0.17141407117840998
- 0.17141407117840998
```

(continues on next page)

(continued from previous page)

```
- 0.17141407117840998
- 0.17141407117840998
velocity: []
effort: []
---
header:
  stamp:
    sec: 1743579392
    nanosec: 474015460
  frame_id: ''
name: []
position:
- 0.17141407117840998
- 0.17141407117840998
- 0.17141407117840998
- 0.17141407117840998
- 0.17141407117840998
- 0.17141407117840998
velocity: []
effort: []
---
header:
  stamp:
    sec: 1743579392
    nanosec: 475985122
  frame_id: ''
name: []
position:
- 0.17141407117840998
- 0.17141407117840998
- 0.17141407117840998
- 0.17141407117840998
- 0.17141407117840998
- 0.17141407117840998
velocity: []
effort: []
```

CHAPTER

# FIFTYEIGHT

# HAPPY DAYS WITH DOCKER

**Warning**

This topic is under construction and this might not even be its final form. Please feel free to open an issue if you spot any typos or other problems.

For a basic treatise see https://uomresearchit.github.io/docker-introduction/.

This tutorial is about the specifics of `ROS2` and `sas` when using docker on Ubuntu.

## 58.1 Installation

**Note**

Information obtained from the main documentation for Ubuntu. https://docs.docker.com/engine/install/ubuntu/

```
# Add Docker's official GPG key:
sudo apt-get update
sudo apt-get install ca-certificates curl
sudo install -m 0755 -d /etc/apt/keyrings
sudo curl -fsSL https://download.docker.com/linux/ubuntu/gpg -o /etc/apt/keyrings/docker.
↪asc
sudo chmod a+r /etc/apt/keyrings/docker.asc

# Add the repository to Apt sources:
echo \
  "deb [arch=$(dpkg --print-architecture) signed-by=/etc/apt/keyrings/docker.asc] https:/
↪/download.docker.com/linux/ubuntu \
  $(. /etc/os-release && echo "${UBUNTU_CODENAME:-$VERSION_CODENAME}") stable" | \
  sudo tee /etc/apt/sources.list.d/docker.list > /dev/null
sudo apt-get update

# Install the packages
sudo apt-get install docker-ce docker-ce-cli containerd.io docker-buildx-plugin docker-
↪compose-plugin
```

**Warning**

It's best not to call `sudo docker ...` to test it yet. If you're adding your user to the `docker` group, running `docker` as `sudo` once will change the access properties of the socket. It's best to test it after adding your user to the `docker` group.

## 58.2 Adding user to docker group

**Warning**

Adding a user to the `docker` group is a security concern. The users can easily share protected volumes and wipe out a whole shared computer. There does not seem to be a viable alternative currently so the system manager should be aware of this when adding users to the `docker` group.

Further, users' home folders should probably be encrypted to prevent private projects and data from being readable by unauthorised users.

You can add the user to `docker` group by downloading this helper script

```
mkdir -p ~/ros2_tutorial_workspace/docker
cd ~/ros2_tutorial_workspace/docker
curl -OL https://raw.githubusercontent.com/mmmarinho/ROS2_Tutorial/refs/heads/main/docs/
↪source/docker/scripts/set_docker_user.sh
chmod +x set_docker_user.sh
```

then

```
cd ~/ros2_tutorial_workspace/docker
./set_docker_user.sh $USER
```

**Important**

A reboot of the system is in general needed so that the group addition is propagated when you open a new terminal.

If you're curious, here are the main contents of the script.

```
USER_TO_ADD="$1"

# Check Ubuntu version
. /etc/lsb-release

if [ "$DISTRIB_CODENAME" = "resolute" ]; then
    echo "Ubuntu $DISTRIB_CODENAME detected, installing dependencies"
    sudo apt-get install util-linux-extra
fi

# Added the -f so that it does not fail if the group already exists
sudo groupadd -f docker
# Add the user in the argument
```

(continues on next page)

(continued from previous page)

```
sudo usermod -aG docker "${USER_TO_ADD}"
# Activate changes in the group
newgrp docker
# Check if it works without sudo
docker run hello-world
```

## 58.3 Basic testing

In a terminal window, do the following.

```
docker run hello-world
```

If executed for the first time in the machine, it should return something similar to the following. If the image already exists in the system, it will not be pulled.

```
Unable to find image 'hello-world:latest' locally
latest: Pulling from library/hello-world
58dee6a49ef1: Pull complete
c3bdf82c34d1: Download complete
Digest: sha256:0e760fdfbc48ba8041e7c6db999bb40bfca508b4be580ac75d32c4e29d202ce1
Status: Downloaded newer image for hello-world:latest

Hello from Docker!
This message shows that your installation appears to be working correctly.

To generate this message, Docker took the following steps:
 1. The Docker client contacted the Docker daemon.
 2. The Docker daemon pulled the "hello-world" image from the Docker Hub.
    (arm64v8)
 3. The Docker daemon created a new container from that image which runs the
    executable that produces the output you are currently reading.
 4. The Docker daemon streamed that output to the Docker client, which sent it
    to your terminal.

To try something more ambitious, you can run an Ubuntu container with:
 $ docker run -it ubuntu bash

Share images, automate workflows, and more with a free Docker ID:
 https://hub.docker.com/

For more examples and ideas, visit:
 https://docs.docker.com/get-started/
```

## 58.4 SAS testing

For the purposes of this illustration we will use the image `murilomarinho/sas:jazzy`.

### 58.4.1 Docker run interactively

Firstly it would be easier to tackle docker in simple commands before tackling complex scenarios.

We can start with the simple command below.

```
docker run -it --rm -e ROS_DOMAIN_ID=$ROS_DOMAIN_ID murilomarinho/sas:jazzy
```

The flags

1. `-it` will open an interactive shell and
2. `--rm` will remove all changes and return the image to its fresh initial state after we're done.
3. `-e` will pass environment variables to the container.

For this example, we will have `ROS_DOMAIN_ID=$ROS_DOMAIN_ID`. This means that the container will have the same `ROS_DOMAIN_ID` as the host. This is usually sufficient, but keep that in mind in case something is not communicating as expected.

The terminal in which you ran the `docker run` command should now be logged inside the container. The computer from which you ran is called the host. We will use this terminology to help explain where each command should be used.

Then, in the container and in the host we do as follows.

#### Container

```
ros2 run demo_nodes_cpp talker
```

#### Host

```
ros2 topic echo /chatter
```

**Tip**

You can close an interactive session on a container by typing on that terminal.

```
exit
```

Notice that host and container will communicate without any issues. Changing the network settings of the docker container may cause this to stop working, so make sure you know what you're doing.

### 58.4.2 Docker compose

If your host does not have **ROS2** you can also have multiple containers communicating with each other without any direct involvement of the host. For instance with the following compose file named `compose.yml` below.

`simple_example/compose.yml`

```
services:
  talker:
    image: murilomarinho/sas:jazzy
    stop_signal: SIGINT
    stop_grace_period: 10s
    environment:
```

(continues on next page)

(continued from previous page)

```
      - ROS_DOMAIN_ID
    command: /bin/bash -c "
      ros2 run demo_nodes_cpp talker
      "

  listener:
    image: murilomarinho/sas:jazzy
    stop_signal: SIGINT
    stop_grace_period: 10s
    environment:
      - ROS_DOMAIN_ID
    command: /bin/bash -c "
      ros2 topic echo /chatter
      "
```

In the folder where `compose.yml` exists, we do

```
docker compose pull
```

to make sure the image we have locally is indeed the latest.

**Warning**

If the `compose.yml` refers to a `Dockerfile` instead of an image, then `pull` will not have the intended effect. Instead, we do

```
docker compose build --pull
```

**Tip**

The `--pull` flag is useful most of the time. This will guarantee that any images that your compose file depends on will be pulled to their latest version. This can save a lot of time.
The `--no-cache` flag is useful when things changed in the image but did not trigger a re-build of that part of the cache. Most of the time you won't need to use it and instead it will cost you additional time.
You can combine them like so.

```
docker compose build --pull --no-cache
```

This will show the output of the two containers communicating over **ROS2**. Notice that there is no input from the host and no complicated network setup involved. Each container is called a service.

You can stop the process with `CTRL+C`. Note that we are using `stop_signal: SIGINT` in the docker compose file because otherwise it will send `SIGTERM`. In this toy example this is not an issue, but this is a major issue for nodes that control real resources. This can mean that a robot will not be safely disconnected before the container is destroyed leaving it in an undetermined state. For the same reason we have `stop_grace_period` to give the container enough time to close safely. Depending on your resource you'd prefer to have a larger value so that it is not escalated into a `SIGTERM` or `SIGKILL` before your container had enough time to sort out its shutdown procedure.

You will note that `SIGINT` sent this way might not kill one or another service. I haven't found too much discussion about the topic in the **ROS2** planet, but the gist of it is that we are not running **ROS2** commands directly in these examples. We are running a **bash** shell, which will configure itself with the **ROS2** workspace, then running `ros2 run` inside it. Therefore, the `SIGINT` is sent to **bash** first which will forward that signal to whatever it is running below. If the program ignores it, then it will be ignored.

Although in **sas** and in these tutorials there is much effort to guarantee nodes finish cleanly with a `SIGINT`, there will be many examples online that do not do this correctly.

**See also**

You might be interested in reading about the *wait and cooperative exit* implemented in **bash** to save yourself from the headaches of edge cases.

https://mywiki.wooledge.org/SignalTrap

## 58.5 Docker container in a realtime kernel

**See also**

Real-life example: https://github.com/MarinhoLab/sas_ur_control_template

Making things work on a realtime kernel is a relatively simple task.

1. Install the realtime kernel on the host.
2. Set up the container to take advantage of the realtime capabilities.

### 58.5.1 Install `PREEMPT_RT` on the host

**See also**

1. https://ubuntu.com/real-time
2. https://canonical-ubuntu-pro-client.readthedocs-hosted.com/en/latest/howtoguides/enable_realtime_kernel/

**Important**

From Ubuntu 26.04, the following command will suffice. Ubuntu pro is no longer needed.

```
sudo apt update && sudo apt install ubuntu-realtime
```

For ubuntu 24.04, we do the following.

```
sudo pro attach
sudo apt update && sudo apt install ubuntu-advantage-tools
sudo pro enable realtime-kernel
```

**Tip**

You can check if a process has properly been elevated to `SCHED_FIFO` with the following command.

```
ps -eLfc | grep FF
```

### 58.5.2 The `compose.yml`

For real-time performance, additional capabilities must be given to the container.

`realtime_example/compose.yml`

```
services:
  realtime:
    image: murilomarinho/sas:jazzy
    stop_signal: SIGINT
    stop_grace_period: 10s
    environment:
      - ROS_DOMAIN_ID
    cap_add:
      - SYS_NICE # PREEMPT_RT
    ulimits:
      rtprio: 99 # PREEMPT_RT
      rttime: -1 # PREEMPT_RT
    command: /bin/bash -c "
      ros2 run sas_core sas_clock_sched_fifo_example
      "
```

For this example, the relevant parameters are `cap_add`, `rtprio`, and `rttime`. The first one is to add the capability of setting process niceness. Then, the other two are related to the realtime priorities.

The compose file does not make anything realtime. For a realtime thread you will have to set up the thread scheduling properly to `SCHED_FIFO` or `SCHED_RR` We can run one example doing so in `sas_core` is shown below.

```
#include <iostream>

#include <sas_core/sas_clock.hpp>

int main(int, char**)
{
    // 1 ms
    sas::Clock clock(0.001);
    //Alternatively, use sas::Clock clock(0.01,false); to disable automatic statistics␣
↪calculation.
    //They do not take much time, but might affect whatever it is that we're trying to␣
↪time.

    //Set the communication thread to be realtime with SCHED_FIFO.
    sched_param sch;
    int policy;
    pthread_getschedparam(pthread_self(), &policy, &sch);
    sch.sched_priority = 20;
    if (pthread_setschedparam(pthread_self(), SCHED_FIFO, &sch)) {
        std::cout << "Failed to setschedparam: " << std::strerror(errno) << '\n';
    }

    // Always initialize right before the loop, to reduce slowdown in the object␣
↪allocation/initialization.
    clock.init();
    for(int i=0;i<50;i++)
    {
```

(continues on next page)

(continued from previous page)

```
        // Starting the loop with an update reduces overhead when entering the loop the␣
↪first time.
        clock.update_and_sleep();
    }

    //Statistics
    std::cout << "Statistics for the entire loop" << std::endl;
    std::cout << "  Mean computation time: " << clock.get_
↪statistics(sas::Statistics::Mean,sas::Clock::TimeType::Computational) << std::endl;
    std::cout << "  Mean idle time: " << clock.get_statistics(sas::Statistics::Mean,
↪sas::Clock::TimeType::Idle) << std::endl;
    std::cout << "  Mean effective thread sampling time: " << clock.get_
↪statistics(sas::Statistics::Mean,sas::Clock::TimeType::EffectiveSampling) << std::endl;
    std::cout << "  Overrun count: " << clock.get_overrun_count() << std::endl <<␣
↪std::endl;

    return 0;
}
```

With the `compose.yml` above, we do the following.

```
docker compose up
```

Which should output something similar to the following.

```
[+] up 1/1
 ✓ Container realtime_example-realtime-1 Recreated                                  ␣
↪                                                                                   ␣
↪                                                                                   ␣
↪                                          0.3s
Attaching to realtime-1
realtime-1  | ********************************************************************************
realtime-1  | sas::Clock (c) Murilo M. Marinho (murilomarinho.info) 2016-2026 LGPLv3
realtime-1  | ********************************************************************************
realtime-1  | Statistics for the entire loop
realtime-1  |   Mean computation time: 3.42e-07
realtime-1  |   Mean idle time: 0.000999767
realtime-1  |   Mean effective thread sampling time: 0.00100012
realtime-1  |   Overrun count: 0
realtime-1  |
realtime-1 exited with code 0
```

Note that if the correct capabilities are not available or if the correct kernel is not installed, even such a simple example can show clock overruns.

See below one run after removing `cap_add`, `rtprio`, and `rttime`.

```
[+] up 1/1
 ✓ Container realtime_example-realtime-1 Recreated                                  ␣
↪                                                                                   ␣
↪                                                                                   ␣
↪                                          0.1s
Attaching to realtime-1
```

(continues on next page)

(continued from previous page)

```
realtime-1  | ********************************************************************************
realtime-1  | sas::Clock (c) Murilo M. Marinho (murilomarinho.info) 2016-2026 LGPLv3
realtime-1  | ********************************************************************************
realtime-1  | Failed to setschedparam: Operation not permitted
realtime-1  | Statistics for the entire loop
realtime-1  |   Mean computation time: 7.02e-07
realtime-1  |   Mean idle time: 0.00106458
realtime-1  |   Mean effective thread sampling time: 0.00106529
realtime-1  |   Overrun count: 3
realtime-1  |
realtime-1 exited with code 0
```

## 58.6 Tips and troubleshooting

This documents part of my own misunderstandings when getting used to docker (which is an ongoing process) and other difficulties that are contributed by others.

### 58.6.1 Cleaning things up

Related information: https://docs.docker.com/engine/manage-resources/pruning/.

**Caution**

This will remove all containers, cache, and volumes. Check the instructions carefully before moving forward.

```
docker system prune --all --volumes
```

### 58.6.2 The standard shell is not interactive

When we start using **ROS2** we get used to rely on *~/.bashrc* to define environment variables and sometimes aliases.

Unless explicitly said with flags such as `-it`, docker does not run bash in interactive mode. This means that it will not execute anything in `~/.bashrc`. It does not matter what you add, because the first few lines of `~/.bashrc` check if the shell is interactive and return if it's not.

This example image that we use runs `source /etc/bash_env` in noninteractive shells. However, it will not run for interactive shells and aliases that we define will not work. Noninteractive shells by default do not expand aliases.

The "easiest" solution is

1. Set your `Dockerfile` to source `/etc/bash_env`
2. Add `source source /etc/bash_env` to your `~/.bashrc` exactly once.

### 58.6.3 Stop with the `--net=host` for everything

**See also**

1. https://robotics.stackexchange.com/questions/98161/ros2-foxy-nodes-cant-communicate-through-docker-container-border
2. https://github.com/rosblox/ros-template

3. https://github.com/eProsima/Fast-DDS/issues/1750

Although this command can help in some situations it is not recommended unless strictly necessary. It can for instance cause **ros2** to no longer be able to communicate between host and container.

The reason for this is that the current version of **ROS2** is based on `FASTRTPS`. When `--net=host` is defined, `FASTRTPS` tries to communicate via shared memory.

There are two ways of solving this issue.

1. Adjust the docker/linux settings via various mechanisms to expose the shared memory.
2. Adjusting the `FASTRTPS` profile so that the nodes communicate via `UDP`.

The main difficulty with the first solution is when the user calling the container is not `root`. There are many cases in which this won't be true, for instance for shared systems in which docker users are simply added to the `docker` group. In addition, calling `ros2` programs with `sudo` can cause many issues with permissions and further complicate the use of `ros2` in the host. There are smart solutions to aid in setting up the user properly. But I would only recommend going this route if the shared memory communication is paramount for performance which usually is not the case.

Alternatively, my suggested solution is to modify the container to force it to communicate over UDP. This has been well described in this issue.

This can be done by copying the contents below in the file `~/ros2_tutorial_workspace/docker/fastrtps_profile.xml`.

```
<?xml version="1.0" encoding="UTF-8" ?>
<profiles xmlns="http://www.eprosima.com/XMLSchemas/fastRTPS_Profiles" >
    <transport_descriptors>
        <transport_descriptor>
            <transport_id>CustomUdpTransport</transport_id>
            <type>UDPv4</type>
        </transport_descriptor>
    </transport_descriptors>

    <participant profile_name="participant_profile" is_default_profile="true">
        <rtps>
            <userTransports>
                <transport_id>CustomUdpTransport</transport_id>
            </userTransports>

            <useBuiltinTransports>false</useBuiltinTransports>
        </rtps>
    </participant>
</profiles>
```

Then, in the container, you can set before the nodes.

```
export FASTRTPS_DEFAULT_PROFILES_FILE="~/ros2_tutorial_workspace/docker/fastrtps_profile.
↪xml"
```

### 58.6.4 Sad notes on rootless `docker`

Although ideally having a rootless docker environment is what would be expected in a situation with shared computers and equipment the experience as of now adds a lot of complexity. In these notes I will focus on trying to reduce risks by

not adding more docker directives or requirements than those strictly need for things to work but with rootless docker there was a consistent difficulty with things such as.

1. ROS2 discovery. Settings become more difficult although can be circumvented.
2. Port exposure and external access. Some robotic systems need to open reverse sockets with the host and networking becomes more difficult that it should otherwise be in rootless docker.

CHAPTER

FIFTYNINE

# BASIC CMAKE TRICKS FOR C++ DEVELOPMENTS

## 59.1 Using this section

This section provides some tips for CMake and C++.

### 59.1.1 Install a CMake package without sudo privileges

To install a CMake package or library without sudo privileges, we need to define a directory to which we have access. For instance, in ~/.

#### Create a custom folder

In this tutorial, we are going to create a custom folder ~/opt containing the folders lib and include. This will be our directory to install all our CMake packages.

Run the following commands,

```
cd ~/
mkdir -p opt && cd opt
mkdir -p include
mkdir -p lib
```

Then, we update the LD_LIBRARY_PATH, LIBRARY_PATH, and CPATH in ~/.bashrc.

Do the following just once, so that all terminal windows automatically source this new workspace for you.

```
echo "# Update the environment variable LD_LIBRARY_PATH to include ~/opt/lib, as␣
↪instructed in https://ros2-tutorial.readthedocs.io" >> ~/.bashrc
echo "export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:~/opt/lib" >> ~/.bashrc

echo "# Update the environment variable LIBRARY_PATH to include ~/opt/lib, as instructed␣
↪in https://ros2-tutorial.readthedocs.io" >> ~/.bashrc
echo "export LIBRARY_PATH=$LIBRARY_PATH:~/opt/lib" >> ~/.bashrc

echo "# Update the environment variable CPATH to include ~/opt/include, as instructed in␣
↪https://ros2-tutorial.readthedocs.io" >> ~/.bashrc
echo "export CPATH=$CPATH:~/opt/include" >> ~/.bashrc

source ~/.bashrc
```

### Install a CMake package

To install a CMake package, we set the `CMAKE_INSTALL_PREFIX:PATH` flag with our custom folder (`/home/USERNAME/opt`)

```
cmake -DCMAKE_INSTALL_PREFIX:PATH=~/opt ..
make
make install
```

### Example: Installing qpOASES

This example shows how to build and install the qpOASES to be used in your CMake project.

**Note**

Check the official qpOASES documentation for more details.

**Warning**

This example assumes you have git, CMake, Eigen, and a C++ compiler installed in your GNU/Linux distribution.

To install qpOASES as a shared library, we use the instructions provided by the DQ Robotics in cpp-interface-qpoases specifying the installation directory.

```
cd ~/Downloads
git clone https://github.com/coin-or/qpOASES.git
cd qpOASES
sed -i -e 's/option(BUILD_SHARED_LIBS "If ON, build shared library instead of static"␣
↪OFF)/option(BUILD_SHARED_LIBS "If ON, build shared library instead of static" ON)/g'␣
↪CMakeLists.txt
mkdir build
cd build
cmake .. -DCMAKE_INSTALL_PREFIX:PATH=~/opt
make
make install
```

### Example: include and link the qpOASES in your project

#### CMakeLists.txt

`CMakeLists.txt`

```
cmake_minimum_required(VERSION 3.5)

project(test_proxsuite)

set(CMAKE_CXX_STANDARD 20)

FIND_PACKAGE(Eigen3 REQUIRED)
INCLUDE_DIRECTORIES(${EIGEN3_INCLUDE_DIR})
ADD_COMPILE_OPTIONS(-Werror=return-type
                    -Wall -Wextra -Wmissing-declarations
```

(continues on next page)

(continued from previous page)

```
                    -Wredundant-decls -Woverloaded-virtual)

add_executable(${PROJECT_NAME}
               ${PROJECT_NAME}.cpp)

target_link_libraries(${PROJECT_NAME}
            qpOASES
)
```

### test_qpoases.cpp

test_dqrobotics.cpp

```
/*
 * This is an official example provided by qpOASES
 * https://github.com/coin-or/qpOASES/blob/master/examples/example1.cpp
*/

#include <qpOASES.hpp>

/** Example for qpOASES main function using the QProblem class. */
int main( )
{
        USING_NAMESPACE_QPOASES

        /* Setup data of first QP. */
        real_t H[2*2] = { 1.0, 0.0, 0.0, 0.5 };
        real_t A[1*2] = { 1.0, 1.0 };
        real_t g[2] = { 1.5, 1.0 };
        real_t lb[2] = { 0.5, -2.0 };
        real_t ub[2] = { 5.0, 2.0 };
        real_t lbA[1] = { -1.0 };
        real_t ubA[1] = { 2.0 };

        /* Setup data of second QP. */
        real_t g_new[2] = { 1.0, 1.5 };
        real_t lb_new[2] = { 0.0, -1.0 };
        real_t ub_new[2] = { 5.0, -0.5 };
        real_t lbA_new[1] = { -2.0 };
        real_t ubA_new[1] = { 1.0 };


        /* Setting up QProblem object. */
        QProblem example( 2,1 );

        Options options;
        example.setOptions( options );

        /* Solve first QP. */
        int_t nWSR = 10;
        example.init( H,g,A,lb,ub,lbA,ubA, nWSR );

```

(continues on next page)

(continued from previous page)

```
        /* Get and print solution of first QP. */
        real_t xOpt[2];
        real_t yOpt[2+1];
        example.getPrimalSolution( xOpt );
        example.getDualSolution( yOpt );
        printf( "\nxOpt = [ %e, %e ];  yOpt = [ %e, %e, %e ];  objVal = %e\n\n",
                        xOpt[0],xOpt[1],yOpt[0],yOpt[1],yOpt[2],example.getObjVal() );

        /* Solve second QP. */
        nWSR = 10;
        example.hotstart( g_new,lb_new,ub_new,lbA_new,ubA_new, nWSR );

        /* Get and print solution of second QP. */
        example.getPrimalSolution( xOpt );
        example.getDualSolution( yOpt );
        printf( "\nxOpt = [ %e, %e ];  yOpt = [ %e, %e, %e ];  objVal = %e\n\n",
                        xOpt[0],xOpt[1],yOpt[0],yOpt[1],yOpt[2],example.getObjVal() );

        example.printOptions();
        /*example.printProperties();*/

        /*getGlobalMessageHandler()->listAllMessages();*/

        return 0;
}
```

**Warning**

If you have the library installed in two directories, you need to ensure you are linking the library you want.

For instance, let's say you have the DQ Robotics library installed globally (i.e., `/usr/local/lib/`) and locally (i.e., `~/opt/lib`), and you want to use the local one. Then, you can use `find_library` with the `NO_DEFAULT_PATH` flag.

```
find_library(my_local_dqrobotics
        NAMES dqrobotics
        PATHS ~/opt/lib
        NO_DEFAULT_PATH
        )
if (my_local_dqrobotics)
    message(STATUS "Local DQ Robotics installed in ${my_local_dqrobotics}")
target_link_libraries(${PROJECT_NAME}
                        ${my_local_dqrobotics})
else()
    message(FATAL_ERROR "Local DQ Robotics not found!")
endif()
```

CHAPTER

SIXTY

# FREQUENTLY ASKED QUESTIONS (FAQ)

**Note**

Also known as, frequently made comments, things I'd like to mention, etc.

## 60.1 You got the name wrong, it's ROS 2 not ROS2

Besides the humorous nature of the meme below and my love for the 1993's blockbuster, this is an inconspicuous way of showing, in every single section, that these tutorials are not official.


## 60.2 It's not Linux, it's GNU/Linux: Keep all grievances in #vent

The wording on these tutorials is precise as possible. Note that some terms are commonly used with loose meanings, but I hope that the message is still conveyed. This applies to the whole tutorial, given that even official sources are not uniform in their terminology.

So, to end any deep discussions that might distract you from the point of these tutorials before they even start, I'll let you with the world-renowned Linux copypasta edited with what was actually said

> *I'd just like to interject for a moment. What you're referring to as Linux, is in fact, GNU/Linux, or as I've recently taken to calling it, GNU plus Linux. Linux is not an operating system [...]. Many computer users run a modified version of the GNU system every day, without realizing it. Through a peculiar turn of events, the version of GNU which is widely used today is often called "Linux," and many of its users are not aware that it is basically the GNU system, developed by the GNU Project. There really is a Linux, and these people are using it, but it is just a part of the system they use.*
>
> *Linux is the kernel: the program in the system that allocates the machine's resources to the other programs that you run. The kernel is an essential part of an operating system, but useless by itself; it can only function in the context of a complete operating system. Linux is normally used in combination with the GNU operating system: the whole system is basically GNU with Linux added, or GNU/Linux. All the so-called "Linux" distributions are really distributions of GNU/Linux.*

## 60.3 The difference between Python *scripts* and *modules*

According to The Python Tutorial on Modules, the definition of *script* and *module* is not disjoint, in fact, it is said that

> *[. . . ] you can make the file usable as a script as well as an importable module [. . . ]*

In the official documentation, a Python script is defined as

> *[. . . ] a [script is a] somewhat longer program, [for when] you are better off using a text editor to prepare the input for the interpreter and running it with [a script] as input instead [of using an interactive instance of the interpreter].*

and a module is defined as

> *[A module is a file] to put definitions [. . . ] and use them in a script or in an interactive instance of the interpreter.*

There are more profound differences in how the Python interpreter handles *scripts* and *modules*, but in the wild the the difference is usually as I described in *Terminology*.

## 60.4 The difference between Python *modules* and *packages*

According to the Holy Book of Modules, a definition of packages is given *en passant* as follows

> *Suppose you want to design a collection of modules (a "package") [. . . ]*

In practice, the line between modules and packages tends to be somewhat blurred. It could be a single folder with many modules but at the same time they come up with namings such as submodule

> *Packages are a way of structuring Python's module namespace [. . . ]. For example, the module name A.B designates a submodule named B in a package named A.*

What most people want to say when they mention a package is, usually, either a folder with a `__init__.py` or a folder with a `setup.py` that can be built into a `wheel` or something similar.

CHAPTER

# SIXTYONE

## DISCLAIMERS

By reading and/or using this tutorial in total or in part, you agree to these terms.

**Disclaimer**

ANYTHING ON THIS TUTORIAL–EVEN THINGS THAT ACTUALLY WORK–IS ENTIRELY FICTIONAL. SOME MEMES ARE ATTEMPTED. . . .POORLY. THE TUTORIAL CONTAINS MISPLACED MOVIE REFERENCES AND DUE TO ITS LOW-HANGING FRUIT HUMOUR, IT SHOULD NOT BE READ BY ANYONE.

**Disclaimer**

All advice, comments, and terrible memes in this tutorial are my own and not endorsed by anyone or anything else mentioned herein. It's not even endorsed by me.

**Disclaimer**

THIS TUTORIAL AND RELATED SOFTWARE ARE PROVIDED "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE AND/OR TUTORIAL, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

CHAPTER

# SIXTYTWO

# CHANGELOG

## 62.1 2025/08-2025/12

- Updated to ROS2 Jazzy.
- Updated theme to `sphinx_book_theme`.
- Moved ROS2 parameter and launch to before the services.
- Added a section about ROS2 Actions.
- Added a section mostly about `tf2`.
- Added a section mostly about **`Gazebo`**.
- Added a section about `nav2`.
- Added a section about cybersecurity.