ROS 2 Getting Started

Author: Manoj Sharma

Robot Operating System (ROS), www.ros.org, is an open source operating system designed for robots. It is "a flexible framework for writing robot software. It is a collection of tools, libraries, and conventions that aim to simplify the task of creating complex and robust robot behavior across a wide variety of robotic platforms."

This guide complements the official ROS 2 tutorial . Throughout this guide, the Operating System of choice is Ubuntu (Linux) 22.04 along with Jazzy version of ROS 2 (or simply ROS). The programming language of choice is Python with a slight mix of C++.

Affiliation Robotic Systems Lab., Santa Clara University, CA
Correspondence msharma@scu.edu, *Github*, *Homepage*
Draft December 2, 2025

CONTENTS

1 PREPARING FOR ROS

Preparing for ROS tutorial requires setting up Ubuntu, installing ROS, knowing the fundamentals of Linux, and a basic understanding of Python (C++ is optional).

ROS 2 Jazzy is natively compatible with Ubuntu 24. The compatibility chart is shown in Figure 1.

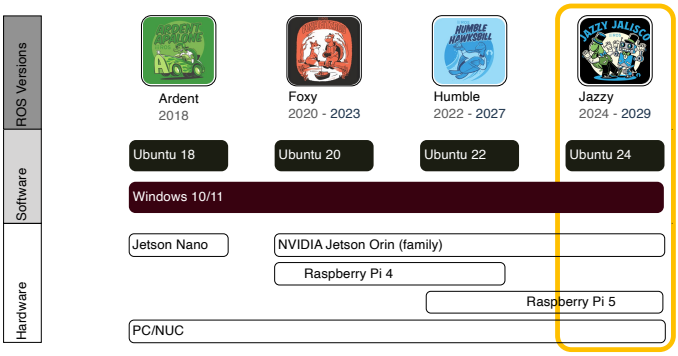


Figure 1: ROS 2 OS and hardware compatibility chart.

1.1 SETUP UBUNTU 24 (NOBLE NUMBAT)

Ubuntu 24 (or Ubuntu 24.04 LTS), codename Noble Numbat is a Long Term Release (LTS) with end-of-life (EOL) of 2029 is a suitable version for various project requiring Ubuntu. Below are three possible ways to setup Ubuntu 24 (pick any one of 1.1.1, 1.1.2, or 1.1.3 although, **1.1.1 is recommended**):

1.1.1 RUN UBUNTU ON A VIRTUAL MACHINE [Recommended] *Click here for instructions.*

1.1.2 NATIVE UBUNTU INSTALL *Click here for instructions.*

1.1.3 UBUNTU ON WINDOWS SUBSYSTEM FOR LINUX (WSL) *Click here for instructions.*

1.2 INSTALL ROS 2 (JAZZY)

Once Ubuntu 24 is setup, *click here* for instructions on ROS 2 Jazzy installation instructions. Additionally, a step-by-step installation guide is available *here*.

1.3 LINUX FUNDAMENTALS

Click here to learn more about Ubuntu Graphical User Interface (GUI).

1.4 Introduction to Python

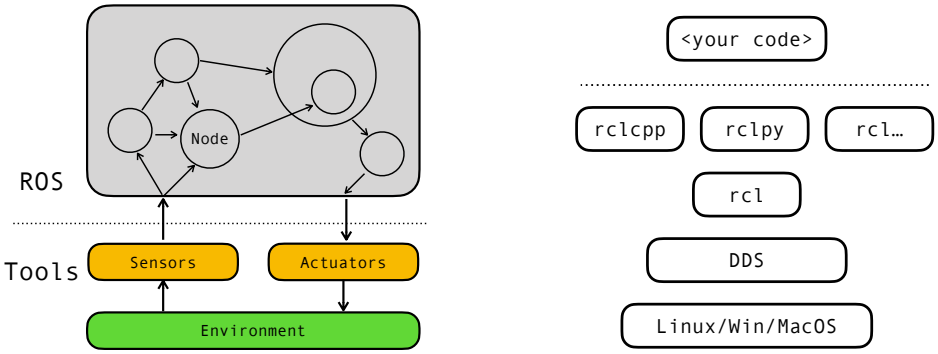
W3schools has put together a step by step Python tutorial that is broken into sub-components. the turotial is *linked here*. Be sure to complete the following components: *Intro* through *Operators, If...Else* and *For Loops*.

1.5 Introduction to C++

Optional - Similarly, W3schools has put together a step by step C++ tutorial that is broken into sub-components. the turotial is *linked here*. Be sure to complete the following components: *Intro* through *Operators, If...Else* and *For Loops*.

2 ABOUT ROS 2

ROS 2 is a ground-up reimagining of ROS 1[1]. ROS 2 was started in 2014. As of 2023 ROS 2 is on eighth named release, called *Jazzy*. ROS 2 Jazzy uses Ubuntu 22 as a host OS to run. Its End-of-Life (EoL) date is May 2027.



(a) ROS 2 organisation (b) ROS big picture

Figure 2: ROS 2 architecture.

ROS 2 is a middleware designed to run on Linux, Windows or MacOS. It is built on DDS[2], a middleware proven in industry. Figure 2b shows this structure. ROS client library is used to communicate between the nodes.

2.1 WHAT IS ROS

ROS allows a problem to be broken down into smaller pieces, nodes and packages which could be developed as a separate entity and can be combined together (similar to *lego* pieces). It provides a framework, tools, and interfaces for distributed development. In essence, ROS is:

2.1.1 PLUMBING Allows data/information between nodes as shown in Figure 2a.

2.1.2 TOOLS Logging, plotting, visualisation, diagnostics.

2.1.3 CAPABILITIES Planning, perception, execution.

2.1.4 ECOSYSTEM Encourages re-use of software pieces and share among the community.

[1]ROS 1 has been around since 2008. It uses custom TCP/IP middleware for the communication network. Its latest release, *Noetic*, has EOL date in May 2025.

[2]Click here for more details on ROS 2 design

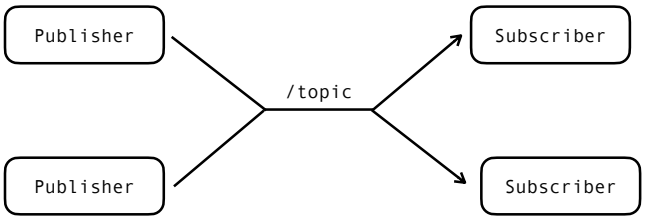


Figure 3: Message: Publisher - Subscriber model.

2.2 Types of Interaction b/w ROS Nodes

ROS offers three core types of interactions between ROS nodes. *Messages* are the simplest type where a publisher continuously broadcasts the message at a user defined rate on a *topic*, refer to Figure 3, (similar to a radio station broadcasts on a specific frequency) to which a subscriber can acquire data by subscribing (similar to tuning a radio to a specific frequency). *Service* and *Actions* are asynchronous means of interact between nodes. Table 1 summarises the strengths and weakness of the three core interactions types.

Type	**Strengths**	**Weaknesses**
Message	Good for most sensors	Easy to overload
	One-to-many	Message drop w/o knowledge
Service	Knowledge of missed call	Blocks until completion
	Well defined feedback	Connection typically re-establishes for each service call (slow activity)
Actions	Monitor long running processes	Complicated
	Handshake	

Table 1: Three core types of interaction between ROS nodes.

2.3 ROS Messages

ROS uses a simplified messages description language for describing the data values (or messages) that ROS nodes publish and/or subscribe:

1. **Standard messages**, *std_msgs*, represents primitive data types and other basic message constructs, such as multiarrays.
2. **Sensor messages**, *sensor_msgs*, represents messages for commonly used sensors, including cameras and scanning laser rangefinders.
3. **Geometry messages**, *geometry_msgs*, represents geometric primitives such as points, vectors, and poses.
4. **Other messages** include *action_msgs*, *diagnostic_msgs*, *nav_msgs*, *shape_msgs*, *stereo_msgs*, *trajectory_msgs*, *visualization_msgs*, and *<custom_msgs>*.
5. **Industrial-core messages**, *industrial_core*, are the messages unique to the industrial manipulator widely used across the industry. Click *here* for more details on ROS Industrial.

2.4 ROS 1 - ROS 2 Bridge [Optional]

There are several instances where a ROS 1 workspace can't be easily converted or ported over to ROS 2 and therefore unable to communicate with ROS 2 workspaces. A workaround to that is to use a ROS 1 to ROS 2 bridge which, as the name suggests, bridges the communication. Click here for more details.

3 ROS 2 TUTORIAL - BASIC

This section covers four main topics - setup a workspace (refer to Figure 4), create a package, create a simple publisher & subscriber nodes, and lastly a launch file to run multiple nodes using a simple launch file.

For additional details, refer to the official tutorial on ROS 2- *click here*.

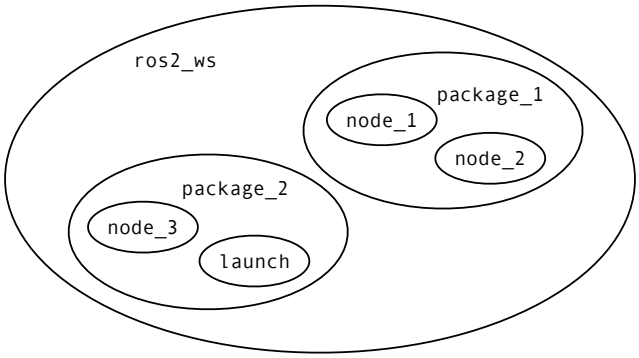


Figure 4: A typical ROS workspace architecture.

3.1 Setup a Workspace

Setting up a workspace refers to creating a workspace directory of choice, resolving the package dependencies (if any), followed by building the workspace.

The first step is to source the ROS 2 environment:

```
source /opt/ros/jazzy/setup.bash
```

Create a workspace in the */home* directory by creating a parent directory of a custom name and a *src* directory within it. Here is how to create a workspace named *ros2_ws*:

```
cd ~/
mkdir -p ~/ros2_ws/src/
```

The directory *ros2_ws* is also known as root directory of this workspace. To move to the root directory from *home* directory:

```
cd ~/ros2_ws/
```

Resolve any package dependencies:

```
rosdep install -i --from-path src --rosdistro jazzy -y
```

Finally, build the workspace:

```
colcon build
```

FYI - The console will return the following (no need to run this commands):

```
Summary: 0 package finished [0.16s]
```

3.2 Create a Package

A package may consists of logical programs (or nodes) for a specific task. This package example consists of a simple subscriber and a publisher.

Navigate to your *src* directory:

```
cd ~/ros2_ws/src
```

Create a package with a name of your choice using *ament python* tools. Syntax to create a package named *'my_package'*:

```
ros2 pkg create --build-type ament_python my_package
```

Next step is to build the package. To do so, move back to the root workspace directory (note - you must be in the root workspace to build[3]):

```
cd ~/ros2_ws/
```

Build the package:

[3] unless any specific packages are installed

```
colcon build
```

FYI - The console will return the following (no need to run these commands):

```
Starting >>> my_package
Finished <<< my_package [1.71s]

Summary: 1 package finished [1.80s]
```

Note - if console responds with "colcon not found" then try installing colcon:

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

3.3 Create/Add Node(s) in a Package

A package may consists of logical programs (or nodes) for a specific task. This example consists of a simple subscriber and a publisher.

Nodes are programs designed for a specific task. A node may consists of subscriber(s) and/or publisher(s). To add a node navigate to the following directory:

```
cd ~/ros2_ws/src/my_package/my_package/
```

3.3.1 Publisher Node A sample publisher python script is available on Github - click here. Download/duplicate this python script in the directory mentioned above.

To keep it simple, use the same file name, i.e. *publisher_member_function.py*. This script initializes the node as *minimal_publisher*, starts to publish *'Hello World'* followed by a timestamp on a topic aptly-named */topic*, and prints it on the console at every $0.5s$. Be sure to read the documentation to understand more about this code.

3.3.2 Subscriber Node A sample subscriber python script is available on Github - click here. Download/duplicate this python script in the directory mentioned above.

To keep it simple, use the same file name, i.e. *subscriber_member_function.py*. This script initializes the node as *minimal_subscriber*, subscribes to a topic aptly-named */topic*, and prints it on the console as it receives the message . Be sure to read the documentation to understand more about this code.

FYI - At this point there should be two additional nodes (one publisher and one subscriber) present within *my_package*. To see these files run:

```
ls ~/ros2_ws/src/my_package/my_package/
```

The console will return the following (no need to run these commands):

```
__init__.py publisher_member_function.py
subscriber_member_function.py
```

3.3.3 Add Entry Points

This is done by editing *setup.py*, located in:

`/ros2_ws/src/my_package/`

Create two entry points for *publisher_member_function.py* and *subscriber_member_function.py* named *pub* and *sub*, respectively. To do so, open *setup.py* using an editor of your choice, scroll to the bottom of the script, and add the following within *'console_scripts'*:

```
'pub = my_package.publisher_member_function:main',
'sub = my_package.subscriber_member_function:main',
```

FYI - The completed *setup.py* file is also available on Guthub - click here.

3.3.4 Build & Run

Navigate to the root of the workspace and then build the package:

```
cd ~/ros2_ws/
colcon build
```

Note - every time changes are made in package and/or nodes, e.g. Python script(s), be sure to “build" and source the workspace before running the node.

To run the publisher - open a new terminal, navigate to the root of the workspace, source ROS 2 environment, and source the workspace:

```
cd ~/ros2_ws/
source /opt/ros/jazzy/setup.bash
source install/setup.bash
```

Use *ros2 run* command to run *pub* node which is present in `my_package` package:

```
ros2 run my_package pub
```

The console will start publishing info, something like this:

```
[INFO] [minimal_publisher]: Publishing: "Hello World: 0"
[INFO] [minimal_publisher]: Publishing: "Hello World: 1"
```

```
[INFO] [minimal_publisher]: Publishing: "Hello World: 2"
...
```

To run the subscriber - open a new terminal, navigate to the root of the workspace, source ROS 2 environment, and source the workspace:

```
cd ~/ros2_ws/
source /opt/ros/jazzy/setup.bash
source install/setup.bash
```

Use *ros2 run* command to run *pub* node which is present in `my_package` package:

```
ros2 run my_package sub
```

The console will start publishing info, something like this:

```
[INFO] [minimal_subscriber]: I heard: "Hello World: 10"
[INFO] [minimal_subscriber]: I heard: "Hello World: 11"
[INFO] [minimal_subscriber]: I heard: "Hello World: 12"
```

3.3.5 Additional Resources/Tools ROS 2 has several tools often helpful in debugging. A few of them are listed below. For more information, refer to a ROS 2 command guide - click here.

While the two nodes are running (in two separate terminals), open another terminal, source the ROS 2 environment:

```
source /opt/ros/jazzy/setup.bash
```

A GUI tool called *rqt_graph* provides a pictorial representation of all the active nodes and ROS structure, shown in Figure 5. To run *rqt_graph* run:

```
ros2 run rqt_graph rqt_graph
```

This will open up a new window showing the active ROS 2 architecture. This can also be used to verify that all the nodes and active properly connected.
Similarly there are many more useful tools for eg., *ros2 topic list*, *ros2 topic echo <topic name>*, *ros2 topic info <topic name>*, etc ...

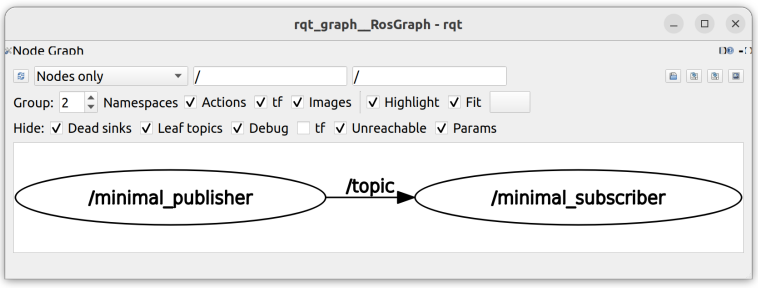


Figure 5: Graphical representation of active nodes and topics.

Note - to stop any activity on a terminal press *Ctrl* + *C*.

3.4 Launch File Setup

A typical workspace may contain several packages where each package may contain multiple nodes in it. This means running multiple nodes one-by-one is very time consuming. A launch file is designed to run and/or launch several nodes and/or launch files. One way to architect this is to create a new package and containing the launch file(s).

3.4.1 Setup a Package (cmake) In the case of a package consisting of launch file(s) it is preferred to build a package using *ament cmake* tools (and not *ament python*). To create a package, navigate to your *src* directory:

```
cd ~/ros2_ws/src
```

Syntax to create a package named *'my_launch_pkg'*:

```
ros2 pkg create --build-type ament_cmake my_launch_pkg
```

Build the package. To do so, move back to the root workspace directory:

```
cd ~/ros2_ws/
colcon build
```

FYI - The console will return the following (no need to run these commands):

```
Starting >>> my_package
Starting >>> my_launch_pkg
Finished <<< my_package [1.27s]
Finished <<< my_launch_pkg [1.57s]

Summary: 2 package finished [1.75s]
```

Next, create a *launch* directory in */ros2_ws/src/my_launch_pkg/*, and change directory to the newly created *launch* directory:

```
cd ~/ros2_ws/src/my_launch_pkg
mkdir launch
cd launch
```

3.4.2 Create a Launch File A sample file python launch script is available on Github - click here. Download/duplicate this python script inside the *launch* directory with a *my_first.launch.py* name. Note that the *entry points* for *pub* & *pub* (Section 3.3.3) acts as executable which are referenced in *my_first.launch.py*. This means launching a (single) *launch* file will run the publisher and subscriber node.

3.4.3 Edit CMake List Lastly the *CMakeLists.txt* in `/ros2_ws/src/my_launch_pkg/` needs to be updated by adding the following:

```
install (DIRECTORY launch DESTINATION share/${PROJECT_NAME})
```

Refer to an updated *CMakeLists.txt* on Github - click here.

3.4.4 Build & Launch Navigate to the root of the workspace and then build the package:

```
cd ~/ros2_ws/
colcon build
```

To launch the *my_first.launch.py* file (in the *my_launch_pkg* package) source the workspace and use the launch command:

```
cd ~/ros2_ws/
source install/setup.bash
ros2 launch my_launch_pkg my_first.launch.py
```

If done correctly, the console should output the publisher and subscriber info similar to the ones discussed in Section 3.3.4.

Additionally, the both the nodes can be verified by running *rqt_graph* which is discussed in Section 3.3.5.

Note that, here, (by design) one launch file is able to start two nodes. It is a good practice to run all the logical nodes via one launch file. There may be cases where a single node needs to be run one at a time for the purpose of debugging.

[Optional] To avoid sourcing the ROS2 environment every time you open a new shell, then you can add the command to your shell startup script

```
echo "source /opt/ros/jazzy/setup.bash" »  /.bashrc
```

Note - sometimes the computer may have multiple versions of ROS installed and therefore the user may need to decide which version to source. In such scenario the the above step is not recommended as it may cause confusion.

3.5 Exercise: Modify ROS Message Type

At this point, you have created a simple publisher and subscriber nodes. The publisher node is publishing "Hello World" message of *Srting* type over a topic called */topic*. Complete the following exercise:

1. Modify the publisher node to publish today's date of *Int16* type over a topic named */date*.
   (Hint: Remember that after making changes to python scripts, e.g. nodes, be sure to "build" and source the workspace before running the node.)
2. Use the CLI tools (for eg. "ros2 topic echo ...") to verify listen to the */date* topic.
3. Open "rqt_graph" to verify that the publisher and subscriber are communicating with each other. Hint - refer to Figure 5.

4 SETUP ROS ON A SINGLE BOARD COMPUTER (SBC)

Often times a control system needs to be deployed on a target SBC (also known as edge computer). Two of the commonly used SBCs for ROS2 are Raspberry Pi 4 (Model B), Raspberry Pi 5 (Model B)[4], and Jetson Orin family.

4.1 RASPBERRY PI

Items needed - Raspberry Pi model B, USB-C power supply, micro-SD card[5], mini-HDMI to HDMI cable, keyboard & mouse, and the internet connection.

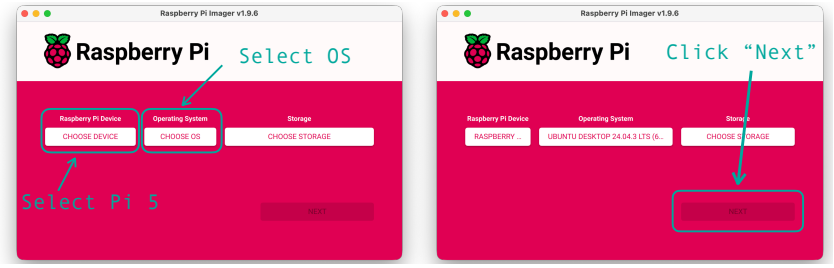


Figure 6: Raspberry Pi Imager: From *CHOOSE DEVICE* menu, select *RASPBERRY PI 5*. From *CHOSE OS* menu, select *UBUNTU DESKTOP 24.04.3 LTS (64-BIT)*. Select the appropriate storage then click *NEXT*, and follow the prompt.

1. **Prepare micro-SD card** - *Raspberry Pi Imager* is an official tool available to create a bootable card for Raspberry pi.
2. **Write the Image** - Open *Raspberry Pi Imager*, Select *Raspberry Pi 5* as the target device then select *Ubuntu 24.04.3 LTS*, and click "NEXT" (shown in Figure 6).
3. **Setup the OS** - Connect the Raspberry pi to a display, power, and follow the screen prompts to setup the OS. Note - using generic password like *ubuntu* helps to conveniently access the account over the network.
4. **Install ROS** - install *ROS 2 jazzy*.
5. **Install SSH Server** - [optional] *Secure Shell (SSH)* allows the Raspberry pi to be controlled via SSH via a different computer. Follow the steps to install *Open SSH Server* on Ubuntu.

[4] preferably 4GB RAM or more.
[5] preferably 32GB or more.

4.2 Jetson Orin Nano Devloper Kit

Items needed - Raspberry Pi model B, USB-C power supply, micro-SD card[6], mini-HDMI to HDMI cable, keyboard & mouse, and the internet connection.

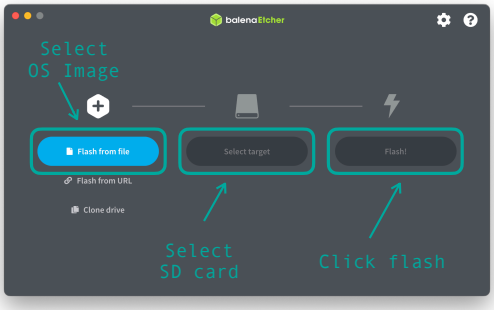


Figure 7: Balena Etcher. Select *Flash from file*, *select target* and then click *Flash!*

1. **Prepare micro-SD card** - *Balena Ecther* is a tool available to create a bootable card for Jetson Orin Nano.
2. **Download *Jetpack*** - The (Ubuntu) OS designed by NVIDIA for Jetson Orin Nano is called *Jetpack*. Click here to download *Jetpack 6.0*, which is based on Ubuntu 22.04 LTS.
3. **Write the Image** - Launch *Balena Etcher*, select *Jetpack 6.0*, and then follow the prompts shown in Figure 6 to write the image.
4. **Setup the OS** - Connect the Jetson to a display, power, and follow the screen prompts to setup the OS. Note - using generic password like *ubuntu* helps to conveniently access the account over the network.
5. **Install ROS** - install *ROS 2 Humble*[7].

[6] preferably 32GB or more.
[7] Note that the *Jetpack 6.x* is based on Ubuntu 22, and therefore ROS2 Humble should be installed.

4.3 Jetson Orin AGX/NX Devloper Kit

Other family of Jetson Orin, for eg. *Jetson Orin AGX* or ***Jetson Orin NX***, require *NVIDIA SDK Manager* to install the OS (or Jetpack). Once done, install *ROS 2 Foxy*.

5 ROS-ARDUINO/PERIPHERAL COMMUNICATION

A simple serial-communication protocol is an easy way to establish ROS-ARDUINO communication. In this tutorial, we use an Arduino to broadcast two integer comma-separated values ending with a newline. Simultaneously, the Arduino can receive an 8-bit integer value and convert it into an LED brightness. A sample script is available here.

5.1 PYSERIAL

Install *pyserial* module. Follow the official documentation listed here.

5.2 CLONE REPOSITORY

Navigate to your *src* directory and clone the repository (which is arduino_pkg):

```
cd ~/ros2_ws/src
git clone https://github.com/irahulone/arduino_pkg.git
```

5.3 UPLOAD ARDUINO-SCRIPT

An example Arduino-script is available under the *src* directory of arduino_pkg; alternatively, click here to download the script.
Note - if the Virtual Machine (VM) is unable to detect the connected Arduino via the USB then refer to Section 7.2.

[Optional:] Use the following command to install Arduino IDE, if you don't have it. Arduino IDE is needed to upload the Arduino-script.

```
sudo apt install arduino
```

5.4 BUILD & RUN

Navigate to the root of the workspace, build the package, and run the node:
(If the console responds with *"Error opening serial port,"* refer to Section 7.3.)
(Make sure to use the appropriate serial port for Arduino. By default it is set to `/dev/ttyACM0`/)

```
cd ~/ros2_ws/
colcon build
ros2 run arduino_pkg read_write
```

5.4.1 VISUALISE TOPICS AND INTERACT In a new terminal check the available topics (`value_a, value_b`) and *echo* them to verify the data reception.
Use the following command to publish the value(s) on a topic (`write`) directly from the console (click here for more details):

```
ros2 topic pub write std_msgs/Int16 "data:  40"
```

We know that the number `40` is the value sent to the Arduino, which is then translated into the brightness of the onboard LED. Try replacing this number with a different one ranging from `0` to `255` and observe the brightness of the LED. change corresponding to the number.

6 ROS DATA VISUALIZATION AND LOGGING

To learn more about visualization, logging, and exporting ROS data, let's use a dummy package called, `fake_data_broadcaster` available on GitHub (click here). This package contains an example node; upon starting, it starts to broadcast a modeled temperature, humidity, and time information on a typical day[8] on topics `\temperature`, `\humidity`, and a few more, respectively.

CLONE REPOSITORY

Navigate to `src` directory and clone the `fake_data_broadcaster` repository :

```
cd ~/ros2_ws/src
git clone https://github.com/irahulone/fake_data_broadcaster_pkg.git
```

BUILD & RUN Navigate to the root of the workspace and then build the package:

```
cd ~/ros2_ws/
colcon build
```

Note - every time changes are made in package and/or nodes, e.g. Python script(s), be sure to "build" and source the workspace before running the node.

Ssource the workspace:

```
cd ~/ros2_ws/
source /opt/ros/jazzy/setup.bash
source install/setup.bash
```

Use *ros2 run* command to run *fake_weather_broadcaster*:

```
ros2 run weather_data_publisher weather_data_publisher_node
```

Once this node is running, the temperature, humidity, and time data will begin to broadcast on `\temperature`, `\humidity`, and a few more topics, respectively.

[8] The data are published at 1 Hz, with each successive value corresponding to a 15-minute increment. This means that the 24-hour day is represented by 96 discrete data points, spanning 98 seconds. The node continues to broadcast in a loop until it is stopped.

6.1 DATA VISUALIZATION

Data, in ROS environment, can be visualized in various ways. Tools such as *ros2 topic echo* and *rqt_plot* can be used to visualize a data in numeral-format and plot-format, respectively.

Note - various third party tools exists (for e.g., PlotJuggler) with advanced plotting visualization and plotting capabilities.

6.1.1 ROS2 TOPIC ECHO Use this command to list all the available topics:

```
ros2 topic list
```

To see the data being published on a topic in a human readable, here is the general syntax (no need to run this command):

```
ros2 topic echo <topic_name>
```

For example, to see the data being published on a topic named `/temperature` is:

```
ros2 topic echo /temperature
```

You should see the data in real time in the terminal, as shown in Figure 8.

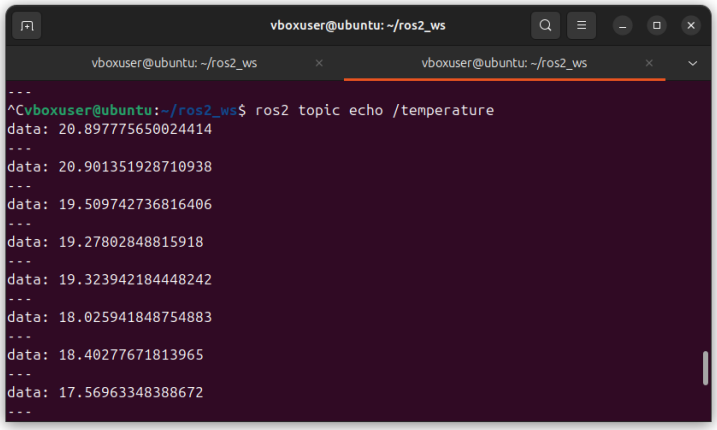


Figure 8: Example of an *ros2 topic echo* window.

6.1.2 RQT_PLOT It is a tool that provides a simple and quick way to visualize the flow of numerical data from a topic in real time. It's perfect for debugging, monitoring, and understanding the data flow.

To launch *rqt_plot* window, run this command in a new terminal window:

```
ros2 run rqt_plot rqt_plot
```

This should launch an *rqt_plot* window (you may have to resize the window), shown in Figure 9.
Once open, add the topic by typing the topic name in the dialog box in the top left corner of the window.

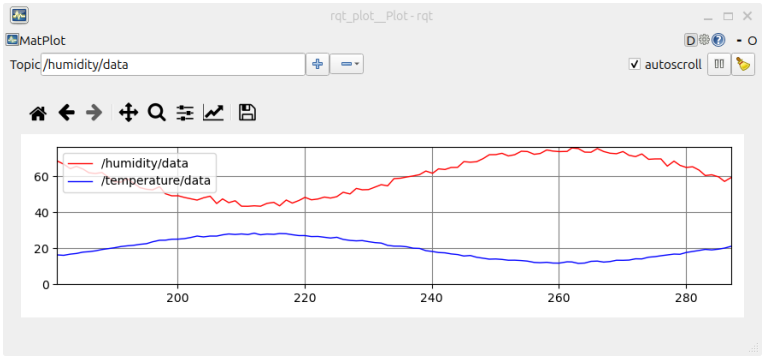


Figure 9: Example of an *rqt_plot* window.

6.2 DATA LOGGING

ROS has an inbuilt tool, called *rosbag record* (click here for official documentation), to record the messages from various topics during runtime. It is a part of the rosbag toolset, which is used to record, play back, and analyze ROS message data. When you use rosbag record, it saves the messages published on the selected topics to a file, which can later be used for analysis, debugging, or reproducing experiments.
Source the workspace:

```
cd ~/ros2_ws/
source /opt/ros/jazzy/setup.bash
source install/setup.bash
```

6.3 EXPORTING DATA TO CSV

7 ADDITIONAL RESOURCES

7.1 INSTALL MICROSOFT VISUAL CODE

7.1.1 X86 For Ubuntu running on **x86**, open **Ubuntu Software**, search for *visual studio code*. Generally the first result app is the **VS Code**, click *install* and follow the directions.

7.1.2 ARM For Ubuntu running on **ARM**, for example Raspberry pi, NVIDIA Jetsons, or other ARM architecture based devices:

```
sudo apt update
sudo apt install code
```

7.2 USB PORT HANDOVER TO VIRTUAL MACHINE

Arduino may require additional steps when connected to Virtual Machine (VM) to ensure the serial port is recognized by the VM.

Refer to this how to guide on Control Arduino from Ubuntu VirtualBox by The Robotics Back-End.

7.3 ISSUE - "ERROR OPENING SERIAL PORT"

Accessing serial port require additional permission. To set serial port permission, first determine the serial port:

```
ls /dev/tty*
```

The serial port may be `USBx` or `ACMx`. If the port in question is `USBx`, run the following command

```
ls -l /dev/ttyUSB
```

FYI - The console may return something similar

```
crw-rw---- 1 root dialout 188, 0 5 apr 23.01 ttyUSB0
```

Add the Linux user to the dialout group (replace the <username> with the appropriate username)

```
sudo usermod -a -G dialout <username>
```

Log out and log in again (or reboot) for this change to take effect.

7.4 Issue - Add *"sudo"* rights

link here