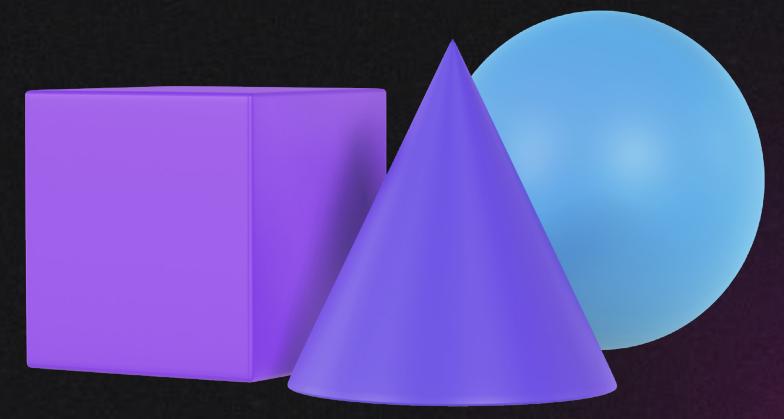


# nv

## The Modern Robotics Framework

With the nv framework, building complex robotic systems has never been easier.



## Simple

Gone are the days of spending countless hours understanding and mastering existing robot frameworks. nv is built to be logical to understand and use, lowering the entry barrier for functional robotics.



## Modern

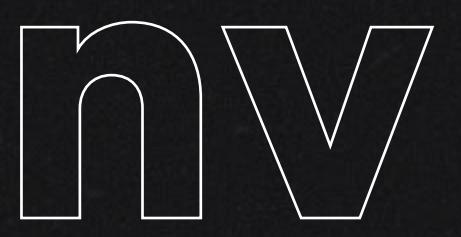
nv supports the world's two most popular programming languages; JavaScript and Python\*. This enables a wide range of developer support, and promotes learning of highly transferrable skills.

\* According to StackOverflow's 2021 Developer Survey, excluding HTML/CSS.



## Adaptable

No assumptions are made about the deployment environment. As a result, nv works great for robots with wheels, tracks, or propellers, programmable robot arms, autonomous distributed workers, and everything in-between.



# **nv** **Concepts**

## **Node**

An entity within the robot system which is responsible for a defined task, such as reading sensor data, processing information, or sending commands to physical actuators.

## **Pub/Sub**

The primary communication method between nodes is through an anonymous publish/subscribe model. Any node can broadcast information to the system, and any other node can chose to listen.

## **Redis**

The backbone of nv communication is a highly performant, open source, in-memory data store, with the ability to run on virtually any device.

A single nv robot can be distributed across many different nodes, containers, physical devices, or even wide area networks. For example, a cloud-based processing server can talk to low-power embedded devices for increased system efficiency. nv doesn't place limits on where or how nodes can run.

```
import json
import os
import pathlib
import subprocess
import threading
import time

import numpy as np
from nv.node import Node

CONFIG_FILE = pathlib.Path(__file__).parent / "config.yml"

class JoyNode(Node):
    def __init__(self):
        super().__init__("joy_node")

        # Load the config file for button mapping
        self.set_parameters_from_file(str(CONFIG_FILE))

        self.JOYSTICK_DEVICE = self.get_parameter("device") or "/dev/input/js0"

        # Load the connected gamepad
        self.gamepad = self._get_gamepad()

        # Start the error checking thread
        self._js_error_check_thread = threading.Thread(
            target=self._js_error_check, daemon=True
        )
        self._js_error_check_thread.start()

        # Read the name of the device
        while not (gamepad_name := self.gamepad.stdout.readline()).startswith("N:"):
            time.sleep(0.1)

        gamepad_name = gamepad_name[2:].strip()

        # Attempt to match the gamepad to a config
        mapping_scheme = self.get_parameter("devices", gamepad_name)
```

```
joy_node:
    # Override the default joystick device, defaults to /dev/input/js0
    device: /dev/input/js0

    # Map the name of the USB device to a button mapping scheme. A gamepad which
    # doesn't match one of the devices here will be ignored.
    devices:
        Generic X-Box pad: xbox_series
        Sony PLAYSTATION(R)3 Controller: ps3
        Sony Interactive Entertainment Wireless Controller: ps4
        Sony Computer Entertainment Wireless Controller: ps4
```

# Easy to setup and use

Comparing the steps required to build a simple publisher node, starting with a clean install.

nv

1. Install nv with **pip** (7 top level dependencies) or **npm** (3 top level dependencies)
2. Write the node
3. Run the node with `python3 subscriber_node.py` or `node subscriberNode.js`

```
○○○  
1 from nv.node import Node  
2  
3 class Publisher(Node):  
4     def __init__(self):  
5         super().__init__("")  
6         self.timer = self.create_timer(0.5, self.publish_hello_world)  
7         self.counter = 0  
8  
9     def publish_hello_world(self):  
10        self.publish("Hello World")  
11        self.counter += 1  
12  
13 def main():  
14     node = Publisher()  
15     node.spin()  
16  
17 if __name__ == "__main__":  
18     main()
```

1. Install dependencies (11 to 33 top level dependencies depending on OS)
2. Install ROS from Debian / brew / chocolatey package manager using pre-built binaries\*  
\* If using an unsupported operating system such as non-debian Linux, build binaries from source
3. Source the ROS installation  
`. /opt/ros/foxy/setup.bash`
4. Create a ROS package  
`ros2 pkg create --build-type ament_python <package_name>`
5. Write the node
6. Add `rclpy` and `std_msgs` dependencies to `package.xml`
7. Update `setup.py` with the node as an entry point
8. Build the package with `colcon build`
9. Source the local workspace with `. install/setup.bash`
10. Run the node with `ros2 run <package_name> publisher_node`

ROS

```
○○○  
1 import rclpy  
2 from rclpy.node import Node  
3 from std_msgs.msg import String  
4  
5 class Publisher(Node):  
6     def __init__(self):  
7         super().__init__("publisher_node")  
8         self.pub = self.create_publisher(String, "hello_world", 10)  
9         self.timer = self.create_timer(0.5, self.publish_hello_world)  
10        self.counter = 0  
11  
12    def publish_hello_world(self):  
13        msg = String()  
14        msg.data = "Hello World"  
15        self.pub.publish(msg)  
16        self.counter += 1  
17  
18 def main():  
19     rclpy.init()  
20     node = Publisher()  
21     try:  
22         rclpy.spin(node)  
23     except KeyboardInterrupt:  
24         node.destroy_node()  
25         rclpy.shutdown()  
26  
27 if __name__ == "__main__":  
28     main()
```

# 419%

Faster messaging than ROS Foxy

Tested using common data types, and a  
Python node for both frameworks.

# 3x

Faster startup time than ROS Foxy

When launching the provided message  
publisher examples.

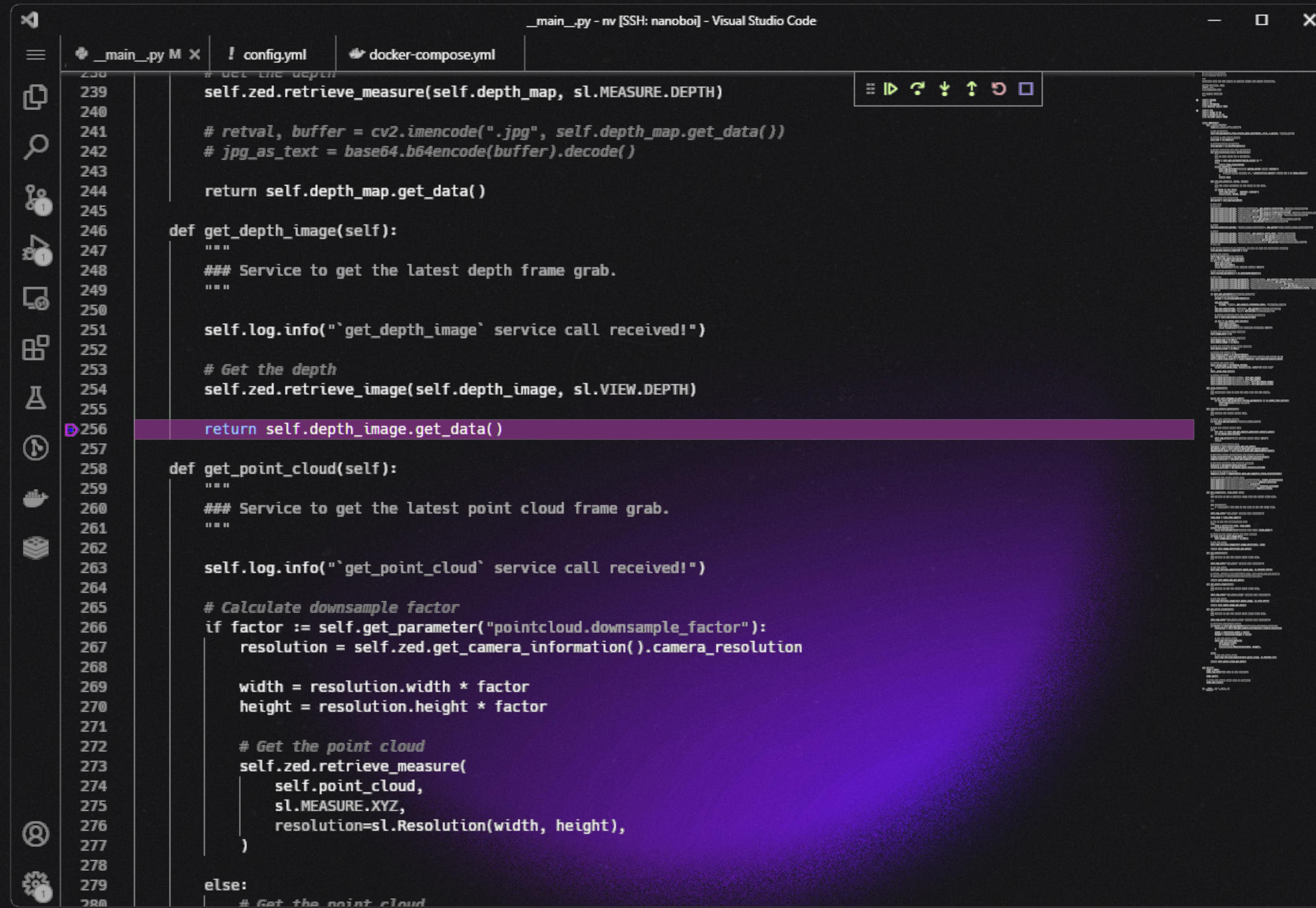
# 0.2ms

Average latency for basic data types

Using the Python framework and TCP  
Redis connections over localhost.

It's fast too.

# Debugging support as standard



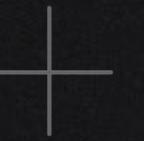
The screenshot shows a Visual Studio Code window with a dark theme. The left sidebar contains icons for file operations like Open, Save, Find, Cut/Copy/Paste, Undo/Redo, and others. The main editor area displays a Python script named `_main_.py`. The code implements two services: `get_depth_image` and `get_point_cloud`. The `get_depth_image` service uses the `zed.retrieve_image` method with the `sl.VIEW.DEPTH` parameter. The `get_point_cloud` service first checks if a `pointcloud.downsample_factor` parameter is set. If it is, it calculates a new resolution by multiplying the current resolution's width and height by the factor. It then uses the `zed.retrieve_measure` method with the `sl.MEASURE.XYZ` parameter and the calculated resolution. If no factor is provided, it falls back to getting the point cloud directly. The code uses the `self.log.info` method to log service calls. The right side of the screen shows a vertical stack of tabs representing other open files in the workspace.

```
239     self.zed.retrieve_measure(self.depth_map, sl.MEASURE.DEPTH)
240
241     # retval, buffer = cv2.imencode(".jpg", self.depth_map.get_data())
242     # jpg_as_text = base64.b64encode(buffer).decode()
243
244     return self.depth_map.get_data()
245
246 def get_depth_image(self):
247     """
248     *** Service to get the latest depth frame grab.
249     """
250
251     self.log.info(`get_depth_image` service call received!")
252
253     # Get the depth
254     self.zed.retrieve_image(self.depth_image, sl.VIEW.DEPTH)
255
256     return self.depth_image.get_data()
257
258 def get_point_cloud(self):
259     """
260     *** Service to get the latest point cloud frame grab.
261     """
262
263     self.log.info(`get_point_cloud` service call received!")
264
265     # Calculate downsample factor
266     if factor := self.get_parameter("pointcloud.downsample_factor"):
267         resolution = self.zed.get_camera_information().camera_resolution
268
269         width = resolution.width * factor
270         height = resolution.height * factor
271
272         # Get the point cloud
273         self.zed.retrieve_measure(
274             self.point_cloud,
275             sl.MEASURE.XYZ,
276             resolution=sl.Resolution(width, height),
277         )
278
279     else:
280         # Get the point cloud
```

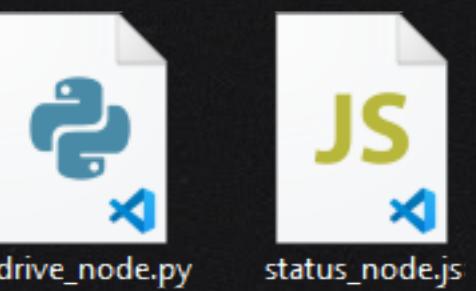
As a standard Python or Node.js package, debugging requires no additional configuration in your existing IDE.

# Write how you like

Write how you like



Mix and match Python and JavaScript nodes, they communicate fluently with each-other (even when using non-standard data types like NumPy arrays).



OOP or Functional programming, use in a module or a script, nv works flawlessly in all situations.

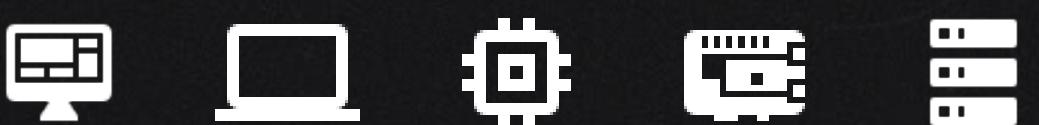
```
class ServiceClient extends Node {  
    async init() {  
        await super.init();  
        await this.waitForServiceReady("odd_even_check");  
  
        const node = new Node({ nodeName: "service_client" });  
        await node.init();  
        await node.waitForServiceReady("odd_even_check");  
    }  
}
```



Docker support is included out-of-the-box. Optimised containers make the development and deployment process even easier.



An integrated command line interface enables interaction with an nv system, for inspection of nodes, or real-time modification of parameters.



Runs on Linux, macOS and Windows. x86 or ARM. nv doesn't impose arbitrary restrictions.

Advanced monitoring and visualisation.



**NODES LIST**

- nv\_tree\_mode\_request\_converter
- nv\_tree
- nv\_io\_led
- nv\_tree\_internal\_node
- nv\_odrive
- nv\_io\_buttons
- nv\_joystick\_converter
- custom\_joy\_node
- nv\_diffdrive

**TOPICS LIST**

|                     |               |
|---------------------|---------------|
| mode_request        | 3 minutes ago |
| blackboard_activity | 1 second ago  |
| joy                 | 1 second ago  |

**SERVICE LIST**

- led\_server
- led\_error
- get\_blackboard\_state
- clear\_blackboard
- get\_costmap
- append\_costmap

**CALL SERVICE - LED\_SERVER**

**Service**

led\_server

**Args**

Kwargs

**TOPIC - SENSORS.DIST.LEFT**

**TOPIC - SENSORS.DIST.RIGHT**

0.153466    0.176844

**PUBLISH - VELOCITY**

**Topic**

velocity

**Data**

{"left": 1.15, "right": 1.15}

Coming soon...

## **Get in touch**

Sales: [ed@unmnd.com](mailto:ed@unmnd.com)

Technical: [callum@unmnd.com](mailto:callum@unmnd.com)