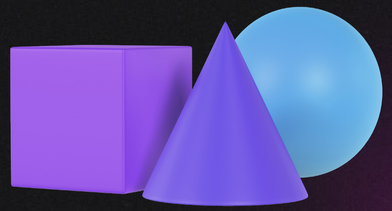
nv

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 worlds 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 propellors, 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.

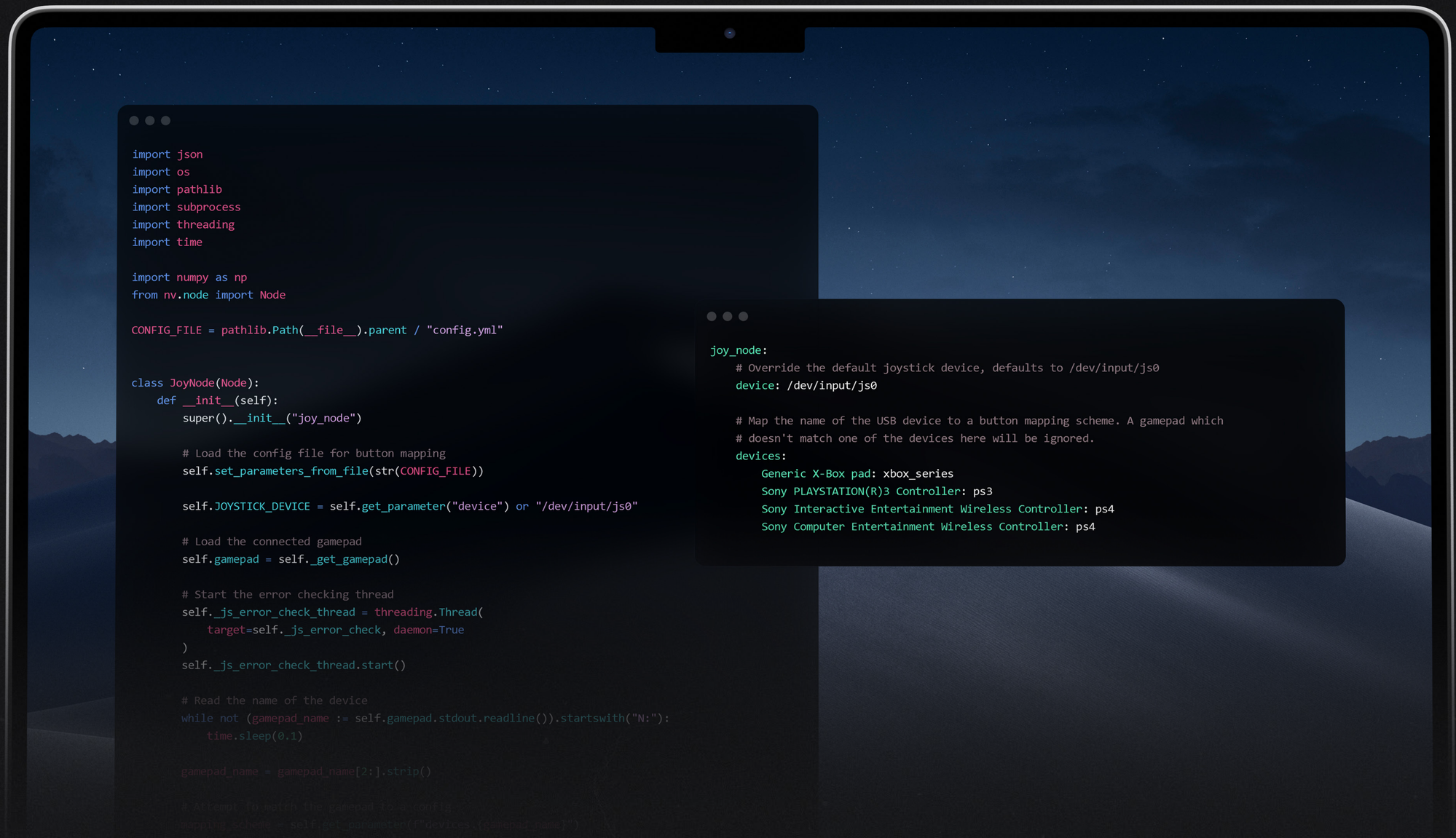
Pub/Sub

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.



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`

```python
from nv.node import Node

class Publisher(Node):
    def __init__(self):
        super().__init__("
        self.timer = self.
        self.counter = 0

    def publish_hello_worl
        self.publish("hell
        self.counter += 1

def main():
    node = Publisher()
    node.spin()

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

ROS

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`

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

class Publisher(Node):
    def __init__(self):
        super().__init__("

        self.pub = self.cre
        self.timer = self.
        self.counter = 0

    def publish_hello_worl
        msg = String()
        msg.data = "Hello W

        self.pub.publish(ms
        self.counter += 1

def main():
    rclpy.init()
    node = Publisher()

    try:
        rclpy.spin(node)
    except KeyboardInterru
        node.destroy_node(
        rclpy.shutdown()

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

Robot Operating System is the current leading robotics framework for custom robotics.

It’s fast too.

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

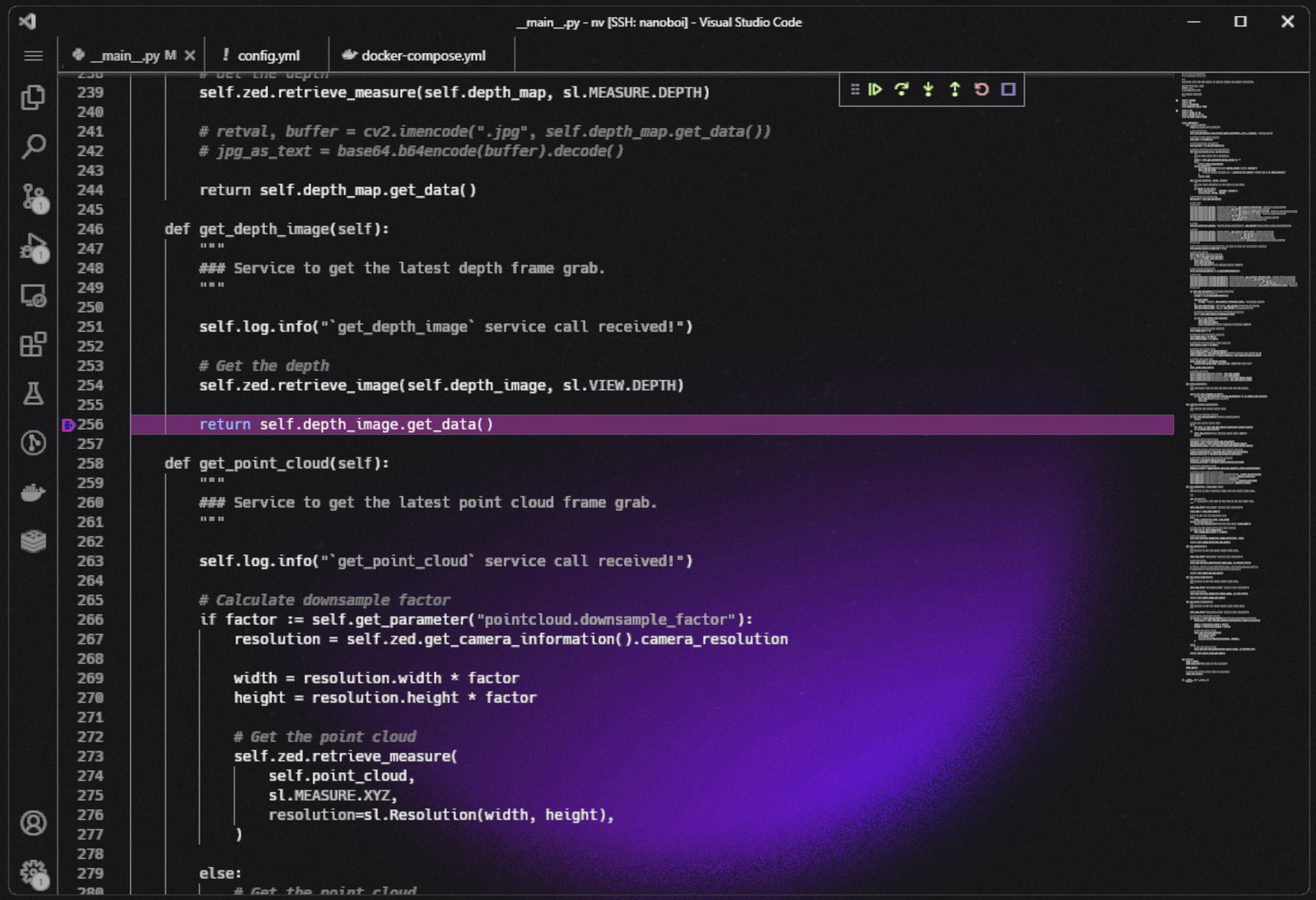
When launching the provided message publisher examples.

0.2ms

Average latency for basic data types

Using the Python framework and TCP Redis connections over localhost.

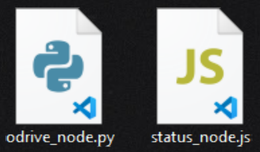
Debugging support as standard



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

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.

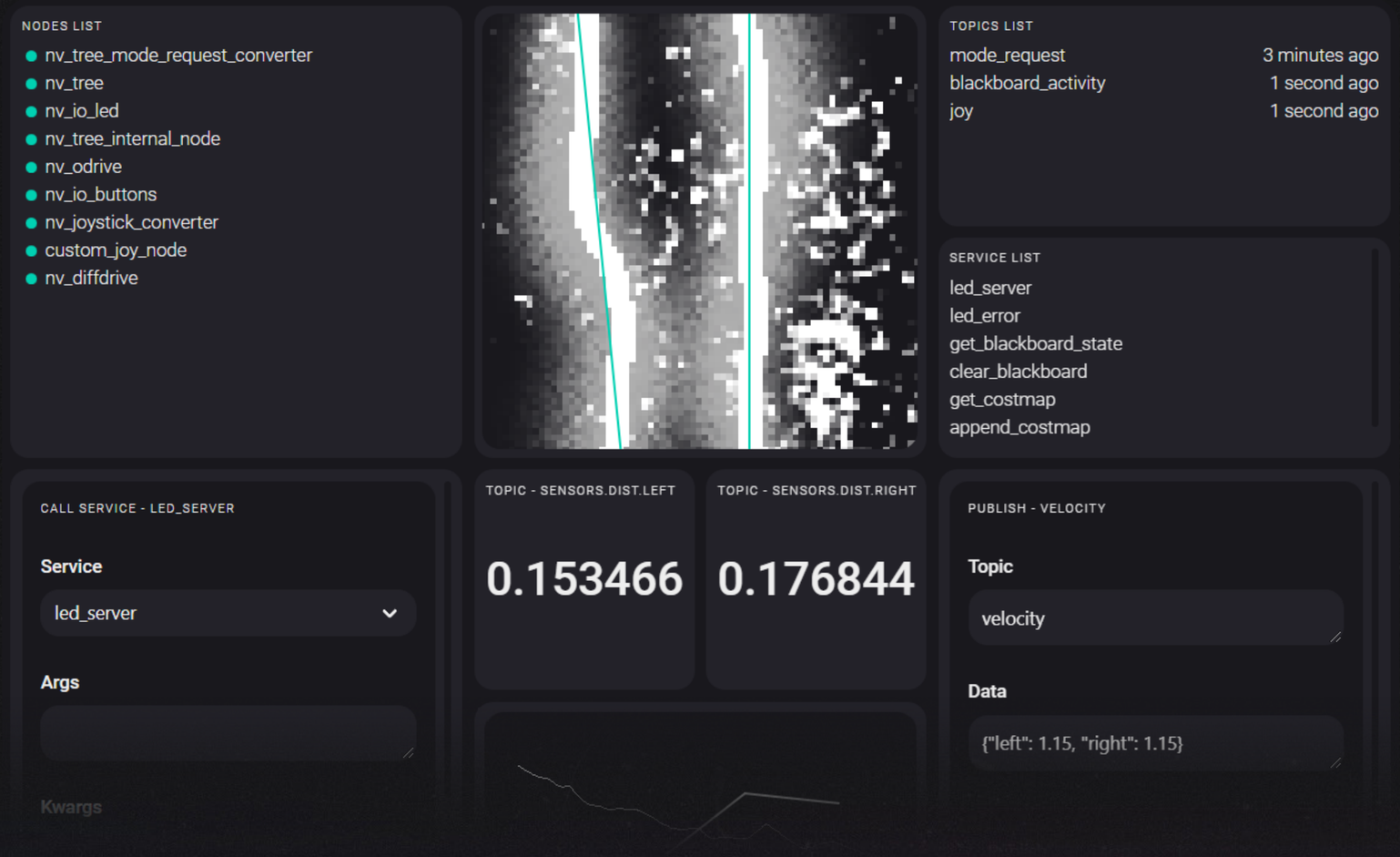
nvcli

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.

nvis



Coming soon...

Get in touch

Sales: ed@unmnd.com

Technical: callum@unmnd.com