JC3504 Robot Technology

Lecture 2: Robot Operating System (ROS)

Dr Xiao Li xiao.li@abdn.ac.uk

Dr Junfeng Gao junfeng.gao@abdn.ac.uk

Outline

Robot Operating System (ROS)

Robot Operating System (ROS) is a flexible framework for writing robot software, offering a collection of tools, libraries, and conventions that aim to simplify the complex process of creating robust and versatile robotics applications.

Robot Operating System (ROS)

<video 02 - ROS Intro.mp4>

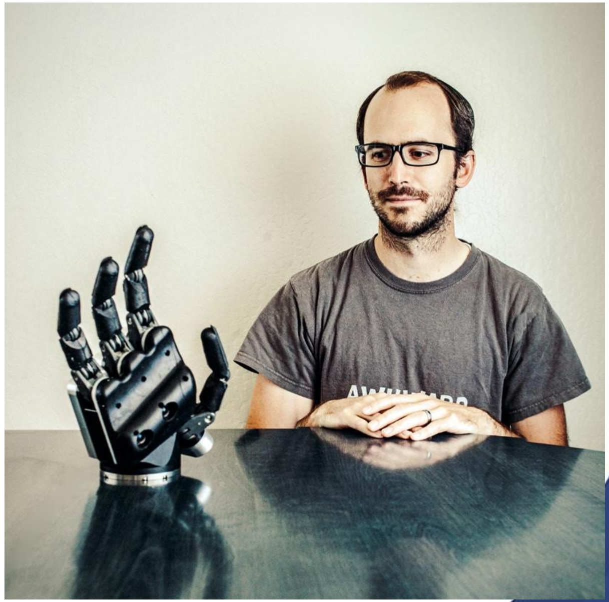
History of ROS

History of ROS

In 2006, Morgan Quigley was a doctoral student at Stanford University, specialising in robotics, under the supervision of Andrew Ng.

Morgan Quigley was one of the members of the STanford AI Robot (STAIR) project, responsible for the integration and adaptation of robot code.

Andrew Ng



Morgan Quigley

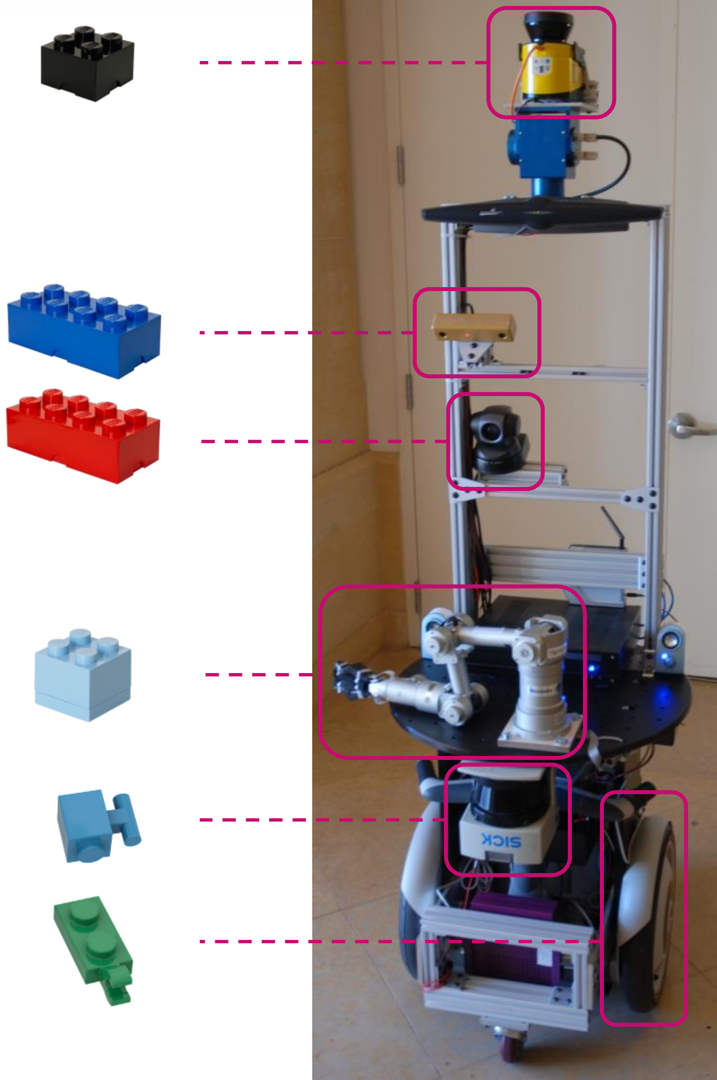
History of ROS

At the time, the STanford AI Robot project encompassed numerous similar robots, rather than just a single model. This made Morgan's code maintenance tasks exceedingly difficult. Each different model of robot operated on independent programmes.

This leaded Morgan to attempt the development of a modular, distributed framework to reduce his workload.

History of ROS

Within ROS, a complete robotic program is decomposed into independent modules (referred to as nodes), with each sensor corresponding to a specific node. Moreover, each actuator and even each algorithm can exist as standalone nodes. When constructing a robot, one simply needs to combine these nodes much like assembling Lego bricks.

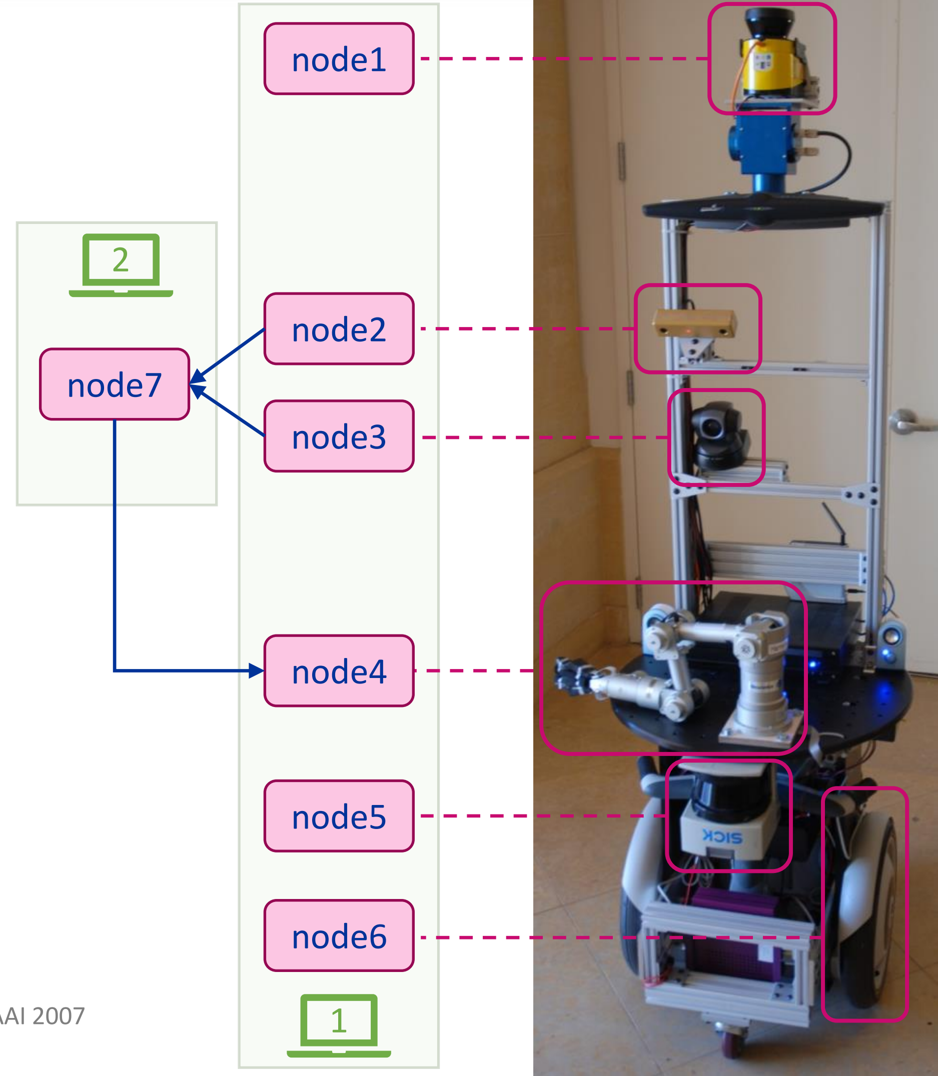


1495 UNIVERSITY OF ABERDEEN

History of ROS

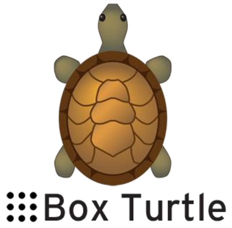
The program is not only divided into separate nodes, but nodes can also operate on different computers. Additionally, each node can be developed in different programming languages (commonly C++ or Python).

This allows for the modules controlling the robot to run on smaller, less powerful computers (such as a Raspberry Pi), while nodes handling tasks like robot vision can operate on more powerful computers.



History of ROS

Subsequently, the Willow Garage company, in collaboration with Ng, developed a robotics project, with Morgan serving as the lead researcher. Within this project, he refined the concept of ROS and applied the first version of ROS (ROS 1.0 -- Box Turtle) to the company's first robot product, the PR2.



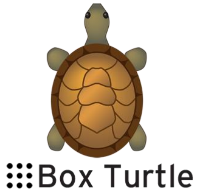
PR2 (2010)

Quigley et al "ROS: an open-source Robot Operating System." ICRA 2009.

History of ROS

Open Source Robotics Foundation

In 2012, Morgan obtained his Ph.D. and established the ROS Foundation (ROSF), dedicated to the development and maintenance of ROS. The various versions of the ROS system that we use today are developed and maintained by this foundation.



1495 UNIVERSITY OF ABERDEEN

History of ROS

ROS has two main versions: ROS 1 and ROS 2.

ROS 1 focuses on providing a robust framework for robotics software development, primarily designed for research and educational purposes, with a single-threaded communication model. It is mainly targeted at Ubuntu.

ROS 2 is an evolution aimed at supporting real-world applications, featuring a multi-threaded architecture with enhanced security, real-time capabilities, and support for **multiple operating systems**, catering to both research and industrial needs.

The course will introduce ROS 2.

ROS 1

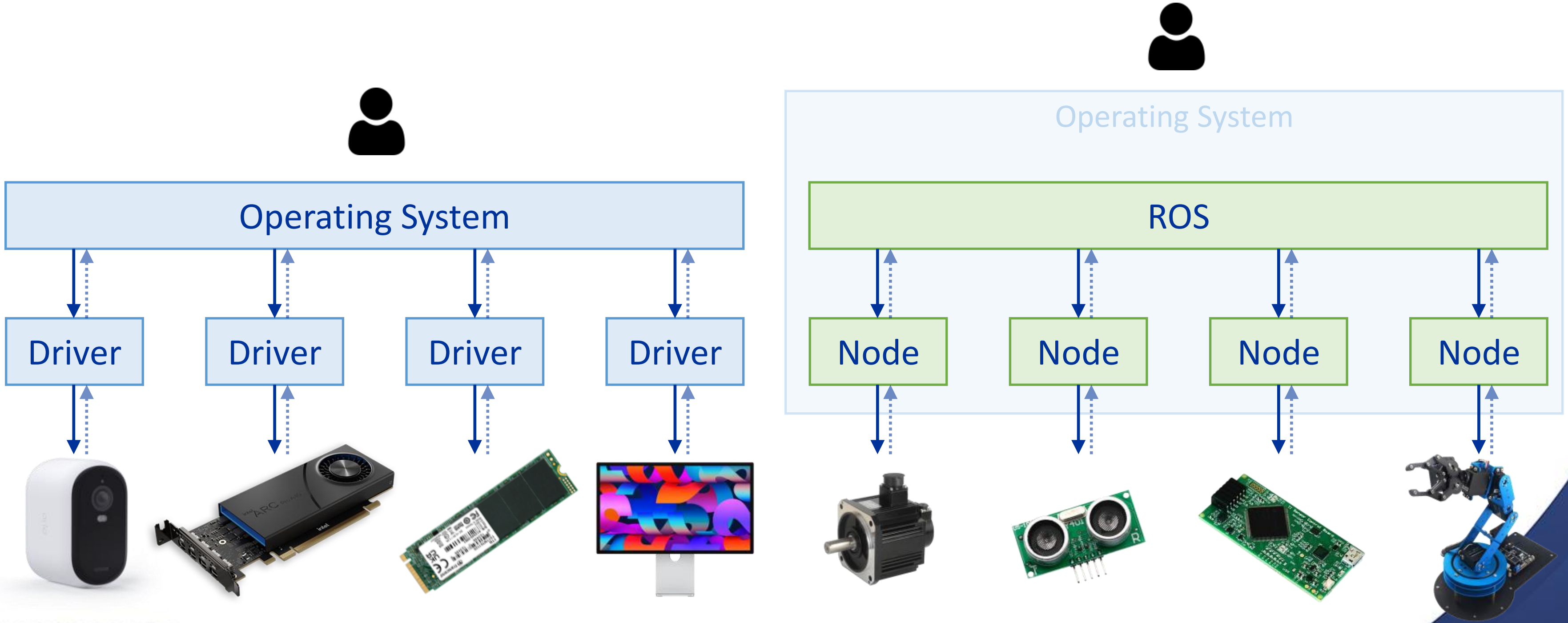
ROS 2

ROS Philosophy

- ROS systems are designed around a peer-to-peer architecture, comprising numerous compact computer programs that communicate by continuously exchanging messages.
- The framework is tool-based, offering an array of small, generic programs capable of performing various tasks, including visualization, logging, and plotting data streams.
- It is multi-lingual, allowing software modules to be developed in any programming language for which a client library is available, with current support for C++, Python, LISP, Java, JavaScript, MATLAB, Ruby, and more.
- ROS promotes a thin design philosophy, encouraging the creation of standalone libraries that are then adapted to communicate with other ROS modules via messages.
- ROS is entirely **free and open-source**, fostering a collaborative environment for development and innovation in robotics software.

1495 UNIVERSITY OF ABERDEEN

ROS vs Operating System

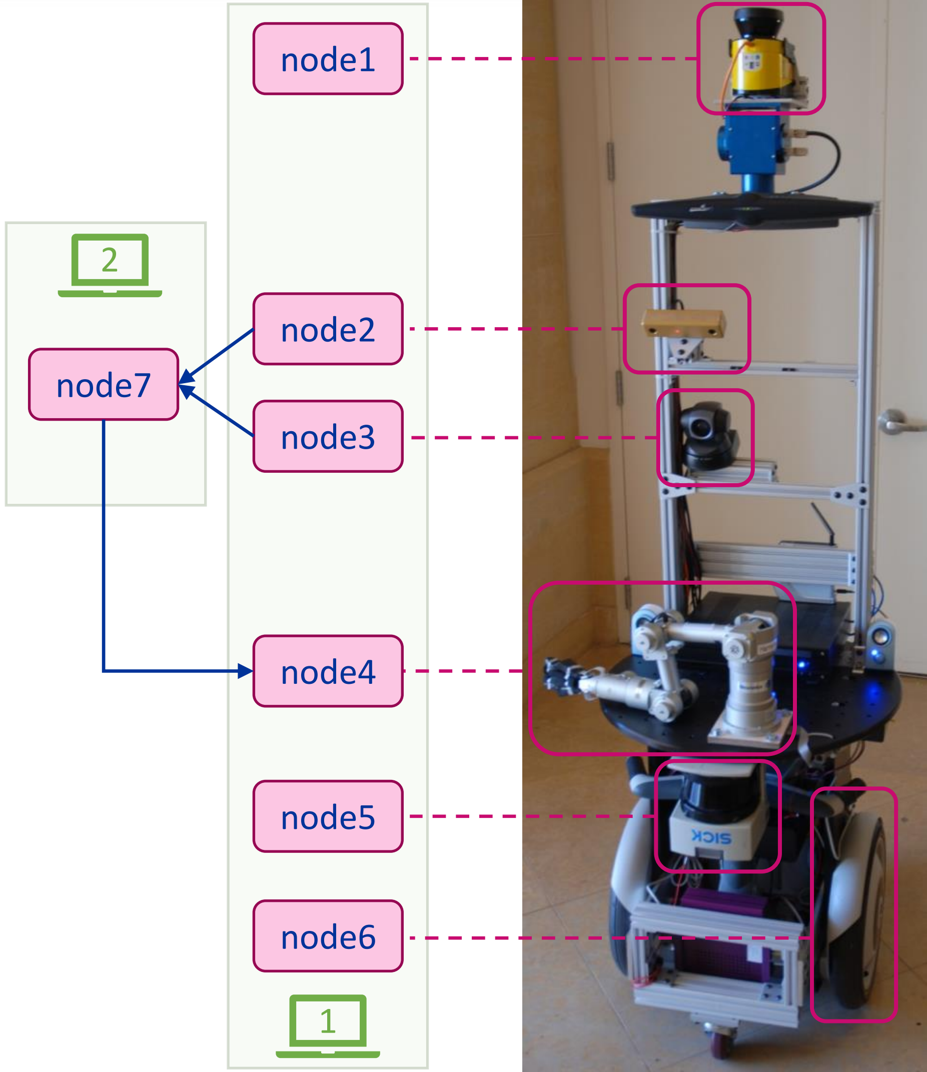


ROS Core Concepts

Node

In ROS, a node represents the smallest executable entity (i.e. program) within the system.

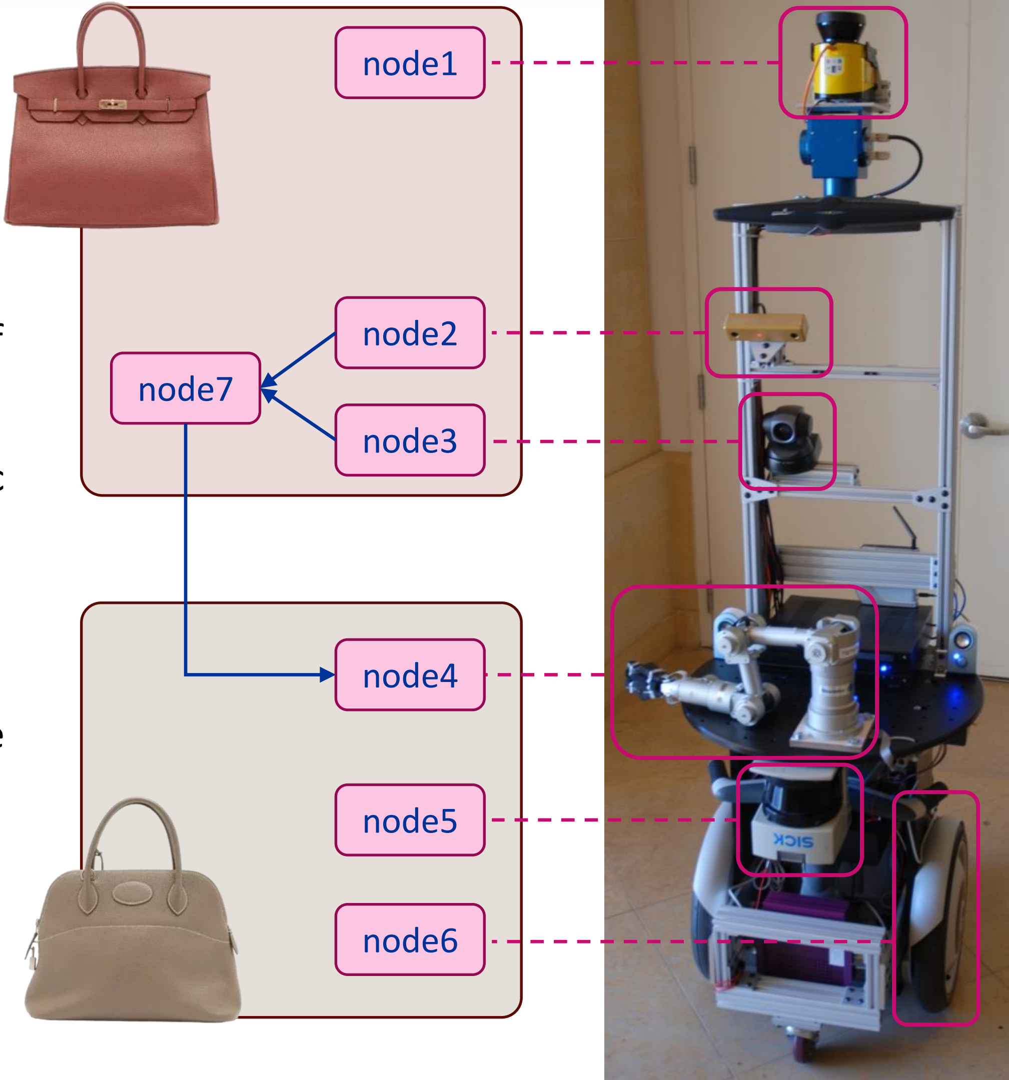
Nodes are designed to perform specific tasks, ranging from sensing and actuation to computation and decision-making processes. Under normal circumstances, each node is responsible for a single function. A robot with complex functionalities is composed of a large number of nodes with simple functions.



Package

A package serves as the fundamental unit of software organisation, encompassing a collection of nodes, libraries, and other resources unified towards fulfilling a specific functionality or purpose within a robotics application.

Package in ROS is like libraries in Python. When you use nodes, especially, when you download nodes, you need to download the package containing the nodes.



1495 UNIVERSITY OF ABERDEEN

Discovery of Nodes

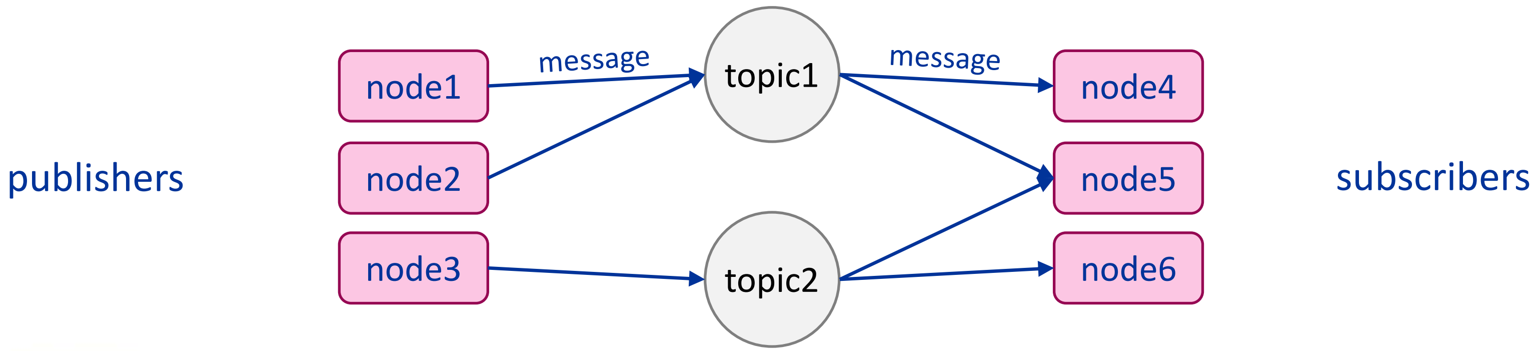
Discovery of nodes is automatically managed by ROS. When a node starts or stops running, ROS announces this node's information to other active nodes, allowing them to automatically establish data channels upon the node's initiation.

Nodes will only establish connections with other nodes if they have compatible Quality of Service settings.

Topic and Message

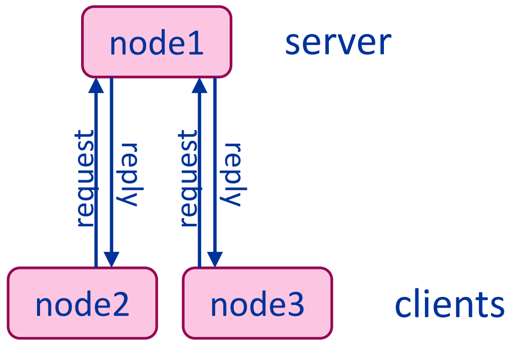
A Topic is a named channel over which nodes exchange data using a publish/subscribe model (i.e Observer pattern), while a Message is a structured data type used for communication between nodes, defining the type of information being transmitted over a topic channel.

NB: topics and messages are objects supported by ROS framework. They are not nodes.



Services

Services is a synchronous, request/response communication mechanism where one node (the client) sends a request to another node (the service) and waits for a response. Services are useful for tasks that require a reply, such as retrieving the state of a node or performing computations.



Parameters

Parameters are utilised for storing and managing configuration data. They enable nodes to store and access configuration values at runtime via a centralised Parameter Server. This system permits nodes to configure themselves and others based on dynamic information, avoiding the necessity for hardcoded values.

Parameters also utilise the Observer pattern; a node can subscribe to a specific parameter, and when that parameter changes, ROS will notify the node through a callback mechanism.

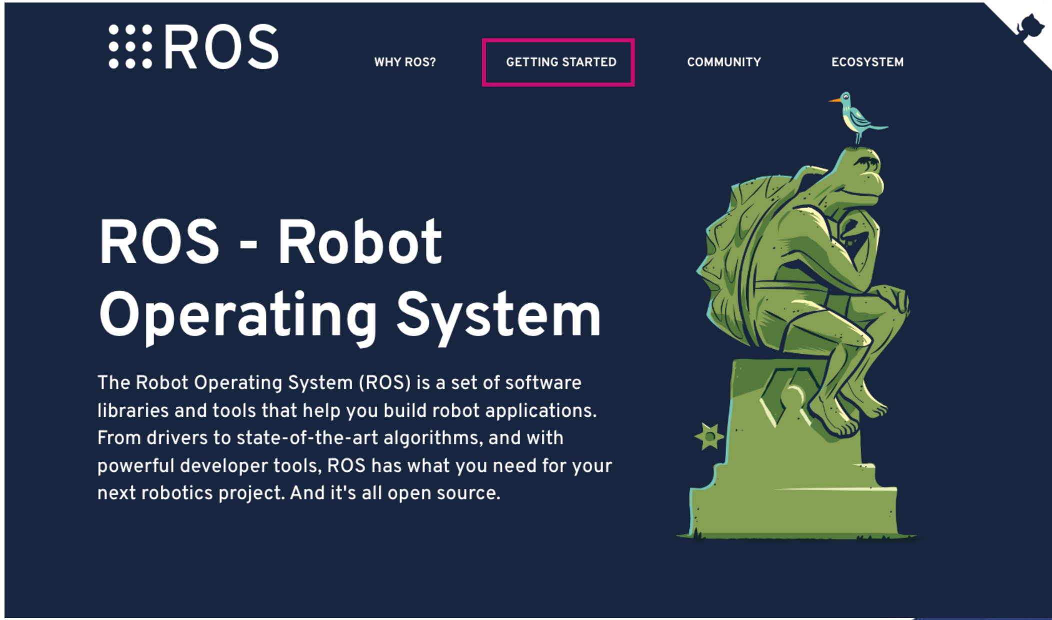
Embark on our journey with ROS

1495 UNIVERSITY OF ABERDEEN

Getting Started

The primary resource for learning ROS and obtaining support is the ROS official website: ros.org.

Initially, click on 'Getting Started' to find many resources tailored for different purposes.



ROS - Robot Operating System

The Robot Operating System (ROS) is a set of software libraries and tools that help you build robot applications. From drivers to state-of-the-art algorithms, and with powerful developer tools, ROS has what you need for your next robotics project. And it's all open source.

https://ros.org/

Primary ROS Resource

ROS Package Documentation

- Documentation for core ROS packages as well as package specific content is hosted on docs.ros.org. On this site you can find the core tutorials and documentation for the project as well as generated API documentation for individual packages.

Robotics Stack Exchange

- If the documentation doesn't address your problem, Robotics Stack Exchange is next.

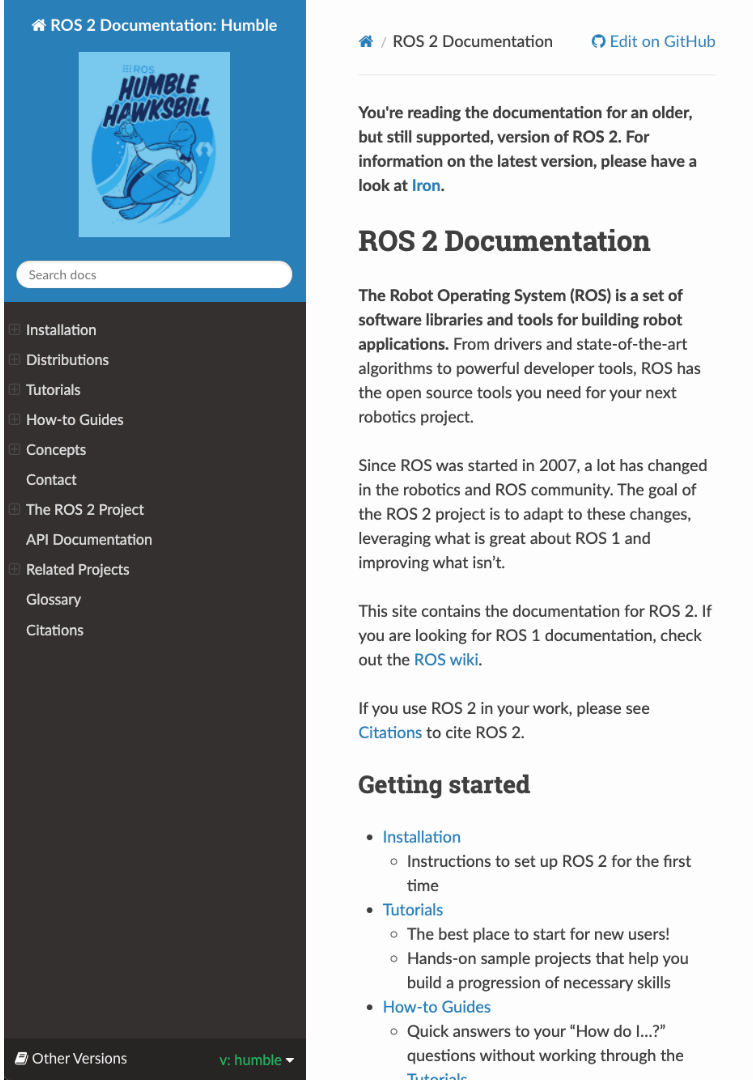
ROS Index

- When you want to find out information about a specific package the index is the best place to start. It connects you to all the important locations relevant to a package. In addition to the official documentation for ROS packages, the wiki contains two key resources you should consult: the Troubleshooting guide and the FAQ.

ROS Package Documentation

The ROS 2 documentation is immensely beneficial for beginners, offering comprehensive installation instructions and tutorials.

- Detailed installation guidance is provided for multiple systems (primarily focusing on Ubuntu and Windows, while the criteria for other systems are more stringent).
- The tutorials include an abundance of instructional content and examples catered to novices. They are invaluable resources for those embarking on their ROS 2 journey.



You're reading the documentation for an older, but still supported, version of ROS 2. For information on the latest version, please have a look at Iron.

ROS 2 Documentation

The Robot Operating System (ROS) is a set of software libraries and tools for building robot applications. From drivers and state-of-the-art algorithms to powerful developer tools, ROS has the open source tools you need for your next robotics project.

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

This site contains the documentation for ROS 2. If you are looking for ROS 1 documentation, check out the ROS wiki.

If you use ROS 2 in your work, please see Citations to cite ROS 2.

Getting started

- Installation
  - Instructions to set up ROS 2 for the first time
- Tutorials
  - The best place to start for new users!
  - Hands-on sample projects that help you build a progression of necessary skills
- How-to Guides
  - Quick answers to your "How do I...?" questions without working through the

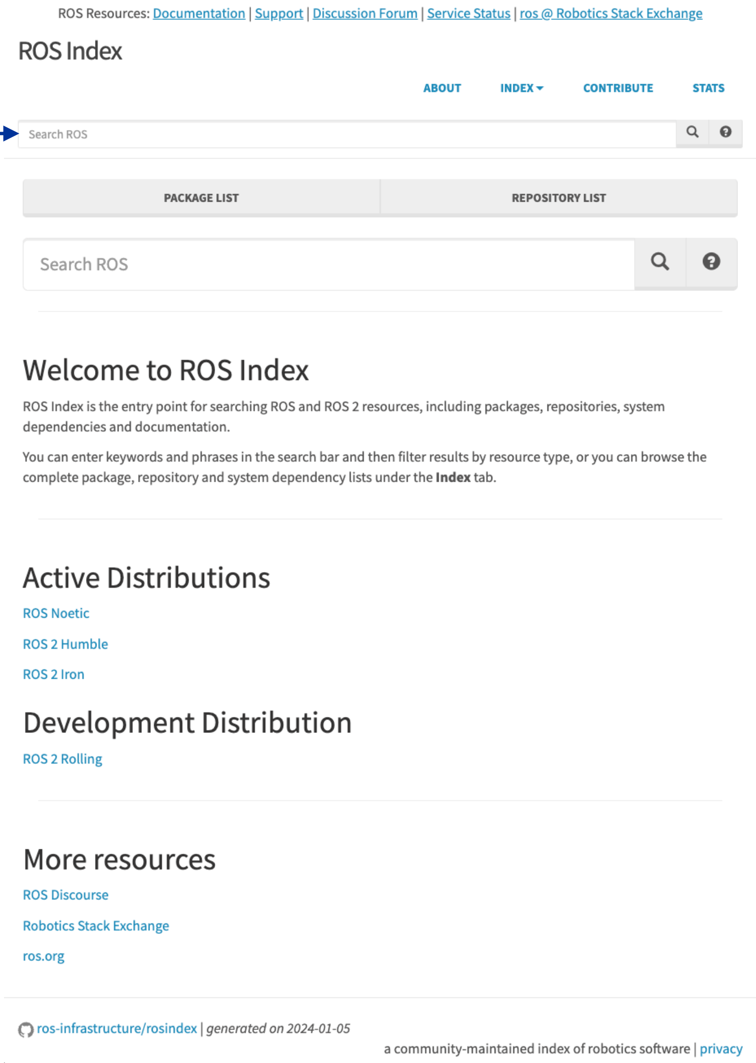
UNIVERSITY OF ABERDEEN

ROS Index

The ROS index contains all downloadable third-party packages and nodes, as well as their descriptions.

You can list the package/node name or search with their names.

E.g. we search 'turtlesim' a package name.



ROS Resources: Documentation | Support | Discussion Forum | Service Status | ros @ Robotics Stack Exchange

ROS Index

ABOUT INDEX CONTRIBUTE STATS

Search ROS

PACKAGE LIST | REPOSITORY LIST

Search ROS

Welcome to ROS Index

ROS Index is the entry point for searching ROS and ROS 2 resources, including packages, repositories, system dependencies and documentation.

You can enter keywords and phrases in the search bar and then filter results by resource type, or you can browse the complete package, repository and system dependency lists under the **Index** tab.

Active Distributions

ROS Noetic

ROS 2 Humble

ROS 2 Iron

Development Distribution

ROS 2 Rolling

More resources

ROS Discourse

Robotics Stack Exchange

ros.org

ros-infrastructure/rosindex | *generated on 2024-01-05*

a community-maintained index of robotics software | privacy

ROS Index

The search results are listed in a table.

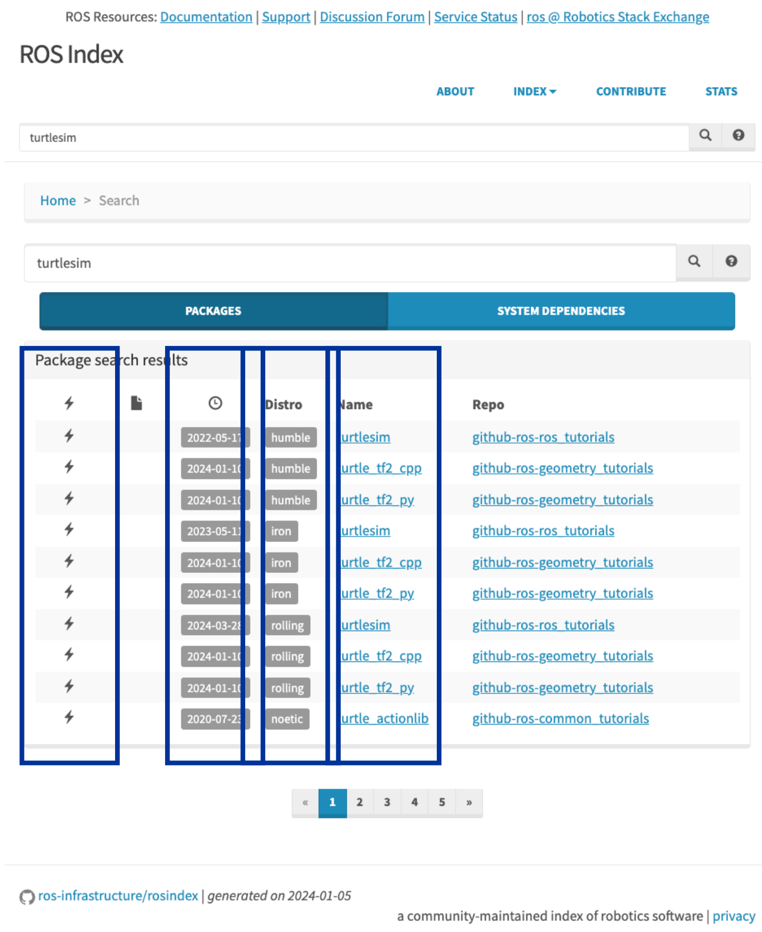
The ⚡ means the packages are published so you can download them.

The date are the last update date (including document update).

Distro refers to the target ROS version. You can find a package supports multiple ROS versions (but usually not all possible versions).

The names are the package names.

We don't always install the latest version of ROS, because it is difficult to get enough package supporting.



ROS Resources: Documentation | Support | Discussion Forum | Service Status | ros @ Robotics Stack Exchange

ROS Index

ABOUT INDEX CONTRIBUTE STATS

turtlesim

Home > Search

turtlesim

PACKAGES SYSTEM DEPENDENCIES

Package search results

⚡	📄	🕒	Distro	Name	Repo
⚡		2022-05-1	humble	urtlesim	github-ros-ros_tutorials
⚡		2024-01-1	humble	urtle_tf2_cpp	github-ros-geometry_tutorials
⚡		2024-01-1	humble	urtle_tf2_py	github-ros-geometry_tutorials
⚡		2023-05-1	iron	urtlesim	github-ros-ros_tutorials
⚡		2024-01-1	iron	urtle_tf2_cpp	github-ros-geometry_tutorials
⚡		2024-01-1	iron	urtle_tf2_py	github-ros-geometry_tutorials
⚡		2024-03-2	rolling	urtlesim	github-ros-ros_tutorials
⚡		2024-01-1	rolling	urtle_tf2_cpp	github-ros-geometry_tutorials
⚡		2024-01-1	rolling	urtle_tf2_py	github-ros-geometry_tutorials
⚡		2020-07-2	noetic	urtle_actionlib	github-ros-common_tutorials

« 1 2 3 4 5 »

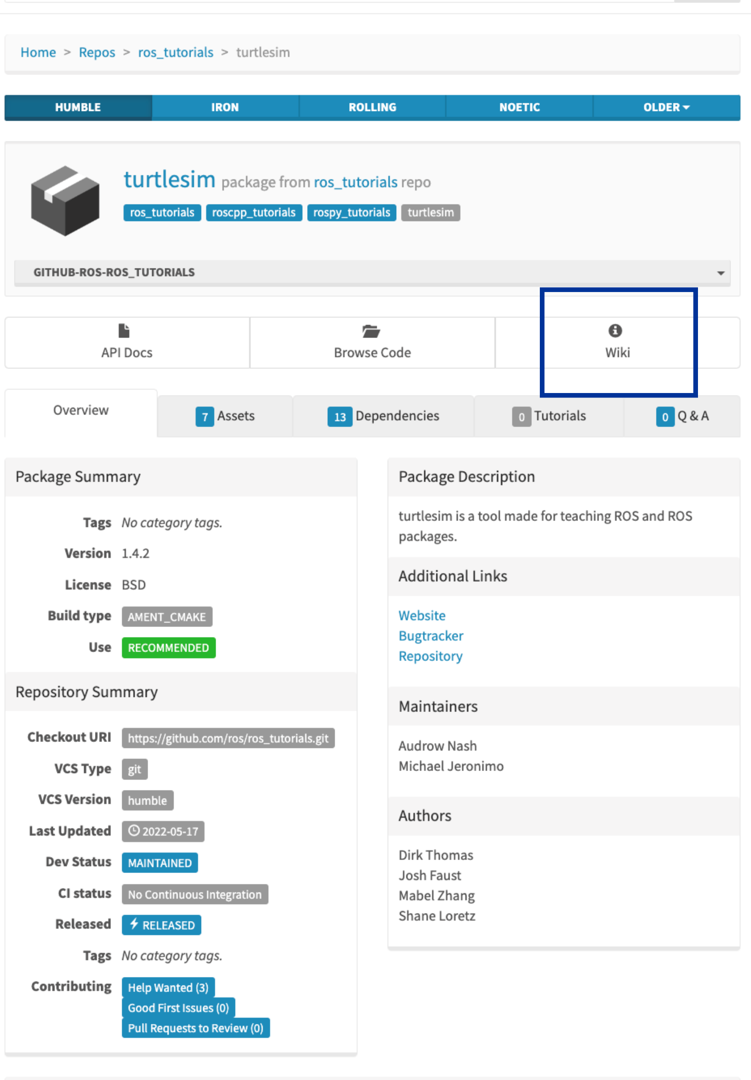
ros-infrastructure/rosindex | *generated on 2024-01-05*

a community-maintained index of robotics software | privacy

ROS Index

On the package page, you can find Package Summary, Repository Summary, and other information.

Most importantly, the wiki page contains specific instructions for using the package.



Home > Repos > ros_tutorials > turtlesim

turtlesim package from ros_tutorials repo

Package Summary

Tags	*No category tags.*
Version	1.4.2
License	BSD
Build type	AMENT_CMAKE
Use	RECOMMENDED

Repository Summary

Checkout URI	https://github.com/ros/ros_tutorials.git
VCS Type	git
VCS Version	humble
Last Updated	2022-05-17
Dev Status	MAINTAINED
CI status	No Continuous Integration
Released	RELEASED
Tags	*No category tags.*
Contributing	Help Wanted (3) Good First Issues (0) Pull Requests to Review (0)

Package Description

turtlesim is a tool made for teaching ROS and ROS packages.

Additional Links

Website
Bugtracker
Repository

Maintainers

Audrow Nash
Michael Jeronimo

Authors

Dirk Thomas
Josh Faust
Mabel Zhang
Shane Loretz

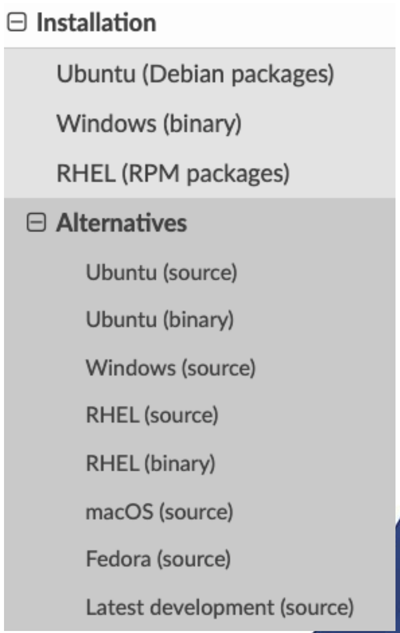
ROS2 Installation

The detailed installation description can be found in the ROS Package Documentation.

Although it supposes many OS, but mainly, people run ROS on Ubuntu. In the OS other than ubuntu, you will need to install Cross-platform supporting Apps (e.g. Qt5) to provide an ubuntu-compatible environment for ROS.

Here are links to the installation instructions for ROS2 Humble.

- Ubuntu: https://docs.ros.org/en/humble/Installation/Ubuntu-Install-Debians.html
- Windows: https://docs.ros.org/en/humble/Installation/Windows-Install-Binary.html



Run ROS 2 Nodes

ROS operations are mainly implemented through CLI (command line interface). We use the example of the little turtle that comes with ROS to show how to run ROS nodes.

First we open the command line, and then we need the ROS environment

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

in Windows, use PowerShell

```
C:\dev\ros2_humble\local_setup.ps1
```

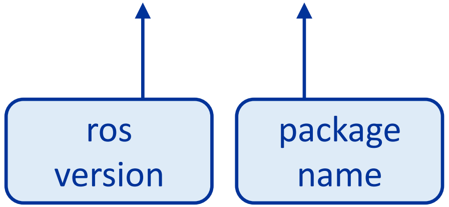
You will need to run this command on every new shell you open to have access to the ROS2. If you don't want always to do so, see https://docs.ros.org/en/humble/Tutorials/Beginner-CLI-Tools/Configuring-ROS2-Environment.html.

UNIVERSITY OF ABERDEEN

Run ROS 2 Nodes

In Ubuntu, we install turtlesim package with the following command:

```
sudo apt install ros-humble-turtlesim
```



In Windows, ROS integrates some commonly used packages, including turtlesim, so we can skip the installation step.

However, if you want to use a package that is not integrated with ROS, it will become very complicated. You need to download the source code of the package and compile it locally.

Run ROS 2 Nodes

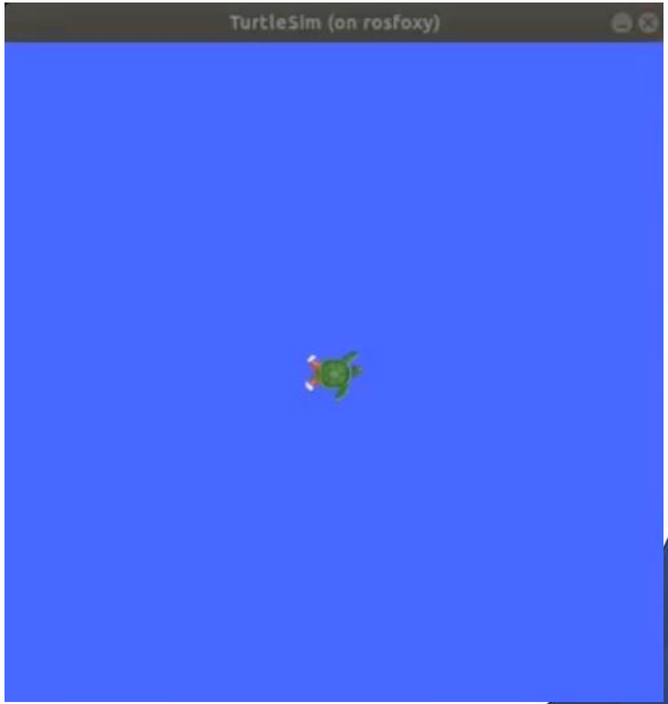
Then, to start turtlesim, enter the following command in your terminal:

```
ros2 run turtlesim turtlesim_node
```

package name

node name

You will lunch the right window.

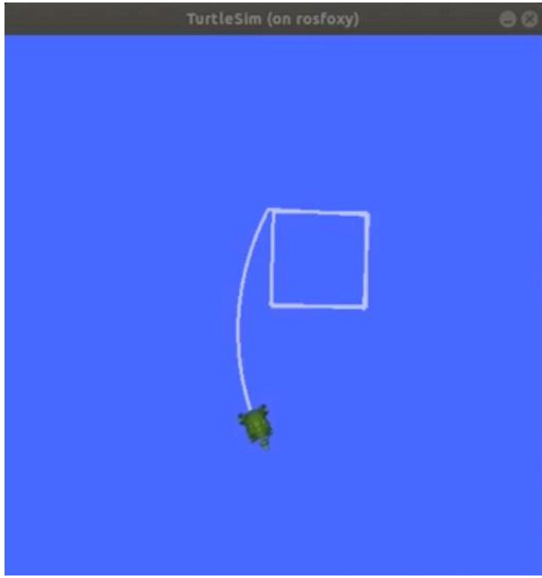


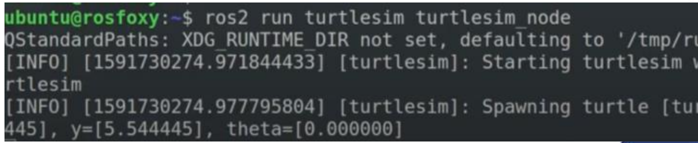
Run ROS 2 Nodes

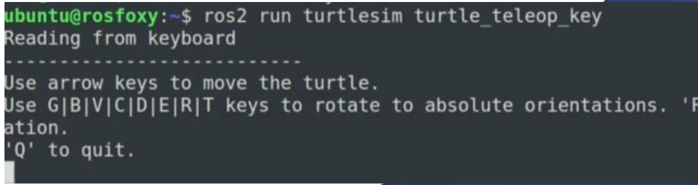
ROS supports multi-node collaboration. In order to control this little turtle, we need to run another control node. To do this, we will need to open a new CLI window again, and then you will run a new node to control the turtle in the first node:

```
ros2 run turtlesim turtle_teleop_key
```

At this point you should have three windows open: a terminal running turtlesim_node, a terminal running turtle_teleop_key and the turtlesim window. Arrange these windows so that you can see the turtlesim window, but also have the terminal running turtle_teleop_key active so that you can control the turtle in turtlesim.




```
ubuntu@rosfoxy:~$ ros2 run turtlesim turtlesim_node
QStandardPaths: XDG_RUNTIME_DIR not set, defaulting to '/tmp/r
[INFO] [1591730274.971844433] [turtlesim]: Starting turtlesim 
rtlesim
[INFO] [1591730274.977795804] [turtlesim]: Spawning turtle [tu
445], y=[5.544445], theta=[0.000000]
```


```
ubuntu@rosfoxy:~$ ros2 run turtlesim turtle_teleop_key
Reading from keyboard
---------------------------
Use arrow keys to move the turtle.
Use G|B|V|C|D|E|R|T keys to rotate to absolute orientations. 'F
ation.
'Q' to quit.
```

Develop You First Node

1495 UNIVERSITY OF ABERDEEN

Create Workspace Folder

A workspace is a folder containing ROS 2 packages.

A single workspace can contain as many packages as you want, each in their own folder. You can also have packages of different build types in one workspace (CMake, Python, etc.). You cannot have nested packages.

Best practice is to have a `src` folder within your workspace, and to create your packages in there. This keeps the top level of the workspace "clean".

So, our first step is to simply create workspace folder.

A trivial workspace might look like:

```
workspace_folder/
    src/
      cpp_package_1/
          CMakeLists.txt
          include/cpp_package_1/
          package.xml
          src/

      py_package_1/
          package.xml
          resource/py_package_1
          setup.cfg
          setup.py
          py_package_1/
      ...
      cpp_package_n/
          CMakeLists.txt
          include/cpp_package_n/
          package.xml
          src/
```

Create A Package

Make sure you are in `your_workspace/src/` folder.

Then, run the package creation command:

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

use python

package name

Then, it should generate a folder like:

```
py_hello_world/
        package.xml
        resource/py_hello_world
        setup.cfg
        setup.py
        py_hello_world/
```

Create Node File

In `py_hello_world/py_hello_world`, create a file `hello_world_node.py`.

Then, past the code.

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

class HelloWorldNode(Node):
    def __init__(self):
        super().__init__('hello_world_node')
        self.timer = self.create_timer(2, self.publish_hello_world)
        self.get_logger().info('Hello World Node has been started.')

    def publish_hello_world(self):
        self.get_logger().info('Hello, world!')

def main(args=None):
    rclpy.init(args=args)
    node = HelloWorldNode()
    try:
        rclpy.spin(node)
    except KeyboardInterrupt:
        pass
    finally:
        node.destroy_node()
        rclpy.shutdown()

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

1495 UNIVERSITY OF ABERDEEN

Update Setting

Add the following setting in the `entry_points` entry in `py_hello_world/setup.py`

```
entry_points={
    'console_scripts': [
        'hello_world = py_hello_world.hello_world_node:main',
    ],
},
```

Build The Node

Go back to your workspace folder: `your_workspace/`

Now you can build your packages:

```
colcon build
```

NB: this command build everything in your workspace.

Run The Node

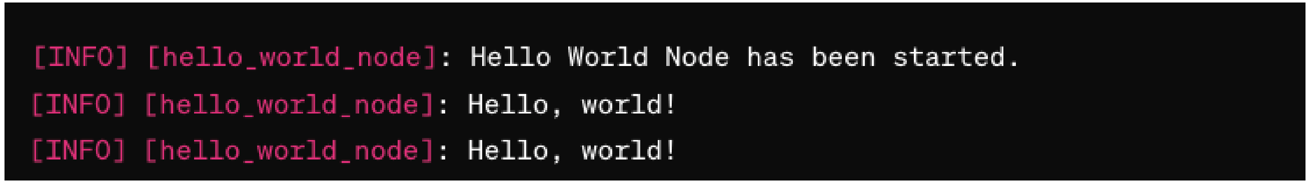
To use your new package and executable, first open a new terminal and source your main ROS 2 installation.

```
source install/local_setup.bash
```

To run the executable you created using the --node-name argument during package creation, enter the command:

```
ros2 run py_hello_world hello_world
```

The results are like:



```
[INFO] [hello_world_node]: Hello World Node has been started.
[INFO] [hello_world_node]: Hello, world!
[INFO] [hello_world_node]: Hello, world!
```

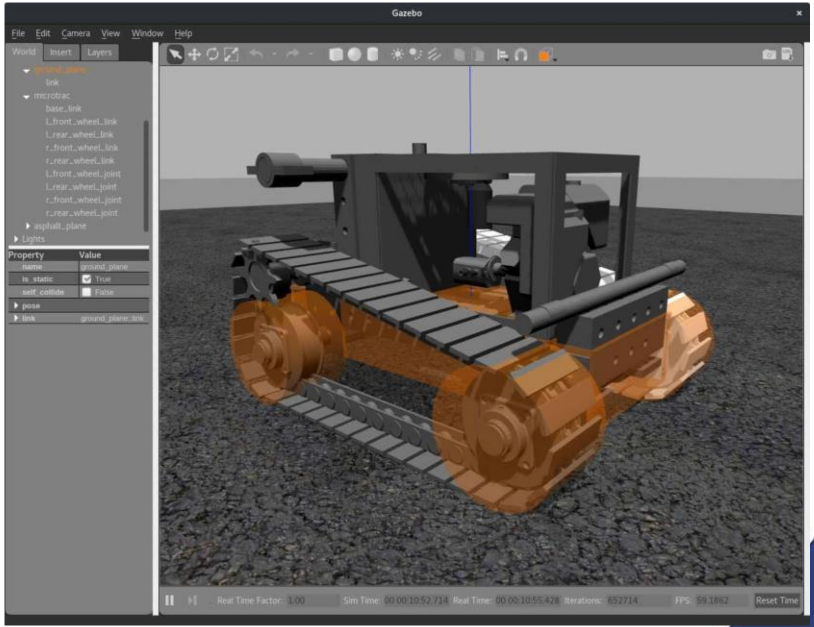
Gazebo

Gazebo

Gazebo is a sophisticated robotics simulation software that enables the design and testing of robots in complex indoor and outdoor environments.

Learning resource:

- https://gazebosim.org/
- https://gazebosim.org/docs/all/getstarted
- https://docs.ros.org/en/humble/Tutorials/Advanced/Simulators/Gazebo/Gazebo.html



Gazebo

<Video: 02 - gazebo overview.mp4>

Conclusion

We introduced the core concepts of ROS, including nodes, topics, services, message passing, and packages, all essential for building complex robotic systems.

Through a simple example, we demonstrated how to create and run a ROS node, a first step towards learning ROS.

Finally, ROS is more than just technology; it serves as a bridge between academia and industry, facilitating the rapid development and application of robotic technologies.

1495 UNIVERSITY OF ABERDEEN